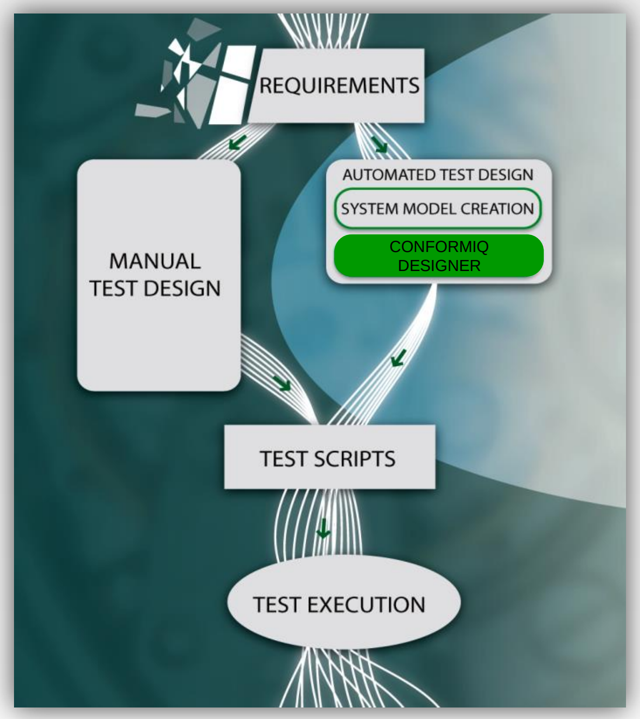
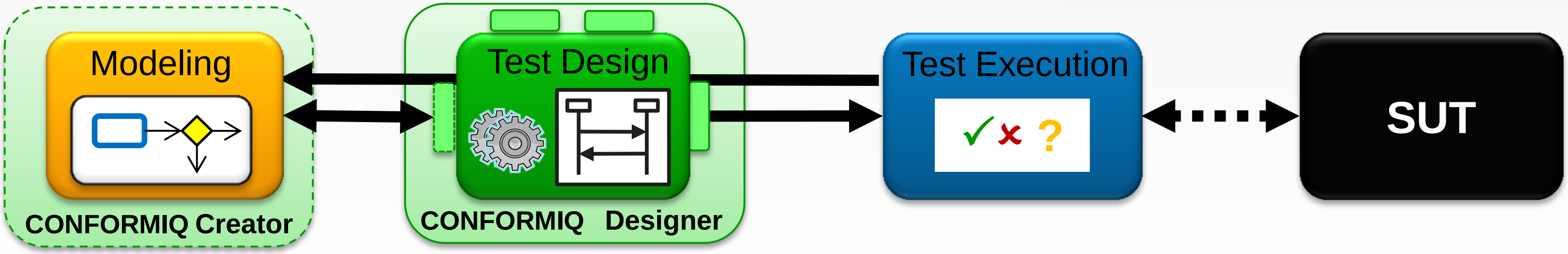
Getting Started with Conformiq Creator





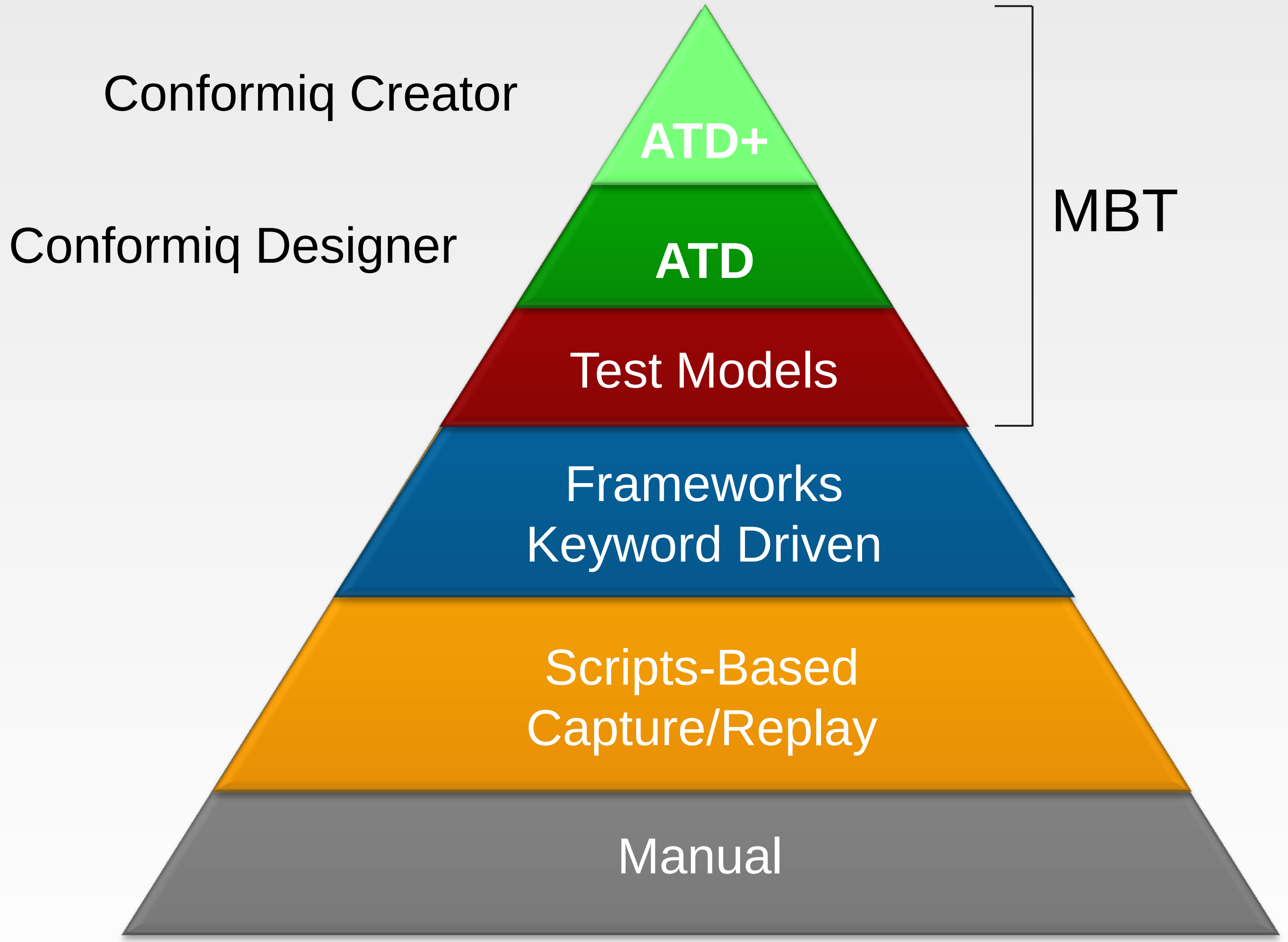
Contents

- **Introduction to Conformiq Creator**
- **Installing Conformiq Creator**
- **Working with Conformiq Creator (Theoretical)**
  - Part A:Modeling with Interface Diagrams
  - Part B:Modeling with Activity Diagrams
- **Getting Started with Conformiq Creator Examples**
- **Getting Started with your own Creator Project (Practical)**

Creator at a glance

Introduction to Conformiq Creator

Conformiq ATD+ Creator version



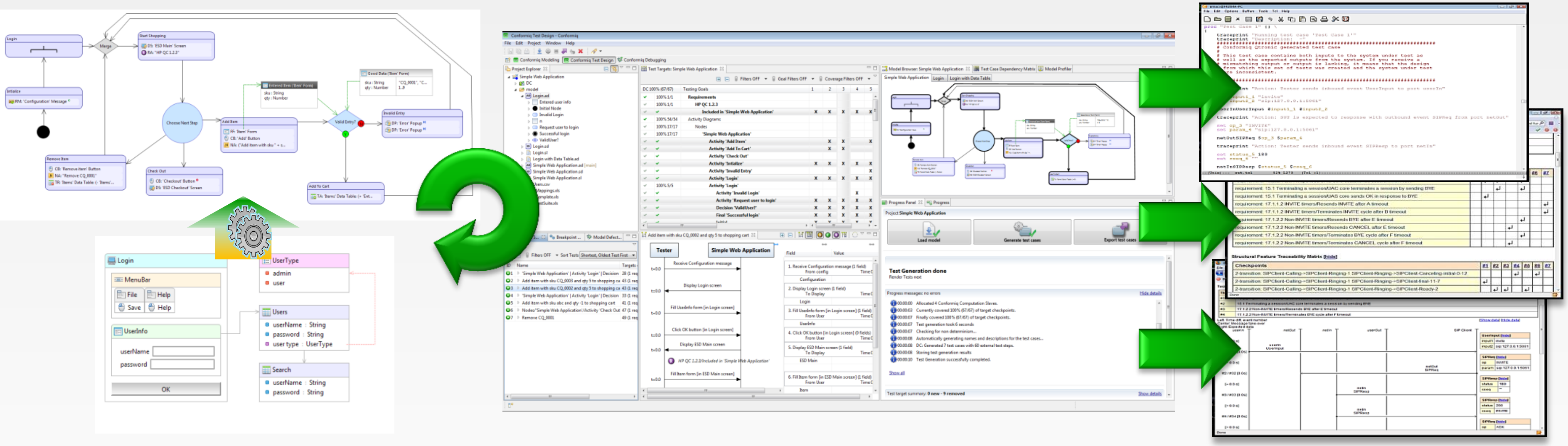
ATD = Automated Test Design

3 – Step ATA(MBT) Process with Conformiq Creator

Model
System Operation
with Creator

Direct & Review
Test Design

Generate Test Scripts
& Documentation



Iterative Approach

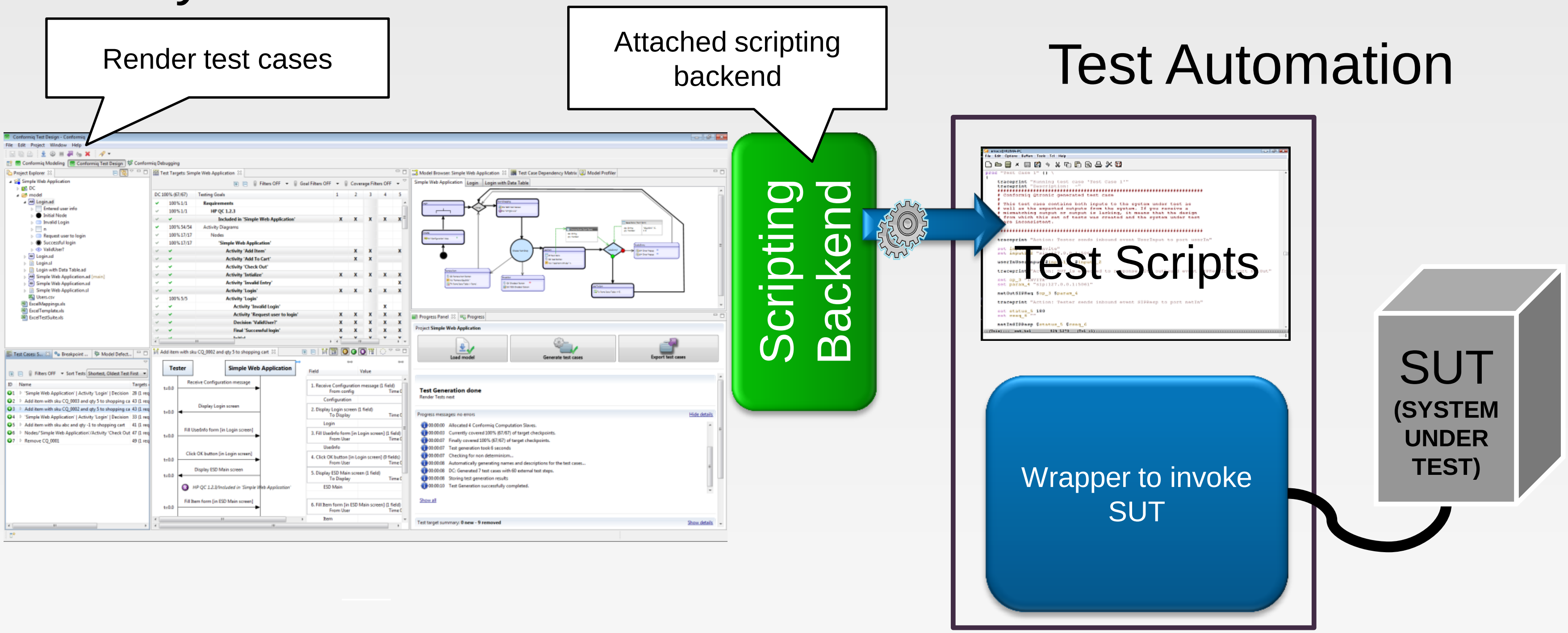
Eclipse based IDE

Publishing the outputs

Creator = Activity Diagrams + Actions (generated from Structure Diagrams)

MBT Architecture using Conformiq

Model system/functional behaviour



Render test cases

Attached scripting backend

Test Automation

Scripting Backend

Test Scripts

Wrapper to invoke SUT

SUT (SYSTEM UNDER TEST)

Conformiq Creator

Test harness

Test Design:

- Validate the model & Generate Test cases and execution scripts
- Validate requirement coverage
- Validate expected results (Pass/Fail)
- Enhance visibility of Functional flows

Auto Test Execution – Seamless Migration

- Tailored integration to the customer framework
- Linked to Test data & SUT
- Even automate progression testing!!

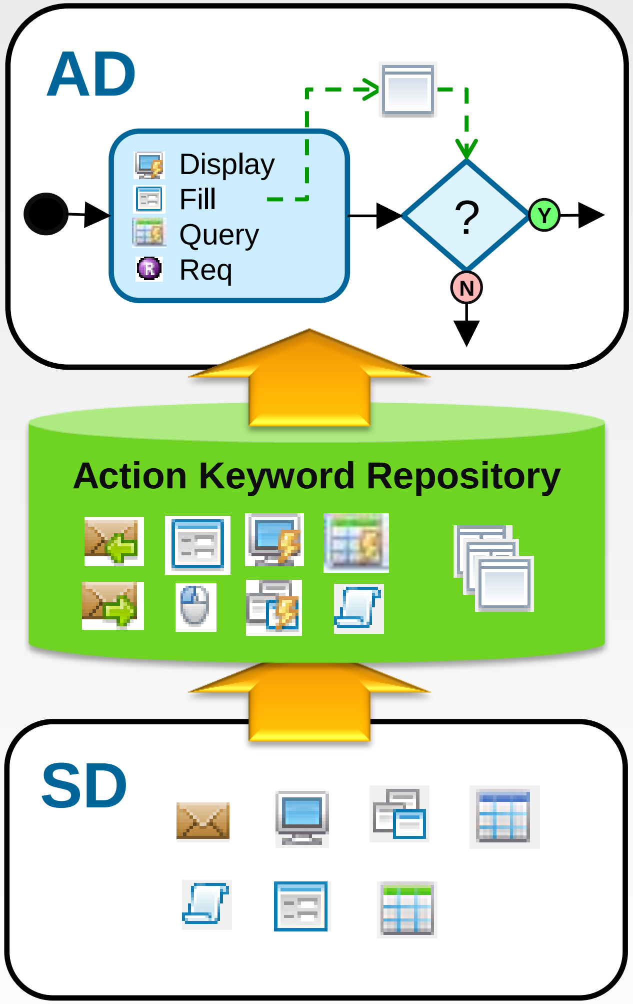
About Conformiq Creator

- Comes with a new modeling *paradigm* developed by Conformiq for creating models for test generation
  - Activity diagrams describe workflows (*what* to test)
  - Where actions that realize activities are specified based on a generated Action Keyword Repository (AKR) (*how* to test)
- New frontend for a proven test generation environment
  - Specifically developed for (but not limited to) using Conformiq in *system & user acceptance testing*
- Designed for
  - People without programming skills to specify, review and contribute to *modeling for test generation*
  - Domain specific customization of models
  - Deeper integration into existing tool chains
  - Further automation of the test process!

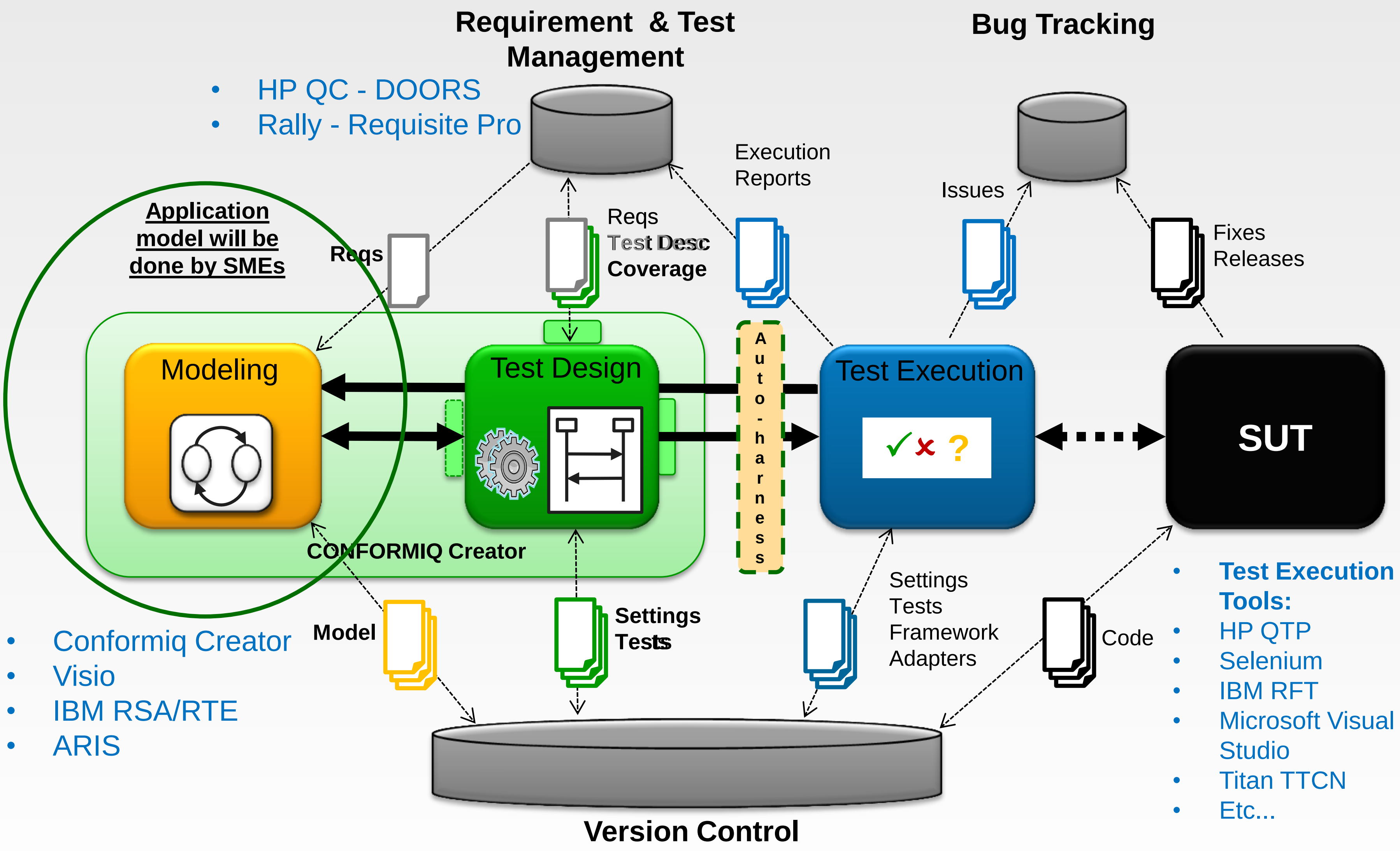
Productivity

Cornerstones for Modeling with Creator

- Activity Diagrams (AD)
  - Specify business/work flows using standard activity diagram symbols
  - Refine activities and decision based on action keywords and data objects from repository
- Action Keyword Repository
  - Action keywords and data objects generated from interface objects
- Structure Diagrams (SD)
  - Define external SUT interfaces available for testing based on predefined interface objects



ATA(MBT) in E2E – Solution Approach



- HP QC - DOORS
- Rally - Requisite Pro

- Conformiq Creator
- Visio
- IBM RSA/RTE
- ARIS

- **Test Execution Tools:**
- HP QTP
- Selenium
- IBM RFT
- Microsoft Visual Studio
- Titan TTCN
- Etc...

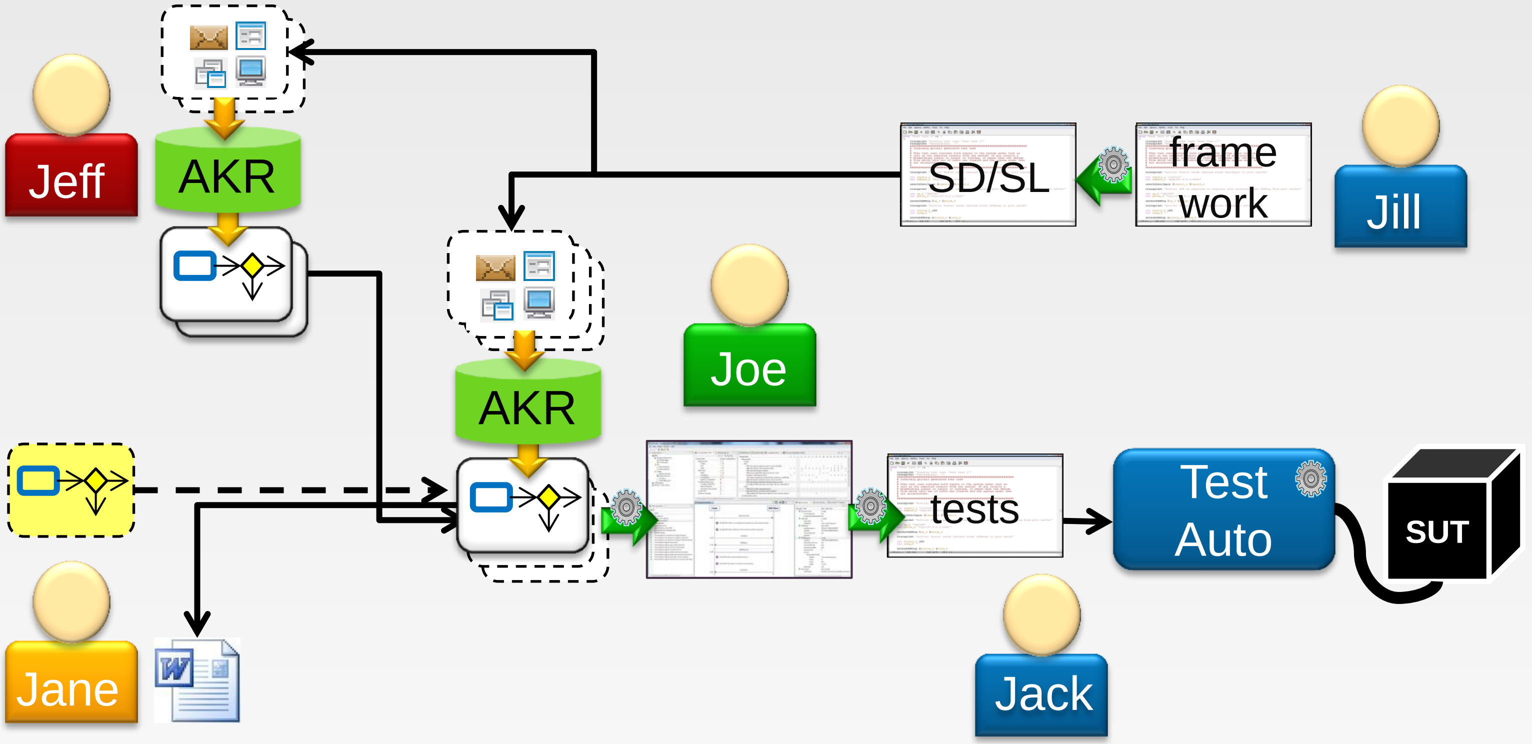
Key Features of Conformiq Creator

- Support for subset of standard activity diagram symbols
  - Mapping to UML activity diagrams
- Support for graphical data flow specification
- Support for modular model construction allowing independent development of isolated functional aspects
  - Enables composition of larger models from supplied parts
- Live check and guidance during model construction
- Support for domain specific SUT interface specification
  - Fully automatic generation of action keyword repository
  - Support for SUT interface change management
  - Import of interfaces from 3rd party (test execution) tools
  - Allows further automation of test harness implementation for user interface testing

Conformiq Creator Benefits

- Allows to *engage* and interact with stakeholders other than testers in directly model creation & review
  - Customers, system engineering, business analysts, etc.
  - Enables a completely new way of working!
- Access to Conformiq's unique and proven test generation engine!
  - Use of all existing coverage algorithms, criteria, analysis, etc.
  - Automatic exploitation of all data dependencies in system operation in test generation
- *Automates* model creation without programming
  - Offers domain specific actions and error checking as building blocks (e.g., login, form entry, sort, search, etc.)
  - Allows reuse and aligning with customer activity diagrams
  - Reuse interface descriptions, e.g., from existing testing harness
- Enables *generation* of test harnesses from structure diagrams

Use Scenario: Model Designer + CoE



- Jill (test automation) provides & maintains SUT interface information
- Jeff (CoE team) supports Joe with custom actions/activity libraries
- Joe models SUT specific functionality and reuses Jeff’s COTS
- Jane reviews Joe’s documentation generated from the model
- Joe & Jeff analyze own generated tests & Jack executes them

From Modeling to Validation to Test Execution
About Sharing, Versioning, and Working in Collaboration

Working with Conformiq Creator

Model Development with Conformiq Creator

1. Model or import *SUT interfaces* available for testing using **Structure diagram(s**) in Conformiq Modeling perspective
   - Can also be done in parallel with step 2

2. Model overall *SUT operation* from highest (business workflows) to lowest level (interface operation) by using standard (informal) **Activity diagram (s)**

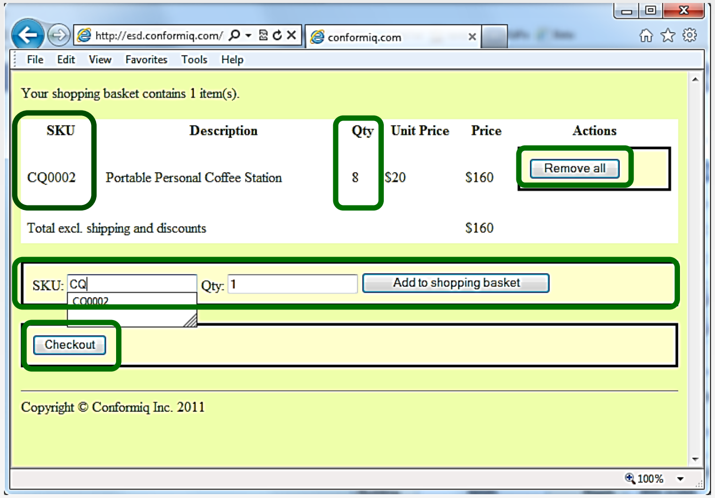
3. *Refine* your model by adding SUT interactions, data flows, and conditions based on generated Action Keyword Repository to reflect business rules

4. Validate your model by *peer review* and *reviewing tests* generated in Conformiq Test Design perspective

5. Render and upload reviewed tests for documentation and/or *test execution*

Modeling of Available SUT Interfaces

- Model all message based and user interfaces required and available to validate SUT operation
- Ideally reuse and leverage existing test harness information
  - Transform and import via XML
- Start (again) small by modeling only relevant interface information
  - Only messages and message fields *relevant* to system operation
  - Only screens, forms etc or widgets like buttons, table columns, text boxes etc *required* to realize workflows
  - Then extend interface definitions based on need



http://esd.conformiq.com/
conformiq.com
File Edit View Favorites Tools Help

Your shopping basket contains 1 item(s).

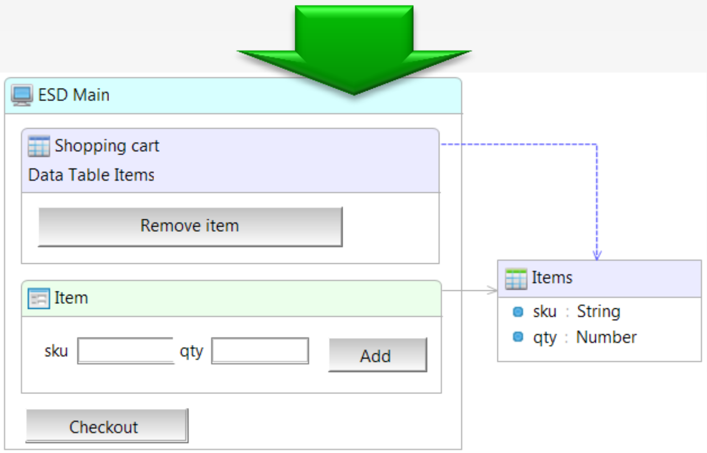
SKU	Description	Qty	Unit Price	Price	Actions
CQ0002	Portable Personal Coffee Station	8	$20	$160	Remove all
Total excl. shipping and discounts				$160	

SKU: CQ Qty: 1 Add to shopping basket

CQ0002

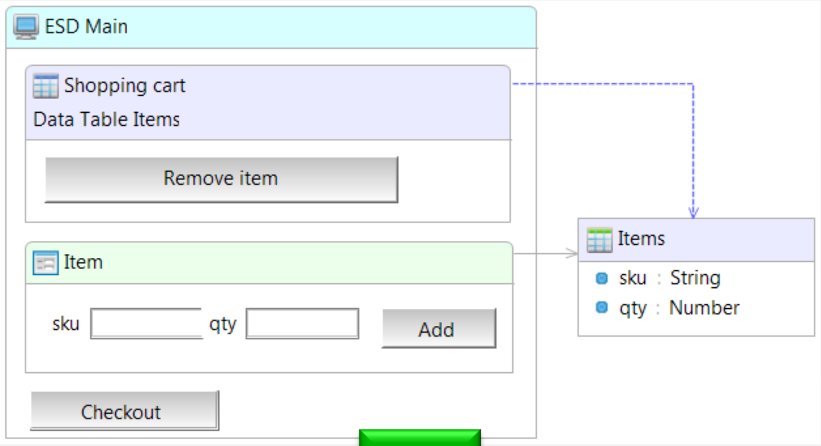
Checkout

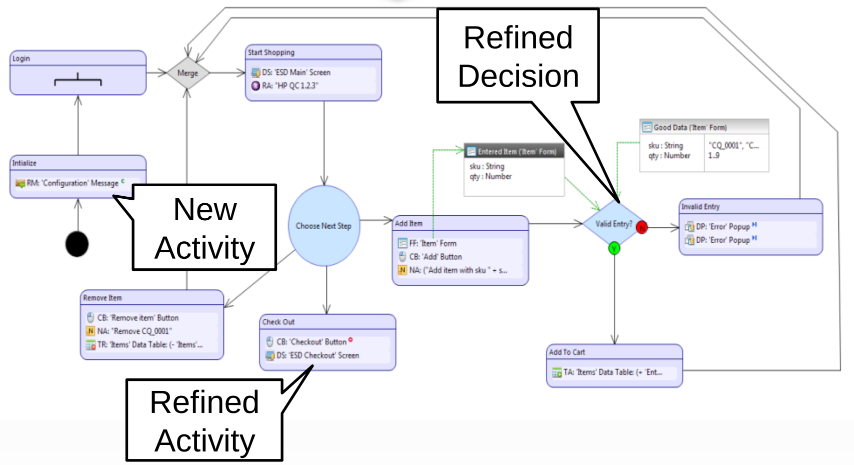
Copyright © Conformiq Inc. 2011



Refining SUT Operation for Test Generation

- Use action keywords and data objects generated from Structure Diagram(s)
- Extend existing flows or add application specific diagram layer if needed
- Add actions to activities
- Add data objects and data flow to control flow
- Add action pre-conditions and decision conditions





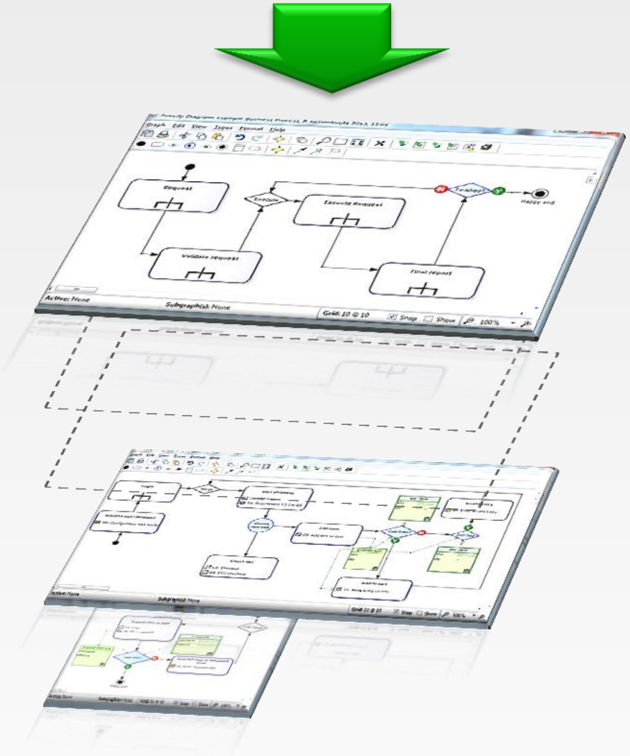
Refined Decision

New Activity

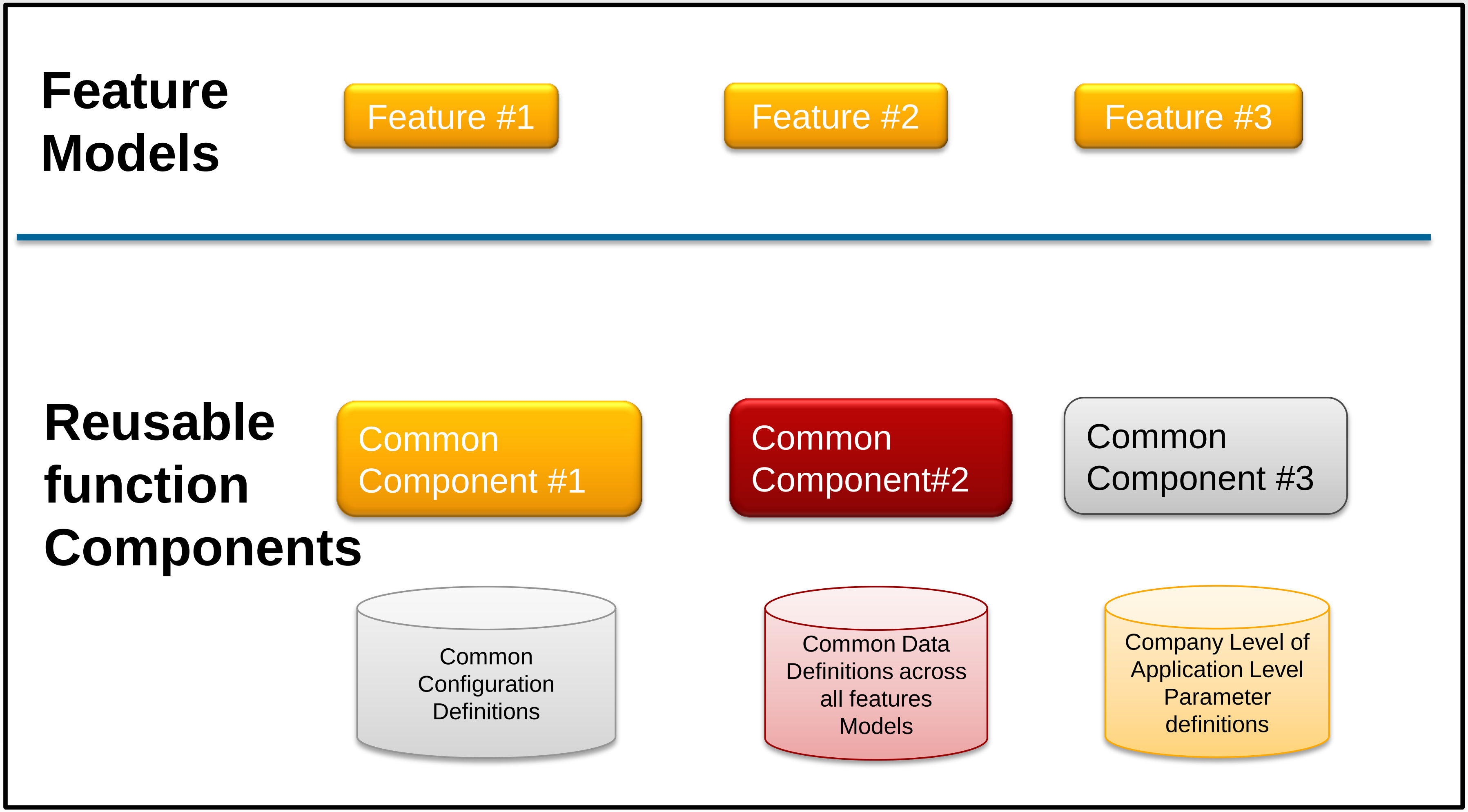
Refined Activity

Multi Level Modeling of Workflows

- Decompose SUT operation starting ideally from business processes
- Refine activities via one or more levels of sub diagrams to reflect actual operation of SUT or application under test
- Follow classic “pipe cleaner” style
  - Start with a few high level flows (i.e. a subset of functionality)
  - Then refine by adding more functionality to the different levels



Important aspect of MBT:
-Functional component Reusability approach



Feature Models

Feature #1

Feature #2

Feature #3

Reusable function Components

Common Component #1

Common Component#2

Common Component #3

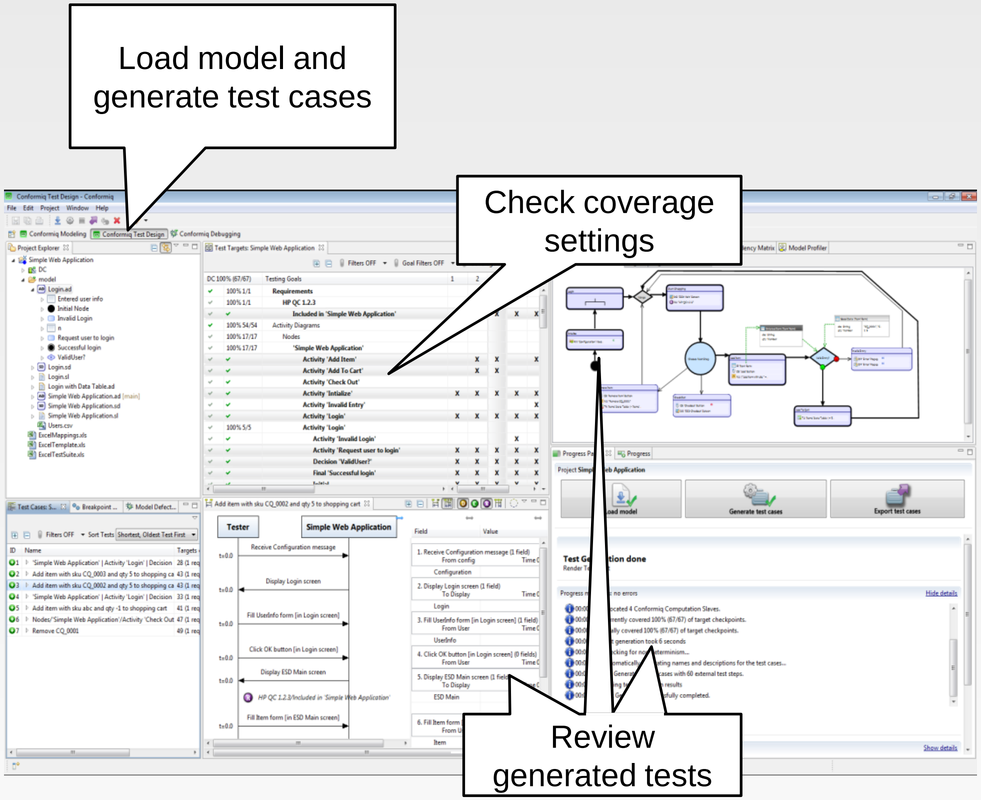
Common Configuration Definitions

Common Data Definitions across all features Models

Company Level of Application Level Parameter definitions

Model Validation by Test Review

- Self review by evaluating tests
  - Adjust default coverage settings in Coverage Editor (if needed)
  - Review generated tests in test case & test step views
  - Use traceability matrix to find & navigate between tests
- Peer review (with others)
  - Print or export models, e.g., HTML
  - Share model files with peers



Load model and generate test cases

Check coverage settings

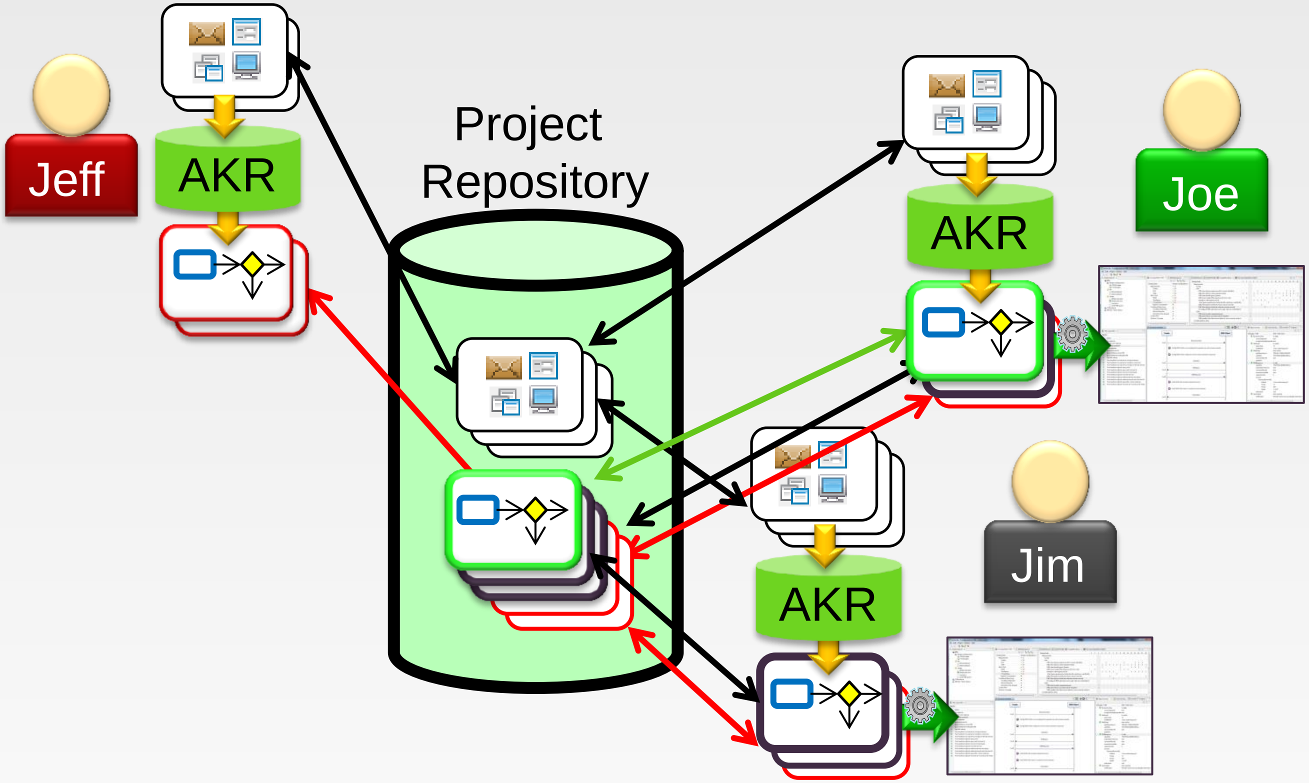
Review generated tests

From Generated Tests to Test Execution

1. Add one or more scripting backend(s) to your Conformiq project Design Configuration(DC)
2. Click the "Render tests" button in the tool bar
   - All tests generated at this point will be converted by all attached scripting backends according to their property settings
   - The output format depends on scripting backend, e.g., Excel Scripter produces spreadsheets for tests and traceability matrix, JUnit scripter produces Java files for tests, etc
3. Before executing generated test scripts a "generic test harness file" needs to be implemented (one time effort)
   - Implementation stubs are generated by scripting backends

Please consult the Conformiq Creator User Manual for more information

Use Scenario: Collaborative



- Jim (MBT Modeler) & Joe work in a huge project on different features
- Jim, Joe & Jeff diagrams via a common project repository
- Joe & Jim analyze generated tests only for their own features
- Jill, Jane and Jack are not shown here but may still be involved

Versioning and Sharing Creator Diagrams

- Diagrams in a project can be versioned as using standard version control software like svn, cvs, git, etc
  - Since diagram files are XML version them as binary
- Structure diagrams can be shared with other users or reused in other projects by copying or linking files
- Activity diagrams can be shared as files or for review by exporting them to a HTML file (or printing them)
  - Sharing of Activity diagrams as files *also* requires
    - Sharing of all relevant Structure diagram files (SD and SL), i.e., basis of actions used in these Activity diagrams
    - Sharing of all Activity diagrams referenced as subdiagrams

Installing Conformiq Creator

Installation Recommendation

- Installation And Hardware Requirements & OS:

Conformiq tool comes in client-server architecture where the client user interface is implemented as an Eclipse plugin. The server component — Conformiq Computation Server — can be installed on the same computer as the Conformiq Eclipse Client or on another computer on the same local area network.

Hardware Requirements and Recommendations

The Following Details the **strict requirements** for the Hardware, It lists 3 types of Installation and the corresponding requirements & recommendations:

1) Full Creator installation including both client and computation server (default)
2) The client only and
3) The server only.

Industrial deployments should be base on 64 bit architecture even if the tool chain operates on 32-bit platform.

	Full	Client only	Server only
Physical Memory (RAM)	4 GB	2 GB	4 GB
Processor	x86 family with 4 cores or 2 HT cores at 2GHz	x86 family with 2 cores or 1 HT core at 2GHz	x86 family with 4 cores or 2 HT cores at 2GHz
Hard disk (for installation only)	400 MB	200 MB	200 MB

where HT stands for “hyper threading”.

Hardware Requirements and Recommendations

The following details the **recommendations** for the hardware

	Full	Client only	Server only
Physical Memory (RAM)	> 16 GB	> 8 GB	> 16 GB
Processor	64 bit x86 family with 8 cores or 4 HT cores	64 bit x86 family with 4 cores or 2 HT cores	64 bit x86 family with 8 cores or 4 HT cores

A good rule of thumb is memory in GB is more than 2 times number of cores, so for example a machine with 8 cores should have at least 16 GB of physical memory.

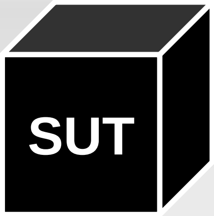
Software Requirements

- Conformiq Creator 2 installer
- Required Java environment for running Conformiq Eclipse Client (QEC) is Sun Java 6 or higher.
- Windows XP, Windows Vista, Windows 7 are supported.
- Windows distributions are provided both as a 32 and 64 bit installations.
- The 32 bit installation can be executed also in 64 bit environments
- It is highly recommended to use native 64 bit installation on 64 bit systems
- It is highly recommended to install SP3 or newer to Windows XP to take advantage of parallel test generation algorithm

Concepts & Examples

Part A:
Modeling with Structure Diagrams

The Structure Diagram SD



- Defines the SUT interfaces available for testing

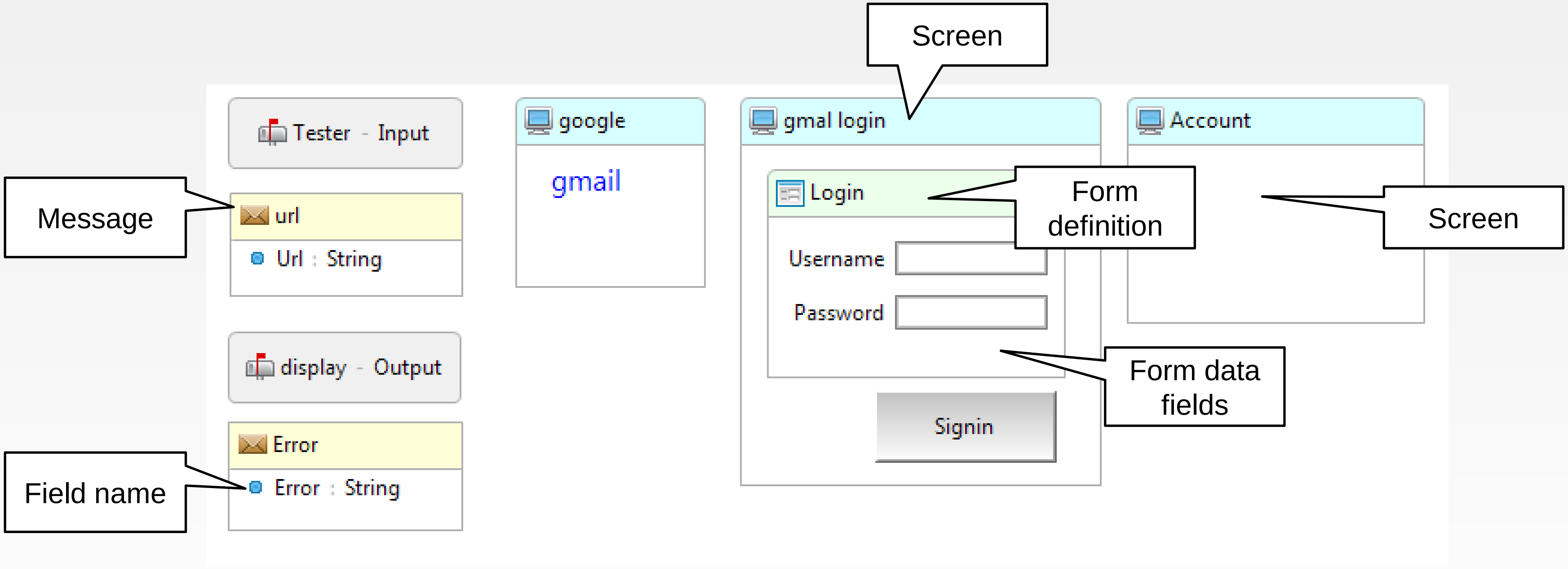
- Message concept for message based interfaces
  - Specifies interface(s) & message fields

- Screen, Popup, Form, Data Table concepts for UI Interfaces
  - Including input widgets like text box, dropdown, checkbox, etc
  - Including action widgets like button, link, tab, expander, menu, etc

- A generic concept of a data table
  - To manage Data Table content or model “data base like” SUT behavior
  - Has to be mapped to a form or message in an structure diagram to store received data, and to a Data Table to display contents
  - Can be initialized via a XLS/CSV file

Structure Diagram for Gmail Login

- ❑ Define external SUT interfaces available for testing based on predefined interface objects
- ❑ Considering Gmail Login Scenario
  - -Receive Error Message ,if username and password are incorrect
  - -Navigate to Account Page, if username and password are correct
    - -Not considering further flow



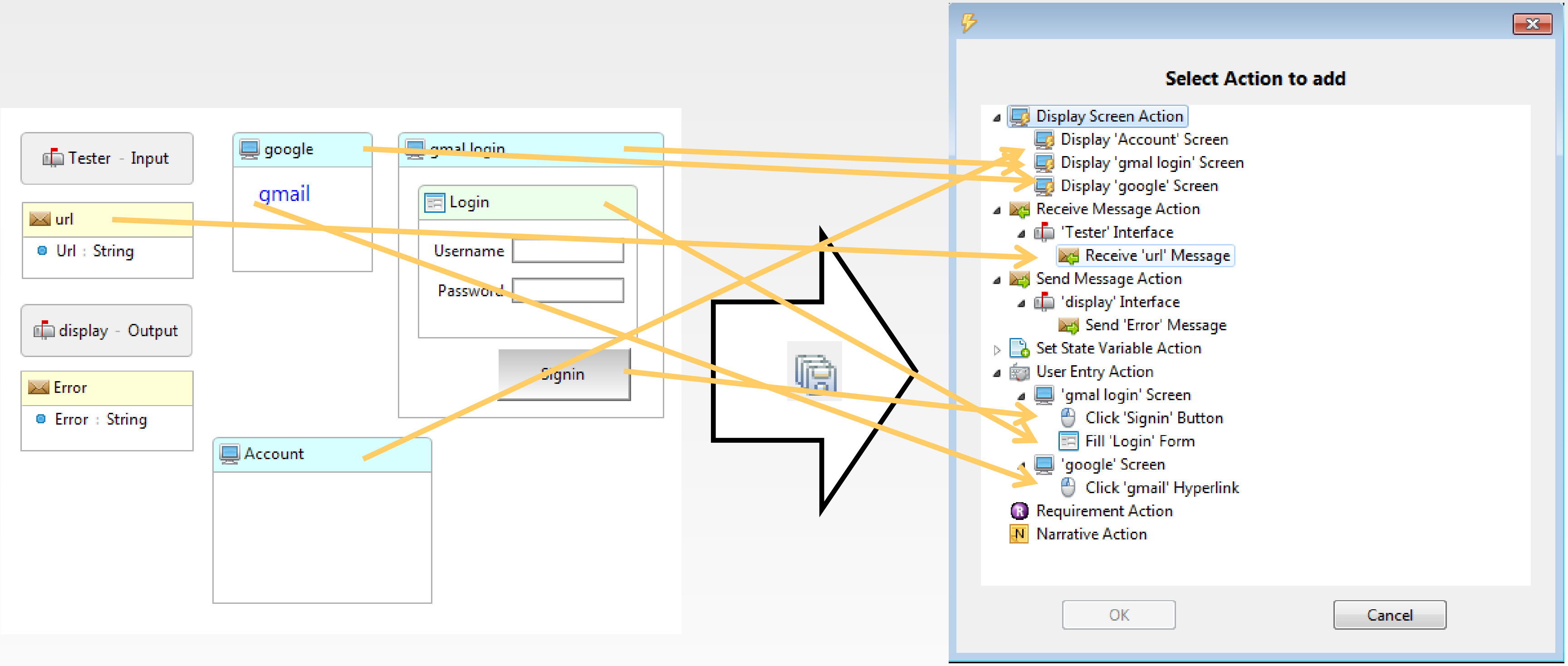
More on Structure Diagrams

- Interface objects can be defined across multiple diagrams
  - Referenced interface objects or objects with mappings (e.g., forms & tables) must be specified in same diagram
- Conformiq Creator includes a importer for WSDL/XSD files that automatically converts and creates a Structure Diagram with Message, Sequence, Choice and External Interface objects defined in selected WSDL/XSD(s)
- **Predefined action keywords and associated data objects are always generated from all structure diagrams in a project**
  - Requires that there is no errors in any Structure Diagram
  - In general one interface object is the source of <u>multiple</u> action keywords to the repository

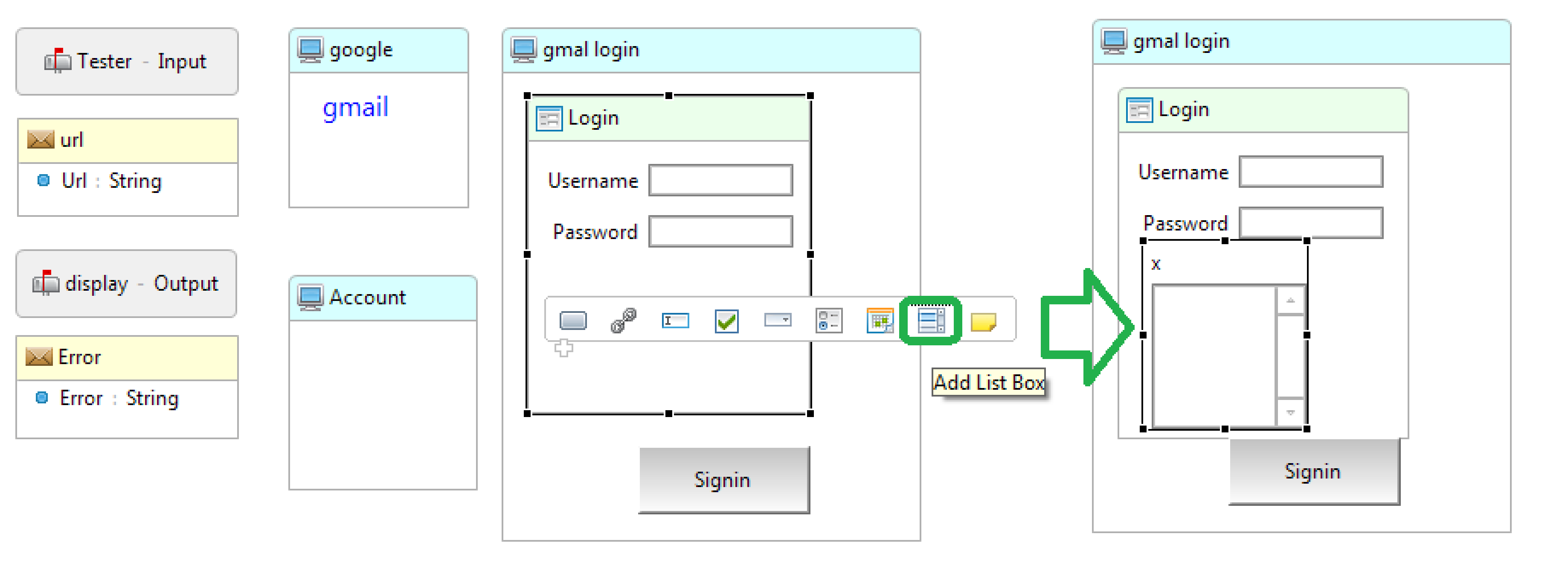
Action Keyword repository(AKR) Generation on Save

- Message, Form and Screen ID Example and Action Keywords

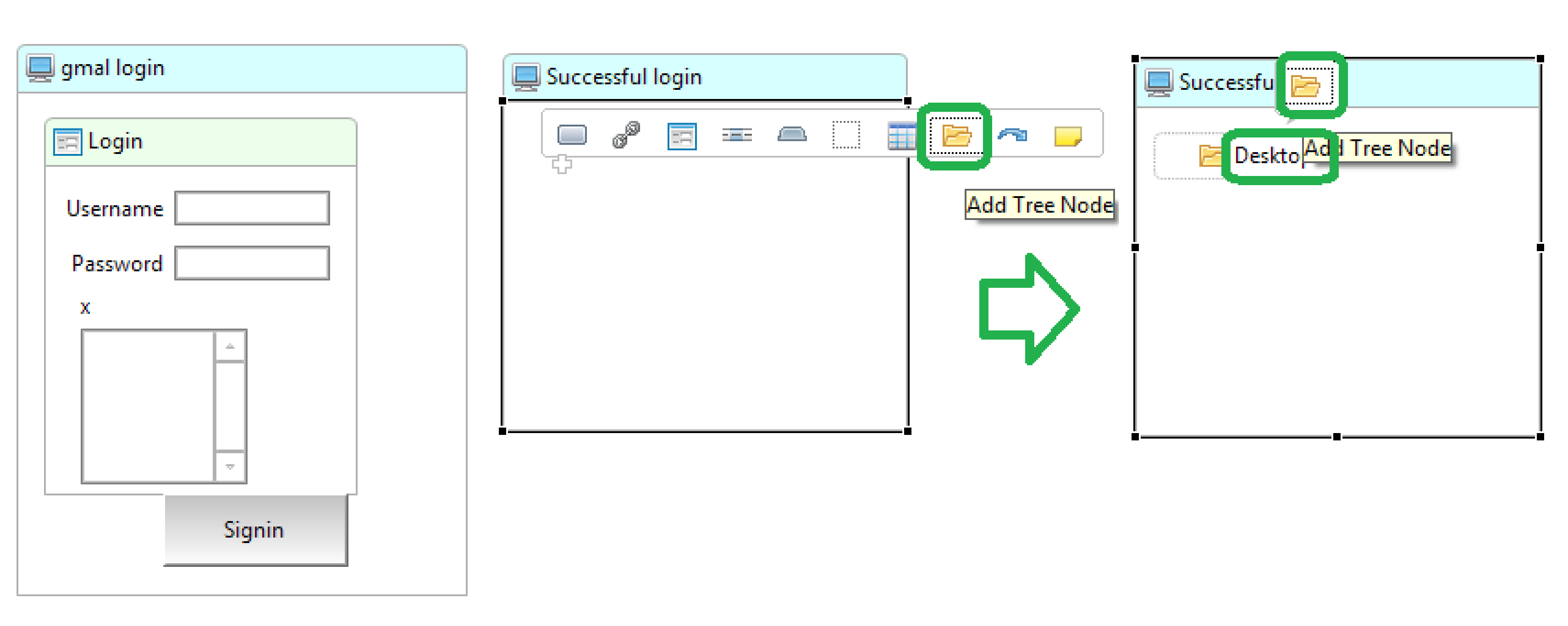
Generated Action Keywords



The Structure Diagram: Listbox widget in forms



The Structure Diagram: Specification of (tree) nodes in SDs as part of screen and popups



Concepts & Examples

Part B: Modeling with Activity Diagrams

Day-2 : FN

The Activity Diagram AD

- Defines the SUT functionality to be tested

Specifies abstract control flow aspects by using standard activity diagram symbols (*what* to test)

- Initial node, (sub)activity, final node, event, decision, merge, control flow

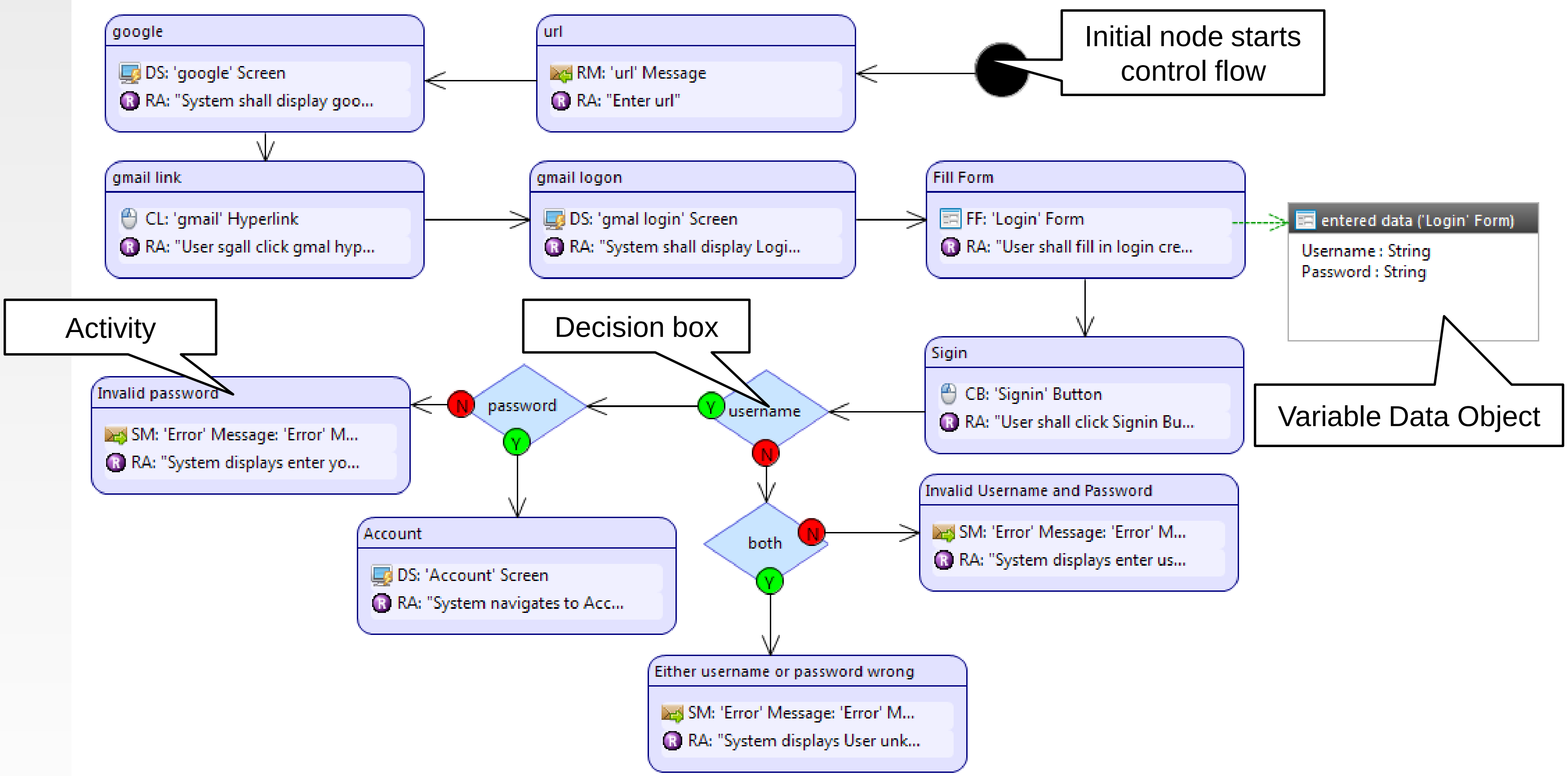
Activities and decision diamonds are refined with actions and conditions from the action repository (*how* to test)

- Action are specified from generated action keywords and can include pre-conditions, constraints on received data, and storing or production of data
- Conditions are specified based on generated data objects including variables and different kinds of values

Data flow aspects can be specified graphically with generated data objects and data flows

Activity flow for Gmail Login

(“*what* to test ”+”*how* to test”)



Initial node starts control flow

url
RM: 'url' Message
RA: "Enter url"

google
DS: 'google' Screen
RA: "System shall display goo...

gmail link
CL: 'gmail' Hyperlink
RA: "User sgall click gmal hyp...

gmail logon
DS: 'gmal login' Screen
RA: "System shall display Logi...

Fill Form
FF: 'Login' Form
RA: "User shall fill in login cre...

entered data ('Login' Form)
Username : String
Password : String

Variable Data Object

Sigin
CB: 'Signin' Button
RA: "User shall click Signin Bu...

Decision box

username
Y
N

password
N
Y

Activity

Invalid password
SM: 'Error' Message: 'Error' M...
RA: "System displays enter yo...

Account
DS: 'Account' Screen
RA: "System navigates to Acc...

both
N
Y

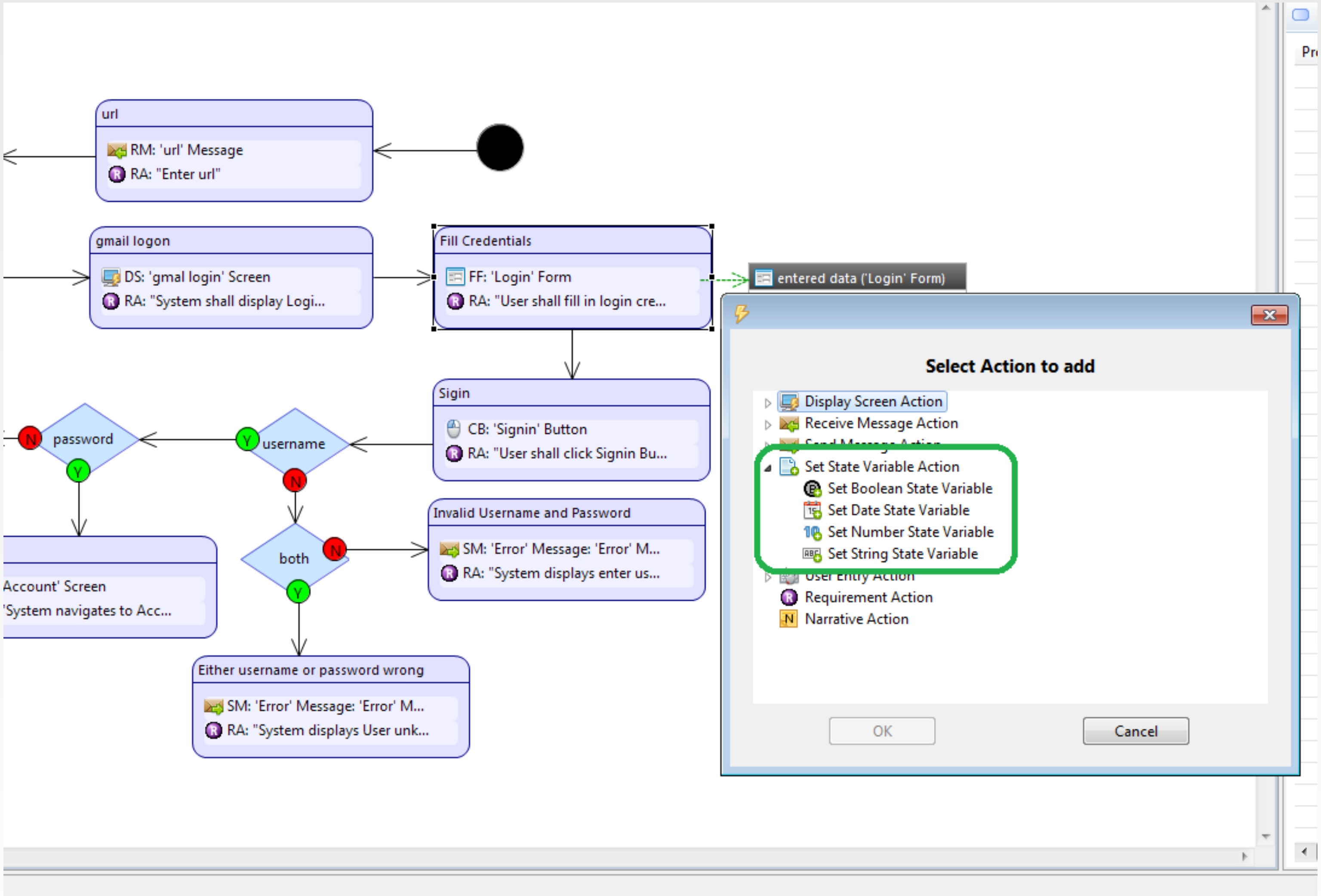
Invalid Username and Password
SM: 'Error' Message: 'Error' M...
RA: "System displays enter us...

Either username or password wrong
SM: 'Error' Message: 'Error' M...
RA: "System displays User unk...

The Activity Diagram: Additional features

- Set State Variable action to modify values of state variables.
- Ability to define state variables of type string, Boolean, Number & Date in Activity Diagrams

The Activity Diagram: State Variables

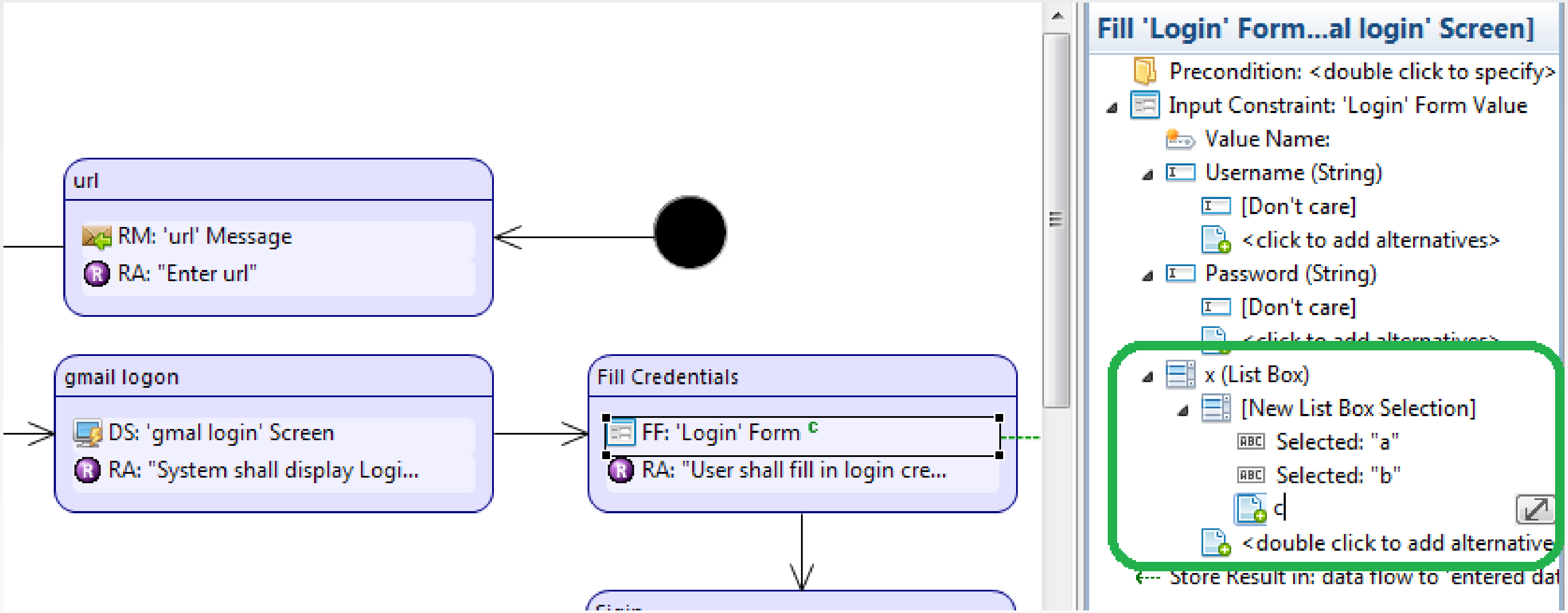


url
RM: 'url' Message
RA: "Enter url"
gmail logon
DS: 'gmal login' Screen
RA: "System shall display Logi...
Fill Credentials
FF: 'Login' Form
RA: "User shall fill in login cre...
entered data ('Login' Form)
Sigin
CB: 'Signin' Button
RA: "User shall click Signin Bu...
password
username
both
N
Y
Invalid Username and Password
SM: 'Error' Message: 'Error' M...
RA: "System displays enter us...
Account' Screen
'System navigates to Acc...
Either username or password wrong
SM: 'Error' Message: 'Error' M...
RA: "System displays User unk...
Select Action to add
Display Screen Action
Receive Message Action
Send Message Action
Set State Variable Action
Set Boolean State Variable
Set Date State Variable
Set Number State Variable
Set String State Variable
User Entry Action
Requirement Action
Narrative Action
OK
Cancel

The Activity Diagram: List box widgets (Forms)

- Users can specify list boxes as part of forms. Please note that when specifying data in conditions or constraints one (specific) list box value models all the selected fields (which maybe none, one or more). Alternative values specify a list of possible selections. Note also that using listboxes with different Data Coverage setting _*requires*_ the explicit enumeration of selection of interest using alternative values in a Form Value, i.e., “all (pairwise) combinations” of possible selections are not automatically computed (since Listbox items are not a static list but may change dynamically.

The Activity Diagram: List box widgets (Forms)



Fill 'Login' Form...al login' Screen]

- Precondition: <double click to specify>
- Input Constraint: 'Login' Form Value
  - Value Name:
  - Username (String)
    - [Don't care]
    - <click to add alternatives>
  - Password (String)
    - [Don't care]
    - <click to add alternatives>
  - x (List Box)
    - [New List Box Selection]
      - Selected: "a"
      - Selected: "b"
      - c
    - <double click to add alternative
- Store Result in: data flow to 'entered dat

Concepts & Examples

Modeling Data

More about Generated Data Objects

- Data objects can be used
  - To specify constraints on or store incoming data in input actions
  - To specify outgoing data to be used in send or display actions
  - To specify the condition that is part of a decision
  - To create graphical data flows
- During action keyword repository generation data objects are generated for actions that can consume or produce data
  - Based on interface object type Conformiq Creator generates either specific message, form, and/or table entry data objects
  - Each specific data object can either be a value, value list, or variable data object

About Data Tables

- Data tables have to be specified in structure diagrams
  - Generally useful for modeling the impact of pre-configured or received data on SUT operation (i.e., internal data management)
  - Have to be used to model the data content of tables shown in a user interface (i.e., UI tables)
- During repository generation add, query, remove, modify action keywords for each data table are generated
  - When adding table entries using the inline approach, form/message to entry transformation functions have to be used
  - “Is Data Table Query Result Valid?” Boolean function returns false if no match to a query could be found.
  - Queries return (only) the first item matching to a query
- A data table can be initialized via a XLS and CSV file

Data Object Value Types

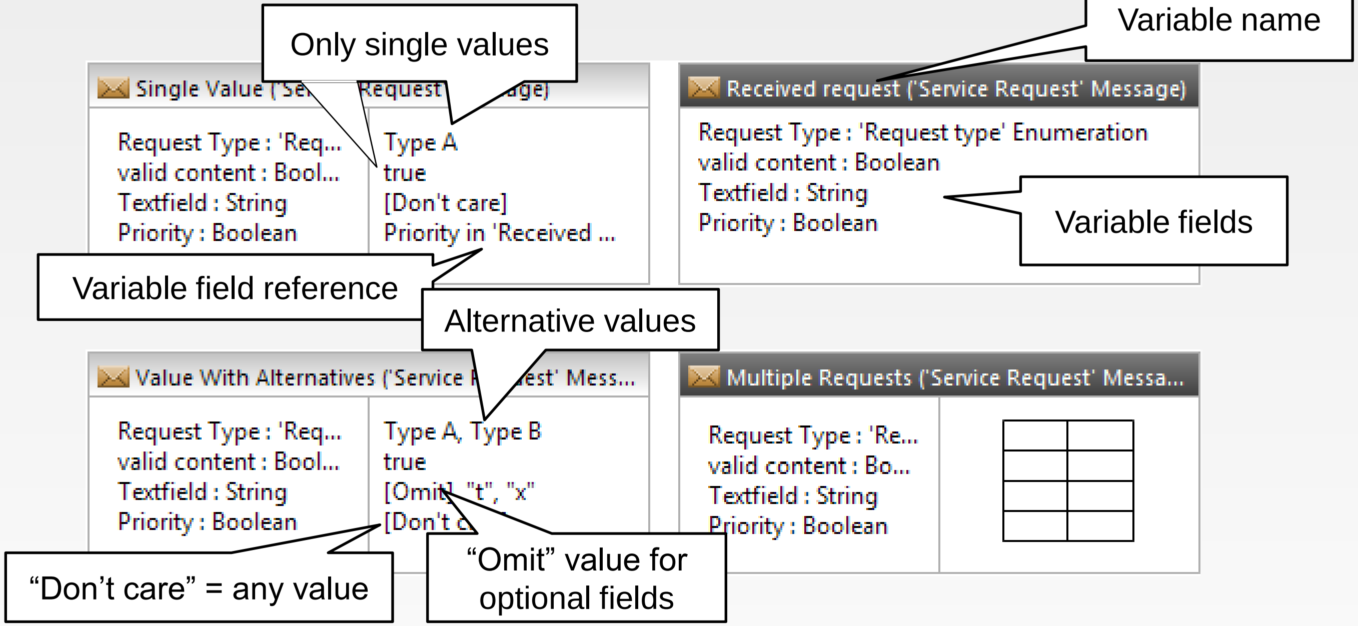
- Value: **<field> EQUAL 1**
  - One specific value (or reference to variable field value) is specified for every field in a Value data object (“Don’t care” = ignore field)
  - Can be be used in verification actions like “Send Message” or “Display Screen” or “Add Table” but also in constraint or condition specifications
- Alternative Value: **<field> EQUAL 1,2,3 or <field> EQUAL 1 .. 3**
  - More than one value is specified for at least one field within a Value data object
  - Best used with String, Enumerated, or Number fields in conditions or constraints
  - Basically short hand “<field> EQUAL 1 OR <field> EQUAL 2 …”
  - Alternative values can not be used in verification actions!

Data Object Types

- Example: Service Request Message

(Single) Value

Variable



Only single values

Single Value ('Se… Request …age)

Request Type : 'Req… | Type A
valid content : Bool… | true
Textfield : String | [Don't care]
Priority : Boolean | Priority in 'Received …

Variable field reference

Variable name

Received request ('Service Request' Message)

Request Type : 'Request type' Enumeration
valid content : Boolean
Textfield : String
Priority : Boolean

Variable fields

Alternative values

Value With Alternatives ('Service R…est' Mess…

Request Type : 'Req… | Type A, Type B
valid content : Bool… | true
Textfield : String | [Omit], "t", "x"
Priority : Boolean | [Don't c…

"Don't care" = any value

"Omit" value for optional fields

Multiple Requests ('Service Request' Messa…

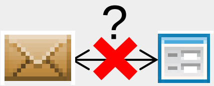
Request Type : 'Re…
valid content : Bo…
Textfield : String
Priority : Boolean

Value with Alternatives

Value List

- Form and table entry values are specified in same way

More About Using Data Objects

- In general values or variables of different data objects can not be directly compared or assigned
  - A message can not be compared against a Form 
  - A “XYZ Message” can not be compared against a “ABC Message” or used in a “Send ABC Message” action

Exception: Message and form objects can be used in “Add Entry” actions **if** they have a mapping in an Structure Diagram

- Fields of different data object types can always be directly compared or assigned if they are of same type
  - String field “a” of “XYZ Message can be compared or assigned to string field “b” of “ABC Message”
- The Creator editor automatically enforces these rules

Conditions

- As shown earlier conditions are the basis for making a decision but also to specify pre-conditions in actions
- A condition compares two data objects against each other based on a logic operator
  - A variable (or variable field) against a variable (or variable field)
  - A variable (or variable field) against a value
- Conditions are one of the most important constructs in the Creator modeling language but have to be used with care
  - They enable Conformiq's test generation engine to explore and generate tests covering all possible scenarios
  - At the same time conflicting conditions (or constraints) in different model parts can prevent test generation

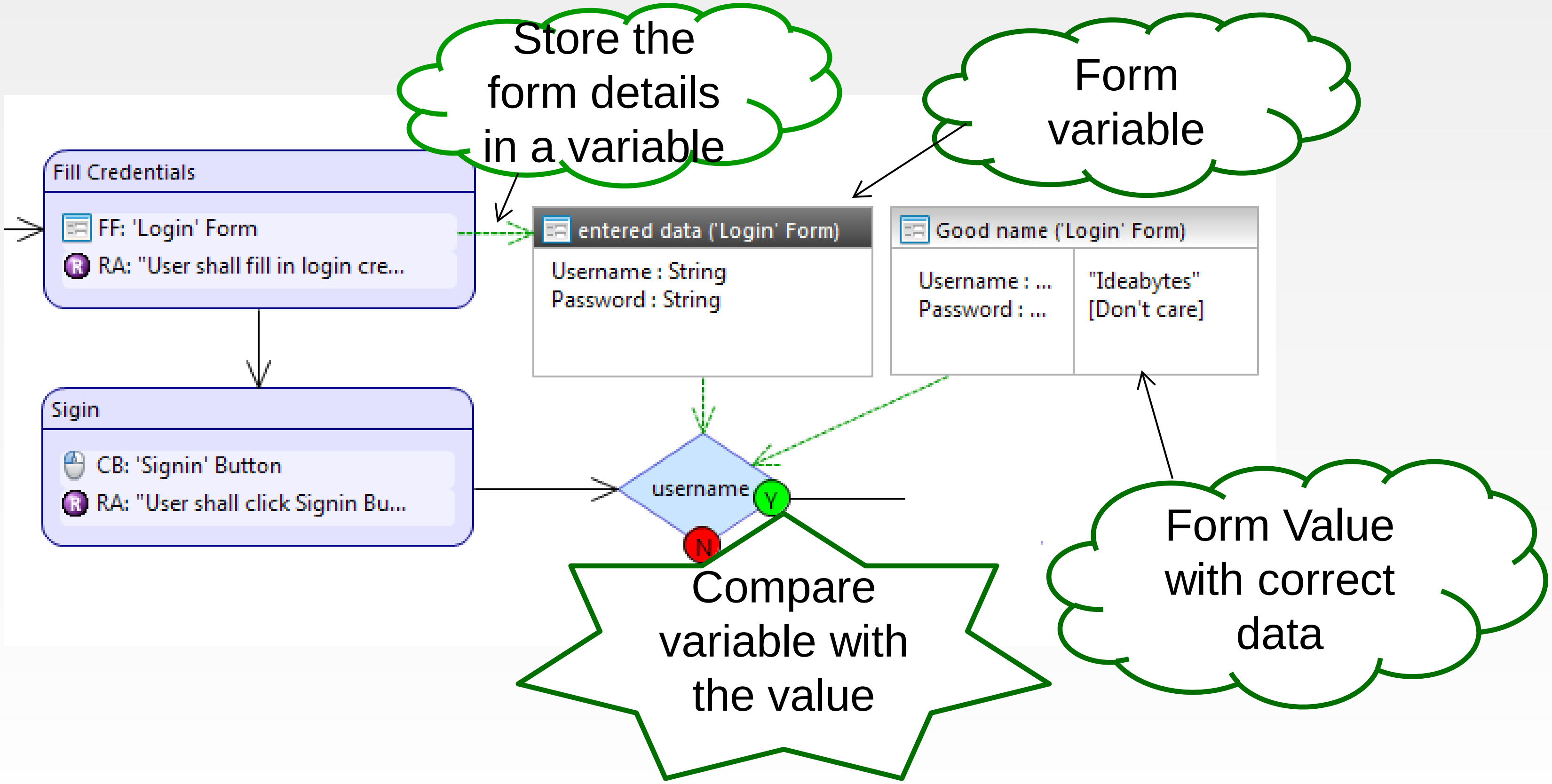
Three Choices for Condition Specification

- Completely graphical specification
  - Condition composed by drawing variable(s) or value to be compared and connecting them to decision diamond via data flows (operator always assumed to be EQUAL)
  - Easiest to specify and review but also most limited in expression
  - Currently only supported for decisions and only for comparing (complete) data objects (only) for equality
- Completely inline specification
  - Entire condition composed by selecting the variable(s) (field) or value to be compared and operator via Properties View
  - Most flexible that can be used in any condition specification
- Mixed graphical and inline specification
  - One or both sides to be compared are specified graphically and the rest (other side or operator) is specified via Properties View
  - Best used for decisions based on variable fields

Simple Condition Specification

Steps to Compare graphically

1) Connect Fill Form action (FF) to Form Variable using data flow
2) Connect Form Variable to decision using data flow
3) Connect Form Value to decision using data flow


Store the form details in a variable
Form variable
Fill Credentials
FF: 'Login' Form
RA: "User shall fill in login cre...
entered data ('Login' Form)
Username : String
Password : String
Good name ('Login' Form)
Username : ...
Password : ...
"Ideabytes"
[Don't care]
Sigin
CB: 'Signin' Button
RA: "User shall click Signin Bu...
username
Y
N
Compare variable with the value
Form Value with correct data

Complex Condition Specification

- “Combined conditions” (AND, OR, NOT, XOR) allows to compose multiple, different conditions (e.g. on a data table and message content)

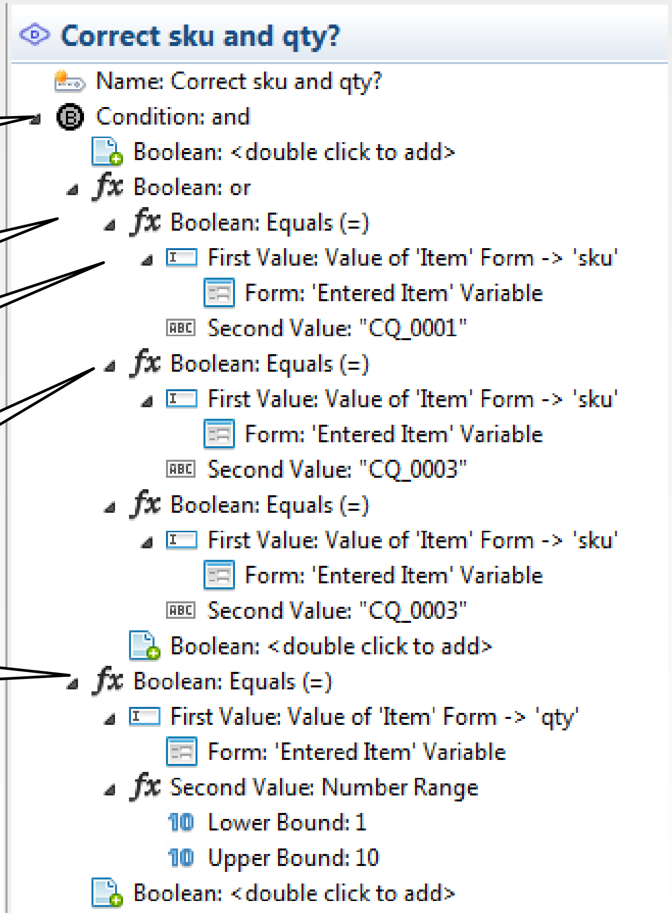
AND condition

OR condition

String condition

String condition

Number condition



Correct sku and qty?

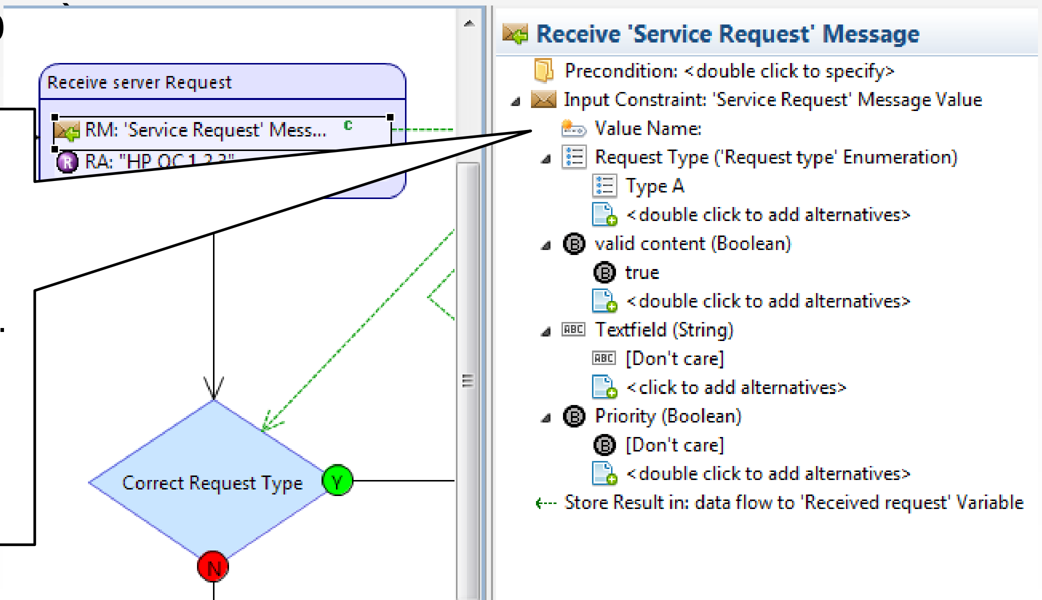
- Name: Correct sku and qty?
- Condition: and
  - Boolean: <double click to add>
  - Boolean: or
    - Boolean: Equals (=)
      - First Value: Value of 'Item' Form -> 'sku'
        - Form: 'Entered Item' Variable
      - Second Value: "CQ_0001"
    - Boolean: Equals (=)
      - First Value: Value of 'Item' Form -> 'sku'
        - Form: 'Entered Item' Variable
      - Second Value: "CQ_0003"
    - Boolean: Equals (=)
      - First Value: Value of 'Item' Form -> 'sku'
        - Form: 'Entered Item' Variable
      - Second Value: "CQ_0003"
    - Boolean: <double click to add>
  - Boolean: Equals (=)
    - First Value: Value of 'Item' Form -> 'qty'
      - Form: 'Entered Item' Variable
    - Second Value: Number Range
      - Lower Bound: 1
      - Upper Bound: 10
  - Boolean: <double click to add>

Input Action Data Constraints

- Allow constraining received data values in input actions (i.e., receive message and user entry actions)
  - Creator automatically constrains fields of enumeration type
  - “Data coverage” setting can be used to control the amount of input data generated by Conformiq Creator (default is at least one)
- Can only be specified using complete data object values
- Can also limit possibilities for Conformiq Creator to generate tests (e.g., to cover all conditio

Specifies that <u>any</u> Service Request received at this point <u>must have</u> “Type A” and “true” values for request type and valid content fields - text and Priority field values are not constrained.

NOTE: This eliminates the ability to specify decisions on request type and valid content fields after this action!

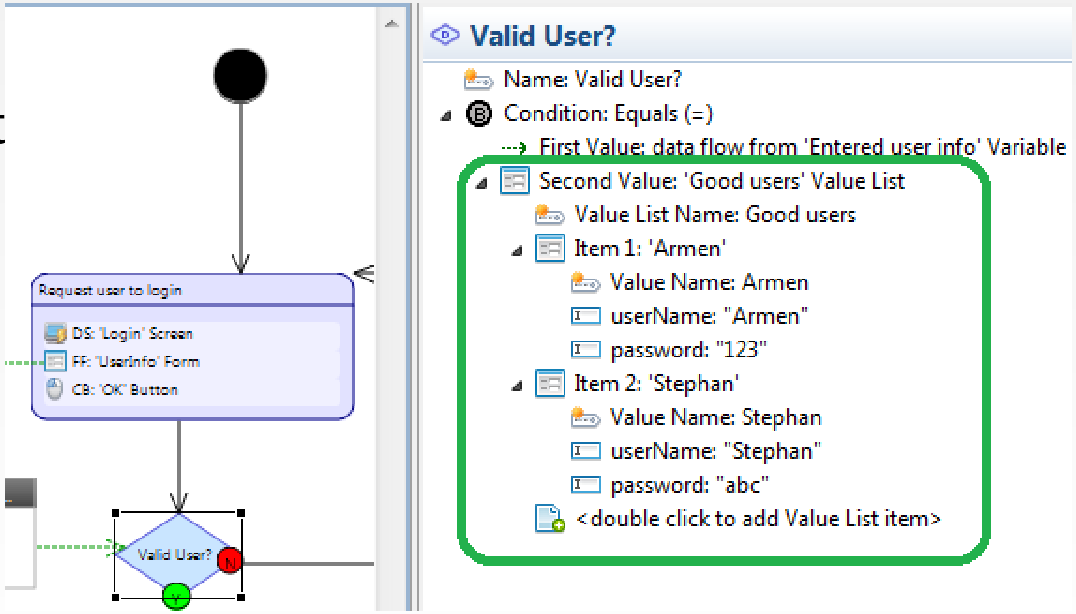


Receive 'Service Request' Message

- Precondition: <double click to specify>
- Input Constraint: 'Service Request' Message Value
  - Value Name:
  - Request Type ('Request type' Enumeration)
    - Type A
    - <double click to add alternatives>
  - valid content (Boolean)
    - true
    - <double click to add alternatives>
  - Textfield (String)
    - [Don't care]
    - <click to add alternatives>
  - Priority (Boolean)
    - [Don't care]
    - <double click to add alternatives>
- Store Result in: data flow to 'Received request' Variable

Dealing with Input Data Dependencies

- Often input field values may have complex dependencies
  - Simple example: User name & password during login
  - More advanced: Country & language selection on a web portal
- Easiest is to constrain input values is to specify relevant value combinations in a value list
  - Copy & paste values to speed up value list specification
  - Can also be used with decisions



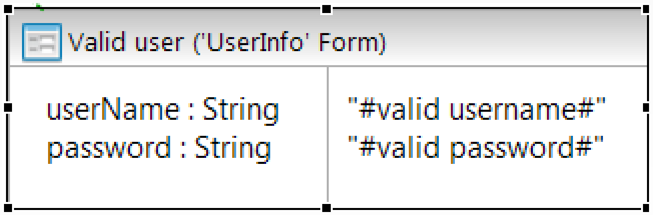
Valid User?

- Name: Valid User?
- Condition: Equals (=)
  - First Value: data flow from 'Entered user info' Variable
  - Second Value: 'Good users' Value List
    - Value List Name: Good users
    - Item 1: 'Armen'
      - Value Name: Armen
      - userName: "Armen"
      - password: "123"
    - Item 2: 'Stephan'
      - Value Name: Stephan
      - userName: "Stephan"
      - password: "abc"
    - <double click to add Value List item>

Data Driven MBT

Dealing with Dynamic Data: Placeholders

- In some situations actual values of fields may not be known at modeling time but important for test execution
  - See bank account number example in slide notes
- Such fields have to be modeled as Strings and so called place holders, i.e., strings with some special characters, to indicate that these values are not real but placeholder values
  - Examples: “#Account No#” or “#Valid User Name#
  - Placeholder can then be replaced at test execution time with real values
- Placeholders can also be used to facilitate data driven test execution
  - Example: Execute same tests generated with “#valid username#” *n* times with *n* concrete valid user names substitutions
  - This technique could be used in the “Simple Login” Activity Diagram part of the “Simple Web Application” example

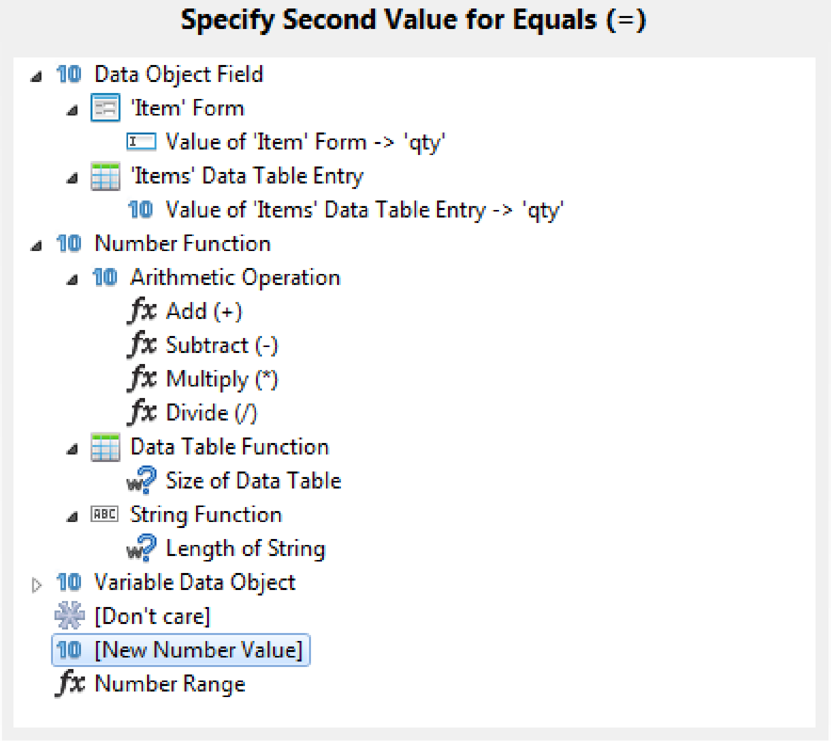


Valid user ('UserInfo' Form)

userName : String password : String	"#valid username#" "#valid password#"

Finally: Actions on data?

- Actions on data, e.g., concatenate strings or arithmetic computation, have to be done as part of verification actions or when adding a table entry
  - Predefined functions for basic types offer most basic data operations



Specify Second Value for Equals (=)

- Data Object Field
  - 'Item' Form
    - Value of 'Item' Form -> 'qty'
  - 'Items' Data Table Entry
    - Value of 'Items' Data Table Entry -> 'qty'
- Number Function
  - Arithmetic Operation
    - Add (+)
    - Subtract (-)
    - Multiply (*)
    - Divide (/)
  - Data Table Function
    - Size of Data Table
  - String Function
    - Length of String
- Variable Data Object
- [Don't care]
- [New Number Value]
- Number Range

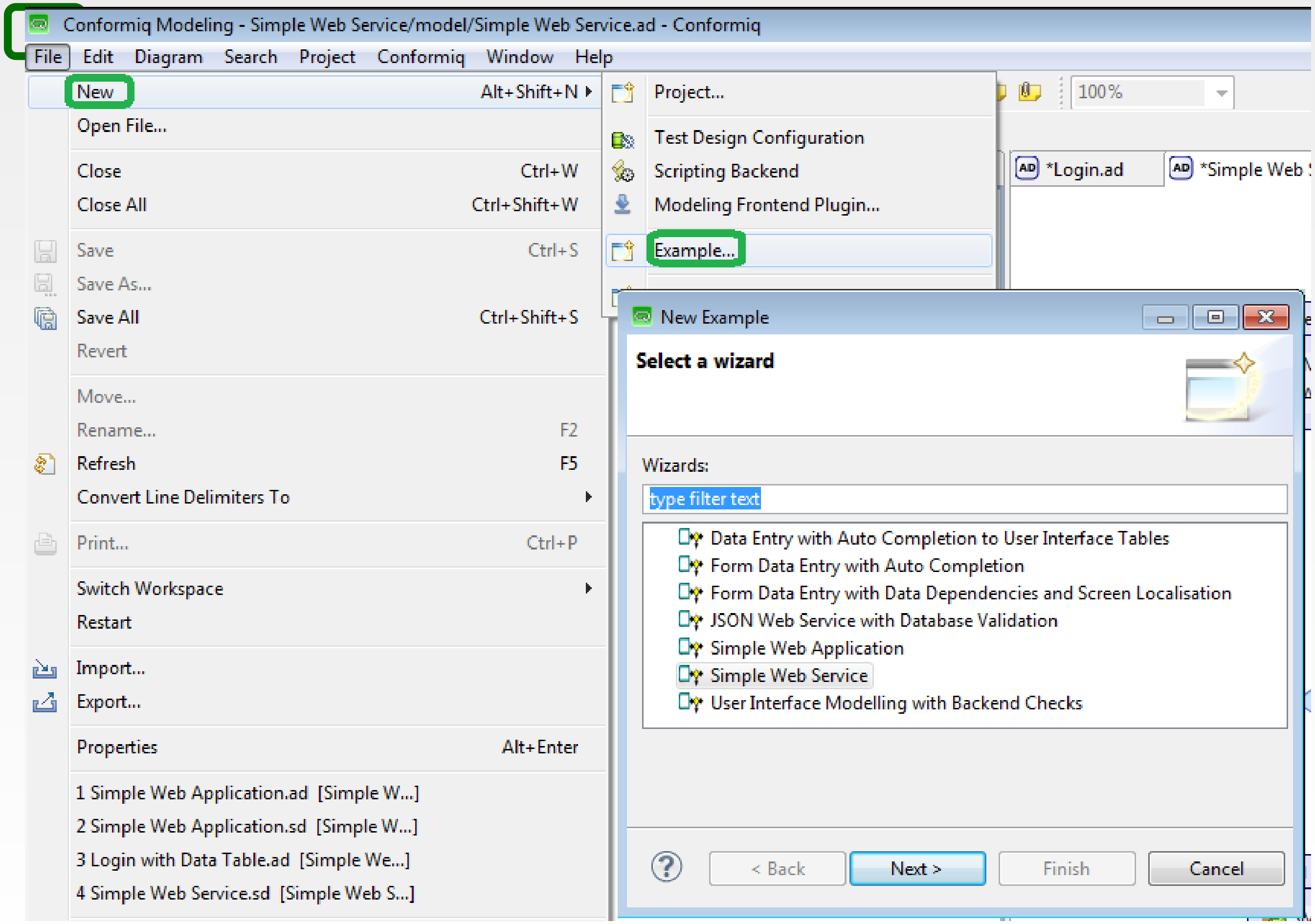
Example Number Functions

Basics of Working with Conformiq Creator Editor

Getting Started with Conformiq Creator Examples

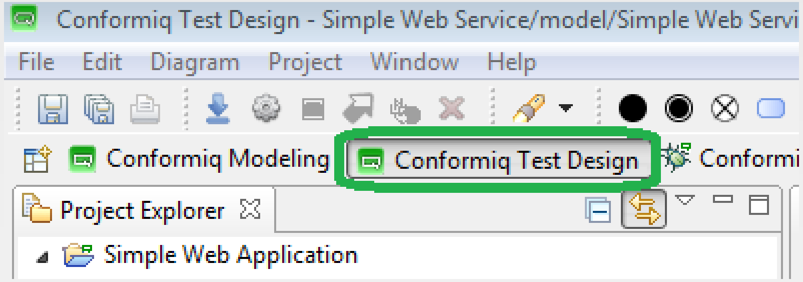
How do I Install an Example?

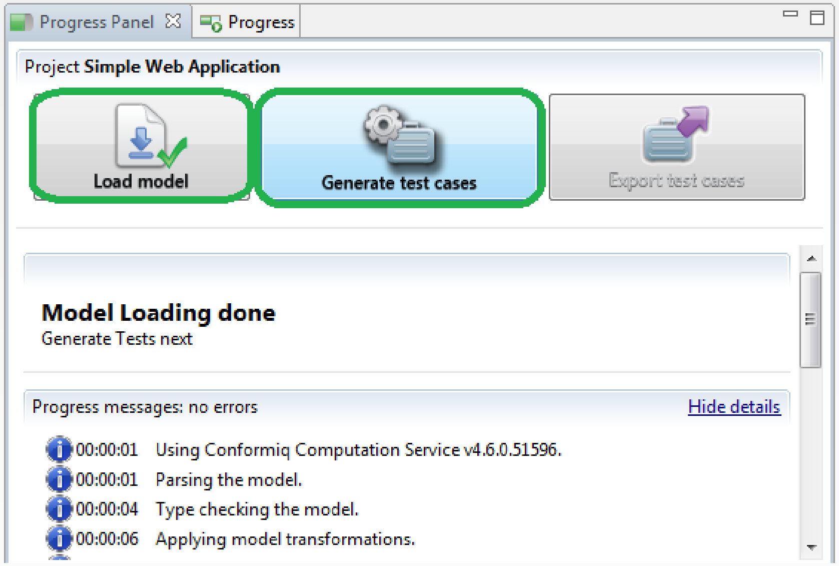
- Select File > New > Example and follow wizard instructions


Conformiq Modeling - Simple Web Service/model/Simple Web Service.ad - Conformiq
File
Edit
Diagram
Search
Project
Conformiq
Window
Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 Simple Web Application.ad [Simple W...]
2 Simple Web Application.sd [Simple W...]
3 Login with Data Table.ad [Simple We...]
4 Simple Web Service.sd [Simple Web S...]
Project...
Test Design Configuration
Scripting Backend
Modeling Frontend Plugin...
Example...
100%
*Login.ad
*Simple Web
New Example
Select a wizard
Wizards:
type filter text
Data Entry with Auto Completion to User Interface Tables
Form Data Entry with Auto Completion
Form Data Entry with Data Dependencies and Screen Localisation
JSON Web Service with Database Validation
Simple Web Application
Simple Web Service
User Interface Modelling with Backend Checks
< Back
Next >
Finish
Cancel

How To Get Tests from the Example(s)

- Select “Conformiq Test Design” perspective
- Click “Load model” and “Generate test cases” in Progress Panel view
- Review test in test case and test case steps view



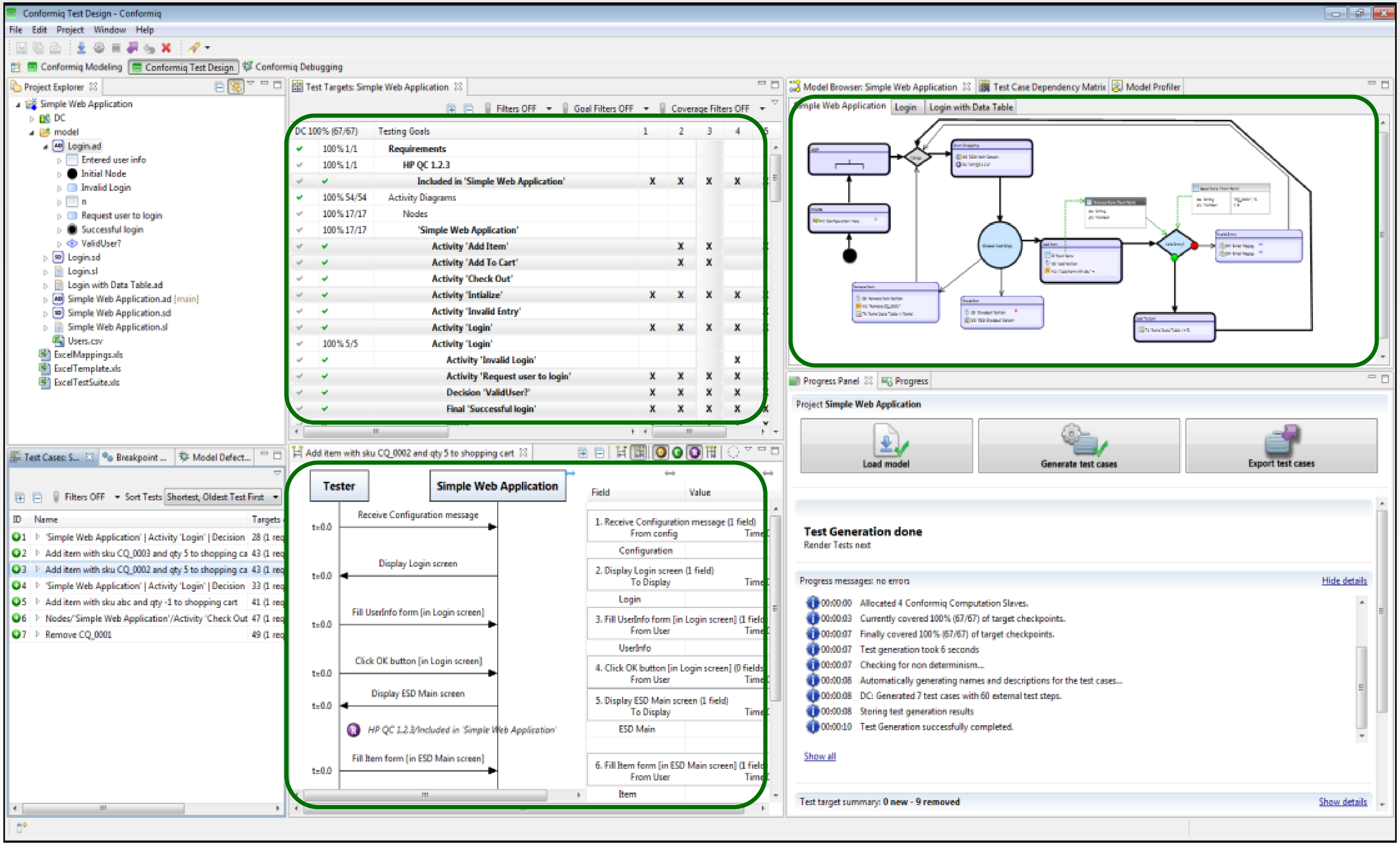


Model Loading done
Generate Tests next

Progress messages: no errors

```
00:00:01  Using Conformiq Computation Service v4.6.0.51596.
00:00:01  Parsing the model.
00:00:04  Type checking the model.
00:00:06  Applying model transformations.
```

How To Review Test Generation Results



DC 100% (67/67)		Testing Goals	1	2	3	4	5
✓	100% 1/1	Requirements					
✓	100% 1/1	HP QC 1.2.3					
✓	✓	Included in 'Simple Web Application'	X	X	X	X	X
✓	100% 54/54	Activity Diagrams					
✓	100% 17/17	Nodes					
✓	100% 17/17	'Simple Web Application'					
✓	✓	Activity 'Add Item'		X	X		X
✓	✓	Activity 'Add To Cart'		X	X		
✓	✓	Activity 'Check Out'					
✓	✓	Activity 'Intialize'	X	X	X	X	X
✓	✓	Activity 'Invalid Entry'					X
✓	✓	Activity 'Login'	X	X	X	X	X
✓	100% 5/5	Activity 'Login'					
✓	✓	Activity 'Invalid Login'				X	
✓	✓	Activity 'Request user to login'	X	X	X	X	X
✓	✓	Decision 'ValidUser?'	X	X	X	X	X
✓	✓	Final 'Successful login'	X	X	X	X	X

ID	Name	Targets
1	'Simple Web Application' | Activity 'Login' | Decision	28 (1 req
2	Add item with sku CQ_0003 and qty 5 to shopping ca	43 (1 req
3	Add item with sku CQ_0002 and qty 5 to shopping ca	43 (1 req
4	'Simple Web Application' | Activity 'Login' | Decision	33 (1 req
5	Add item with sku abc and qty -1 to shopping cart	41 (1 req
6	Nodes/'Simple Web Application'/Activity 'Check Out	47 (1 req
7	Remove CQ_0001	49 (1 req

Tester → Simple Web Application

- t=0.0 Receive Configuration message
- t=0.0 Display Login screen
- t=0.0 Fill UserInfo form [in Login screen]
- t=0.0 Click OK button [in Login screen]
- t=0.0 Display ESD Main screen
- HP QC 1.2.3/Included in 'Simple Web Application'
- t=0.0 Fill Item form [in ESD Main screen]

Field	Value
1. Receive Configuration message (1 field) From config	Time
Configuration	
2. Display Login screen (1 field) To Display	Time
Login	
3. Fill UserInfo form [in Login screen] (1 field From User	Time
UserInfo	
4. Click OK button [in Login screen] (0 fields From User	Time
5. Display ESD Main screen (1 field) To Display	Time
ESD Main	
6. Fill Item form [in ESD Main screen] (1 field From User	Time
Item	

Project **Simple Web Application**

Test Generation done
Render Tests next

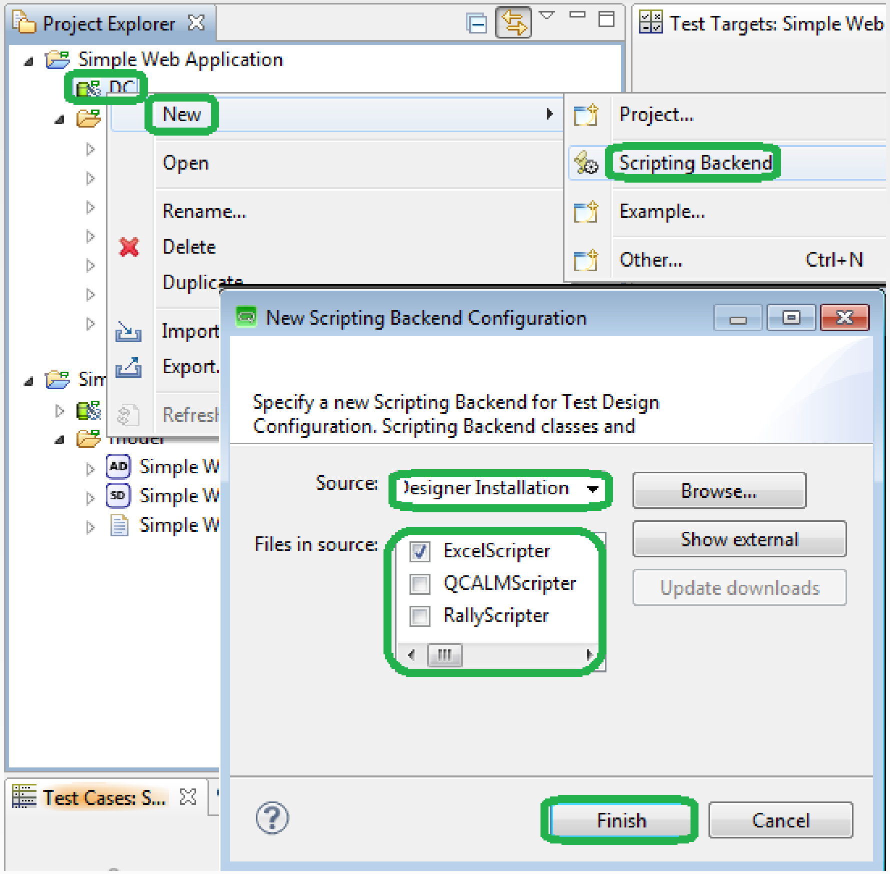
Progress messages: no errors

```
00:00:00  Allocated 4 Conformiq Computation Slaves.
00:00:03  Currently covered 100% (67/67) of target checkpoints.
00:00:07  Finally covered 100% (67/67) of target checkpoints.
00:00:07  Test generation took 6 seconds
00:00:07  Checking for non determinism...
00:00:08  Automatically generating names and descriptions for the test cases...
00:00:08  DC: Generated 7 test cases with 60 external test steps.
00:00:08  Storing test generation results
00:00:10  Test Generation successfully completed.
```

Test target summary: **0 new - 9 removed**

How to Render Tests to Excel, Rally, etc

- In Example project right click on Test Design Configuration icon in project explorer and select New > Scripting Backend
- Select “Designer Installation”
- Select one or more scripting backends
- Click on “Render Tests” button in Conformiq Creator toolbar and open generated output files with applicable tool
- For further details on scripting backend configuration see Conformiq Creator manual



Specify a new Scripting Backend for Test Design Configuration. Scripting Backend classes and

Source: Designer Installation

Files in source:
☑ ExcelScripter
□ QCALMScripter
□ RallyScripter

Modeling with the Conformiq Creator Editor

Getting Started with your own Creator Project

Hands on exercise :

# Creating a Login *Model*

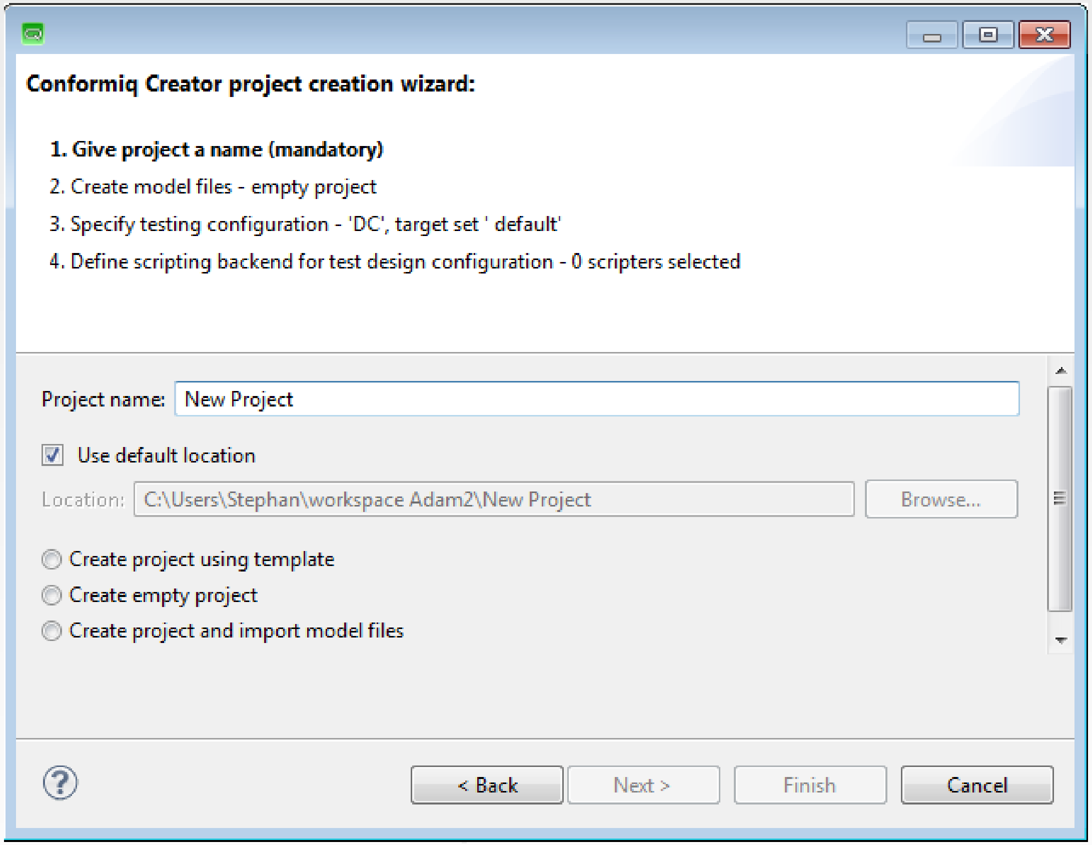
Table of Contents

- Creating a New *Conformiq Creator* Project
- Creating an Activity Diagram
- Creating an *Structure Diagram*
- Refining the *Activity Diagram* with Generated *Actions*
- Generating *Test Cases* from an *Activity Diagram*
- Analyzing *Test Cases* in *Conformiq Creator*
- Modifying a *Model*

Creating a New *Conformiq Creator* Project

1) In *Conformiq Creator*, from the main menu, select

***File->New->Conformiq Creator Project* ➔** A New Project Dialog box appears



Conformiq Creator project creation wizard:

1. Give project a name (mandatory)

2. Create model files - empty project

3. Specify testing configuration - 'DC', target set ' default'

4. Define scripting backend for test design configuration - 0 scripters selected

Project name: New Project

☑ Use default location

Location: C:\Users\Stephan\workspace Adam2\New Project

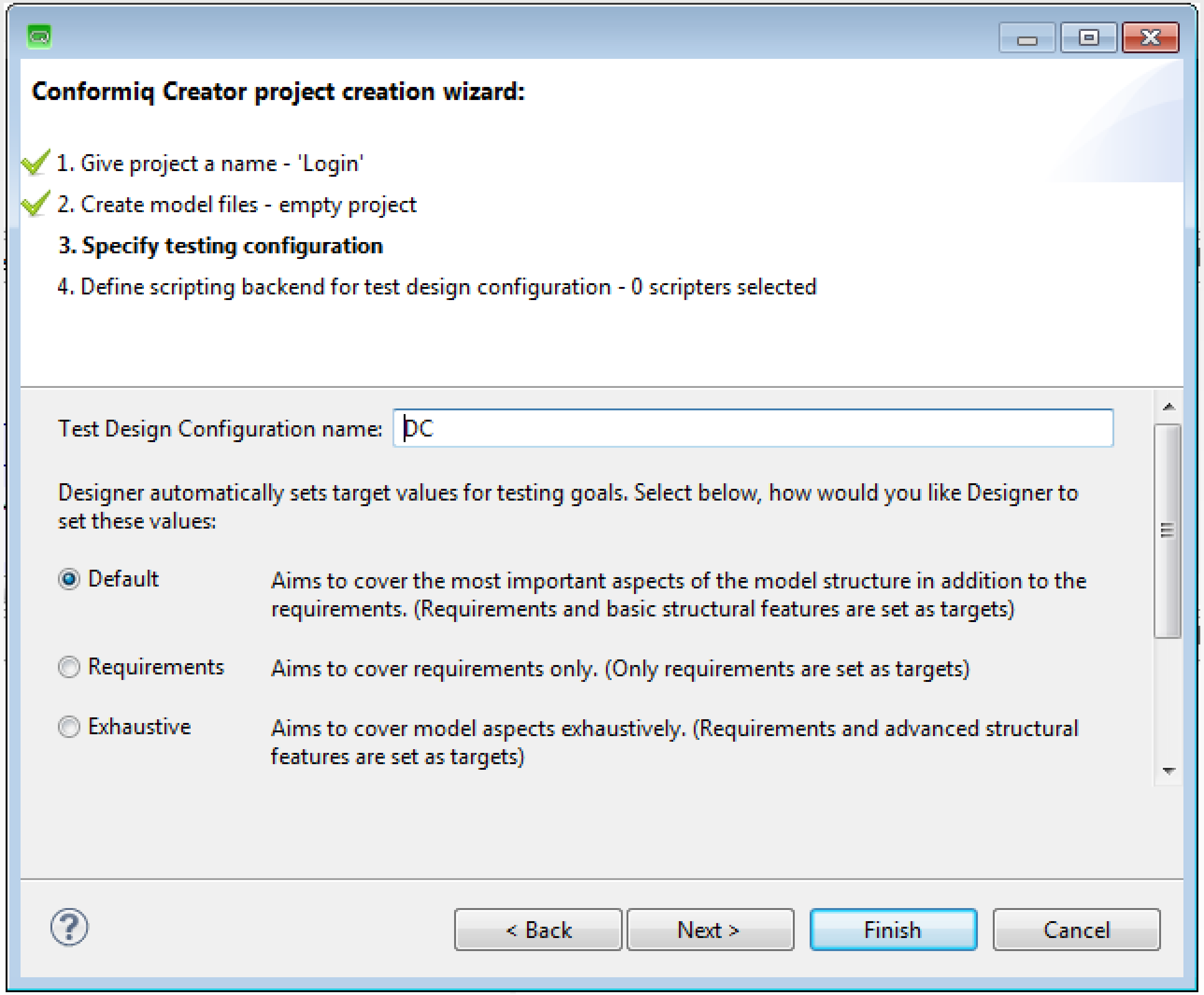
○ Create project using template

○ Create empty project

○ Create project and import model files

2) Type "*Login Model (e.g.. Gmail)*" in Project name field, and select "Create empty project", and click the *Next* button.

2) Once below Window appears, accept the default **Test Design Configuration Name**: “*DC*”, and click the *Next* button.



Conformiq Creator project creation wizard:

✓ 1. Give project a name - 'Login'

✓ 2. Create model files - empty project

3. Specify testing configuration

4. Define scripting backend for test design configuration - 0 scripters selected

Test Design Configuration name: DC

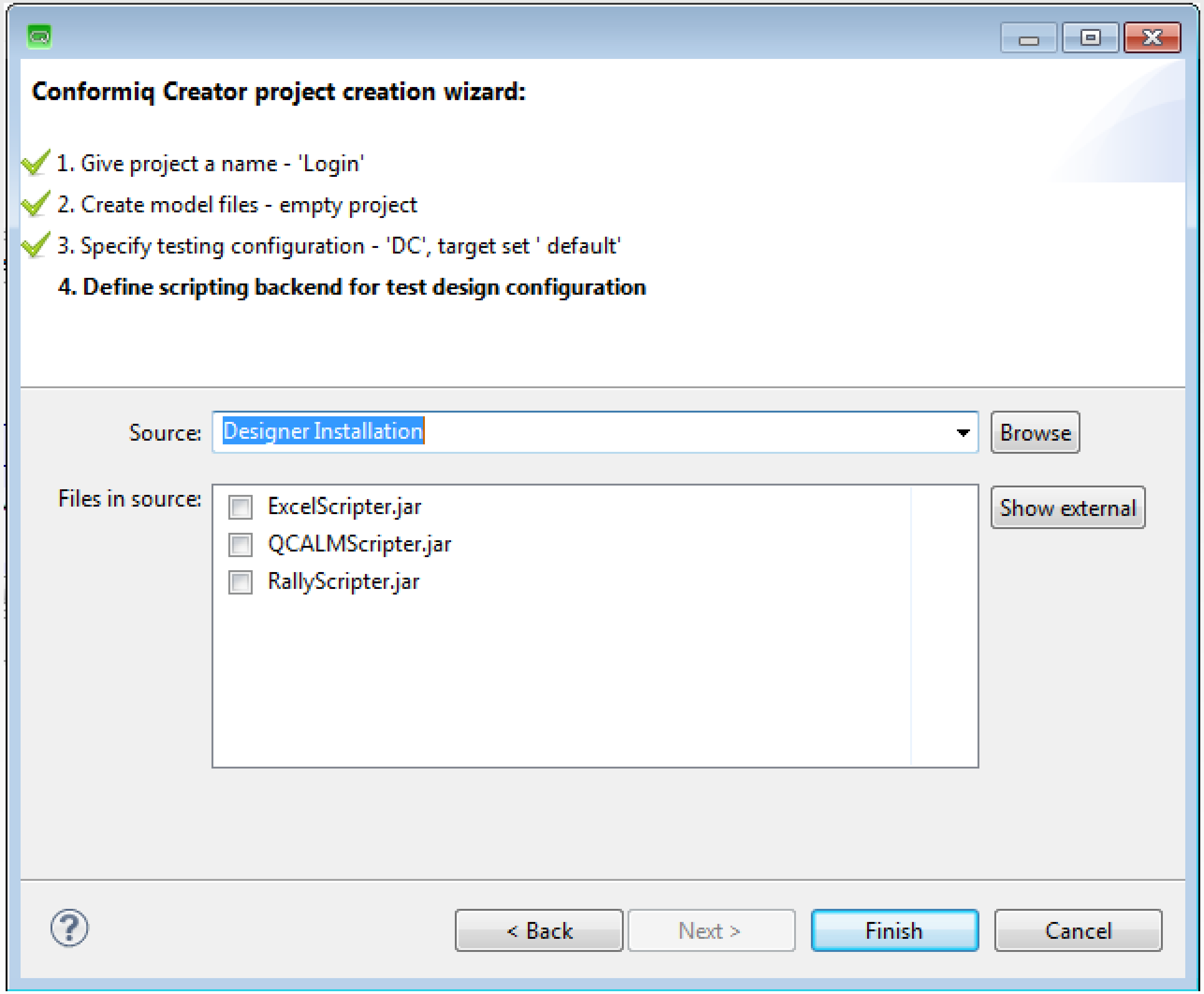
Designer automatically sets target values for testing goals. Select below, how would you like Designer to set these values:

● Default — Aims to cover the most important aspects of the model structure in addition to the requirements. (Requirements and basic structural features are set as targets)

○ Requirements — Aims to cover requirements only. (Only requirements are set as targets)

○ Exhaustive — Aims to cover model aspects exhaustively. (Requirements and advanced structural features are set as targets)

3) Below window appears; you need to define your *Scripting Backend(s)*. In this initial exercise we will not need any, so just click the *Finish* button.



Conformiq Creator project creation wizard:

✓ 1. Give project a name - 'Login'

✓ 2. Create model files - empty project

✓ 3. Specify testing configuration - 'DC', target set ' default'

4. Define scripting backend for test design configuration

Source: Designer Installation

Files in source:

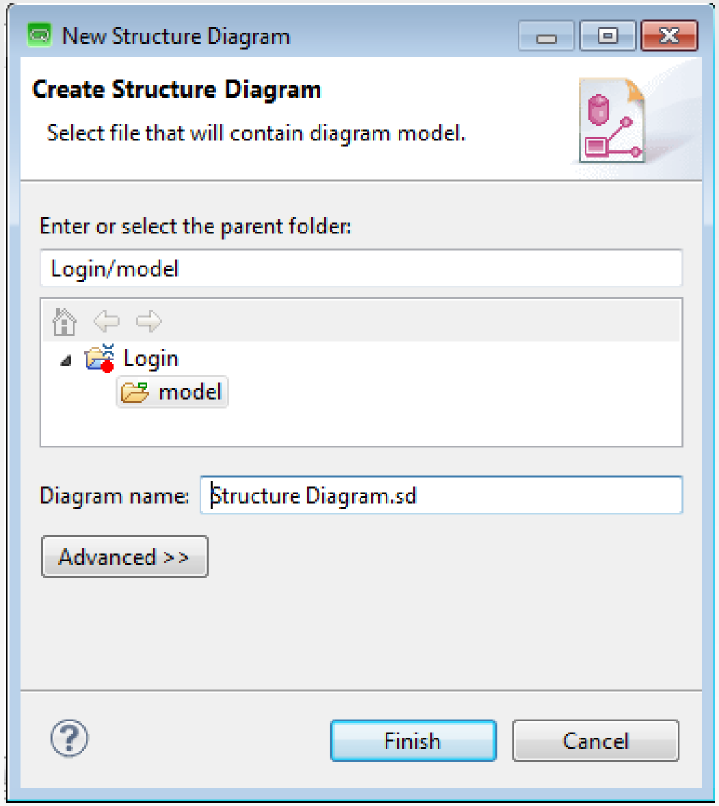
☐ ExcelScripter.jar

☐ QCALMScripter.jar

☐ RallyScripter.jar

Creating an *Structure Diagram*

1) In *Conformiq Creator*, in the *Project Explorer* view, right-click on *model* folder, and select *New > Structure Diagram*.
2) The following *Create Structure Diagram* dialog appears:



New Structure Diagram

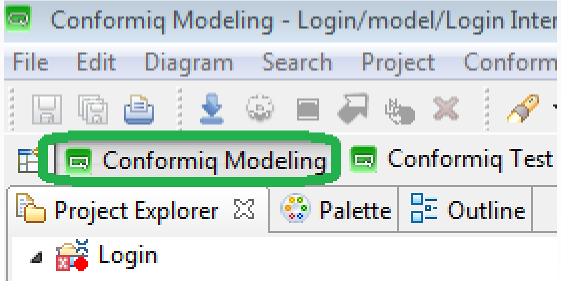
Create Structure Diagram

Select file that will contain diagram model.

Enter or select the parent folder: Login/model

Diagram name: Structure Diagram.sd

3) In the **Diagram Name** field, type "*Login Interfaces*".
4) Click on "Conformiq Modeling perspective" – an empty *Structure Diagram* should be opened in the File viewer.

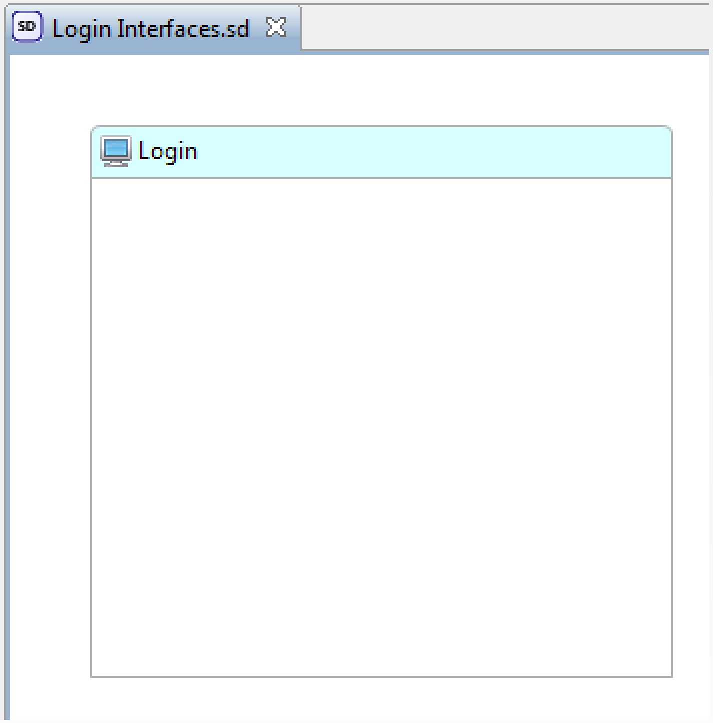


Conformiq Modeling - Login/model/Login Inter

5) Click on the *Screen* symbol 🖥 from the Modeling toolbar and then click anywhere in the canvas to place it.

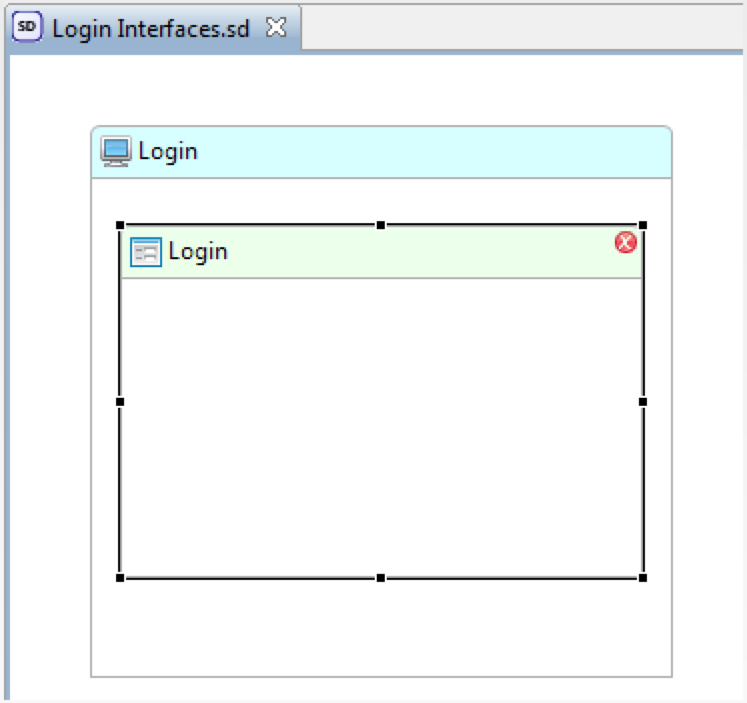
6) Type in “Login”. Click *Save*.

7) A “Login” Screen should appear in the Structure Diagram as shown below:

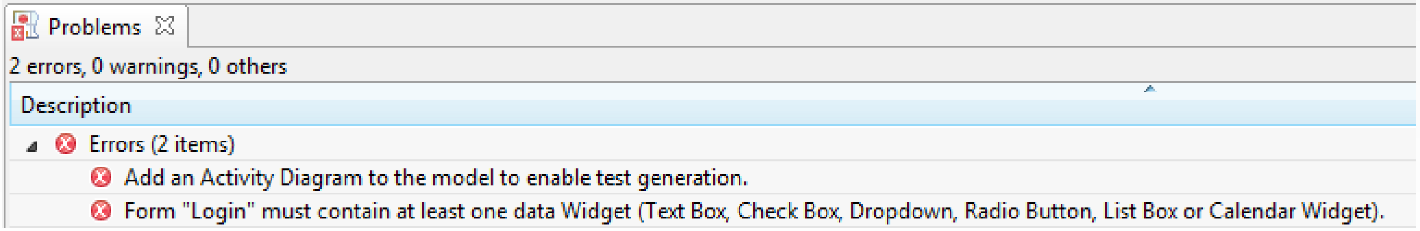


8) Click on the Form symbol ▤ from the Modeling toolbar, and then click within the screen object to place it there.

9) Type “Login”. A “Login” *Form* is now added to the Structure Diagram. Click *Save*.



10) Note the livecheck errors in the Problems view.



2 errors, 0 warnings, 0 others

Description

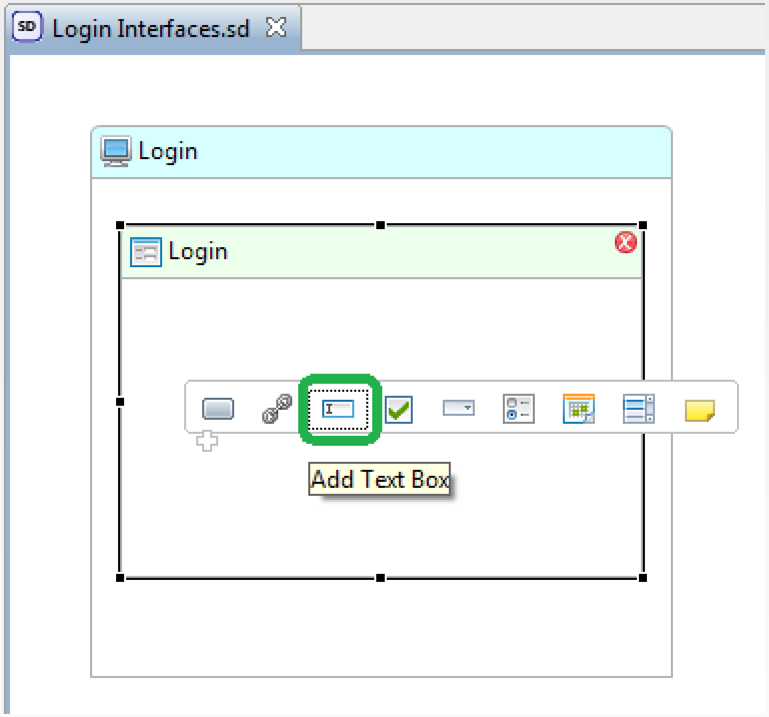
Errors (2 items)

Add an Activity Diagram to the model to enable test generation.

Form "Login" must contain at least one data Widget (Text Box, Check Box, Dropdown, Radio Button, List Box or Calendar Widget).

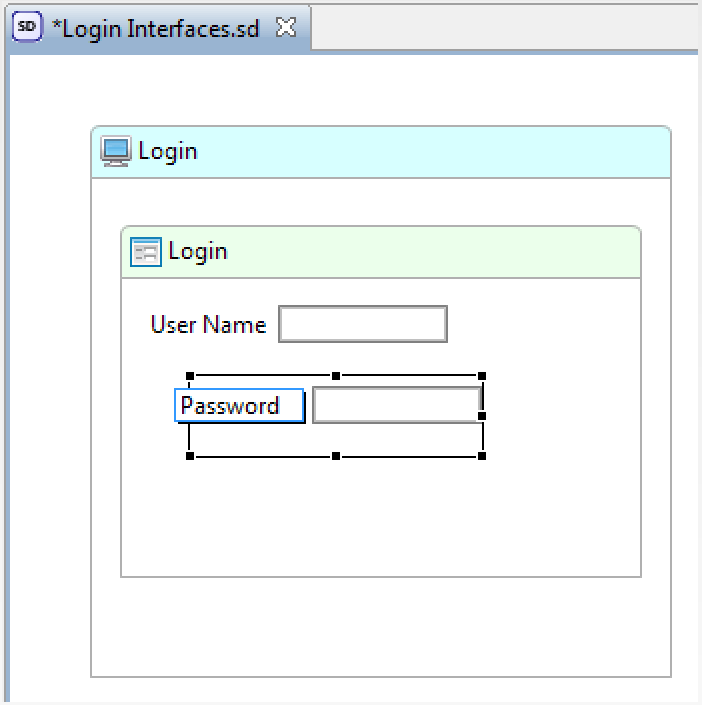
11) A *Form* should have at least one data widget (see related Error). Let's add some data widgets to the newly created form.

12) Hover over the *"Login" Form* and select the text box widget icon from the hover tool bar.

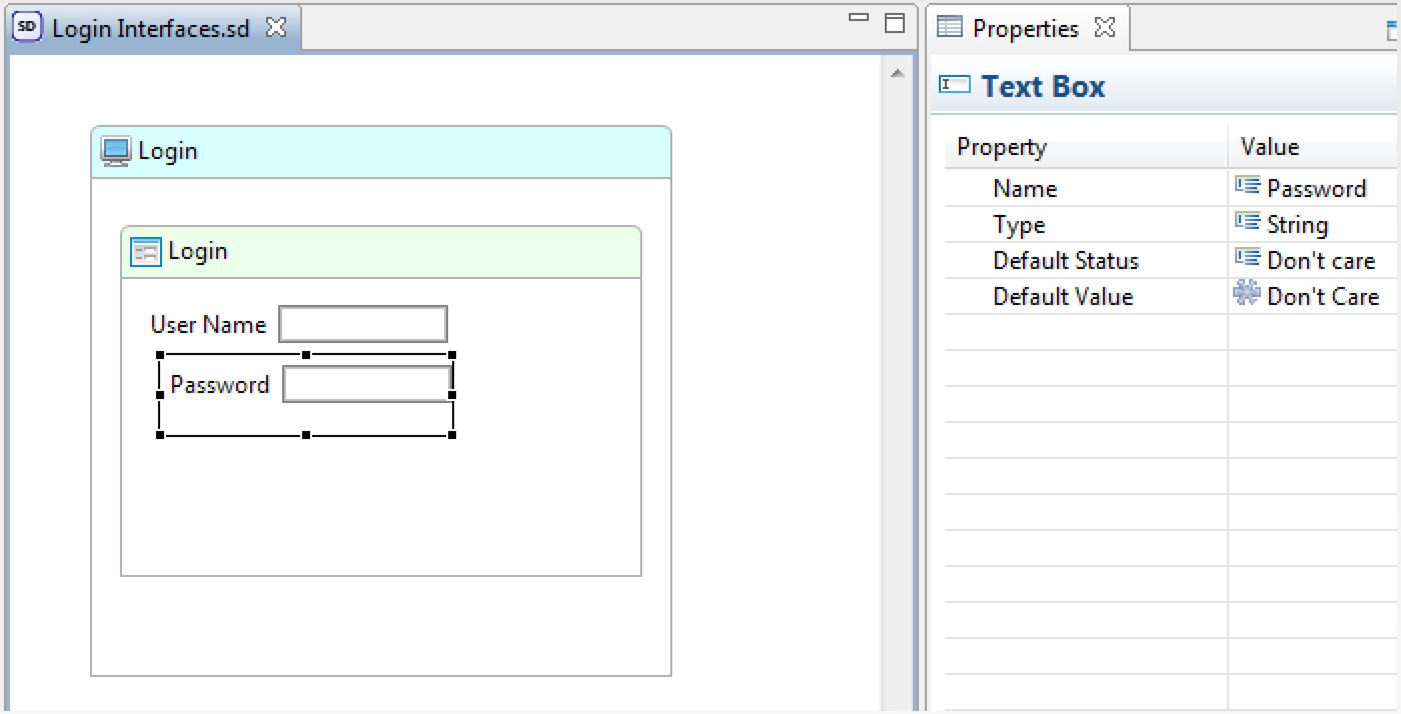


13) Type "*User Name*", followed by <*ENTER*>. Click *Save*. Note that one of the livecheck errors has now disappeared

14) Repeat the last step to add another *Text Box*, named *“Password”*. Click *Save*.



15) Your *Structure Diagram* should look similar to this at this point:



Property	Value
Name	Password
Type	String
Default Status	Don't care
Default Value	Don't Care

Note that you can also edit the properties of selected items like name and default widgets status etc for selected objects in the Properties View

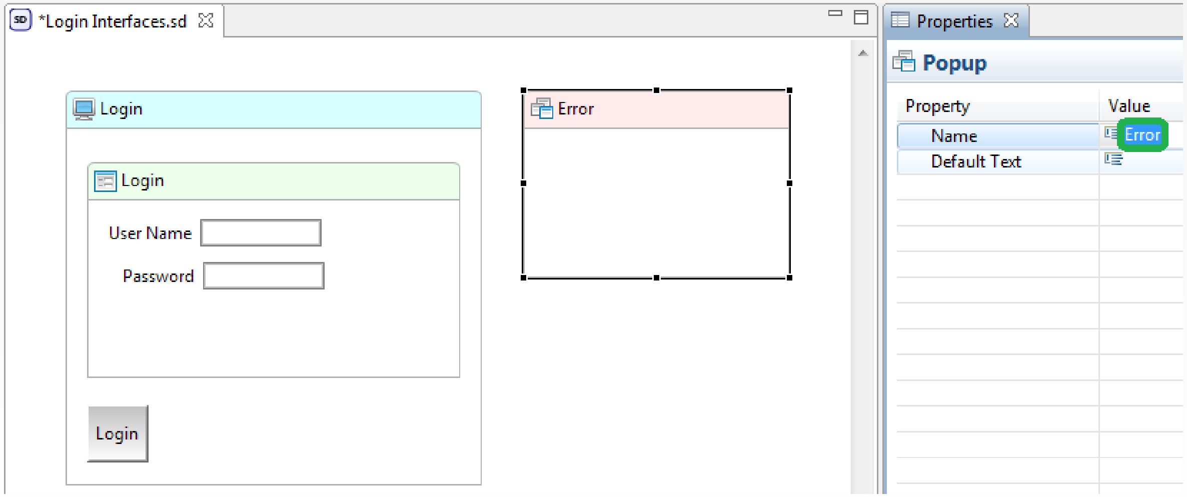
16) Let’s add one more widget: a “*Login*” button. We can either add it to the *Form* or to the *Screen*. Let’s add it to the *Login Screen*.

17) Add a *button widget* to the *Login Screen* (look at the previous steps if needed), this time select *Button* from the list of elements (under *Basic Widgets*).

- Name the button “*Login*”.
- Next, let’s create a *Popup* dialog for an error message.

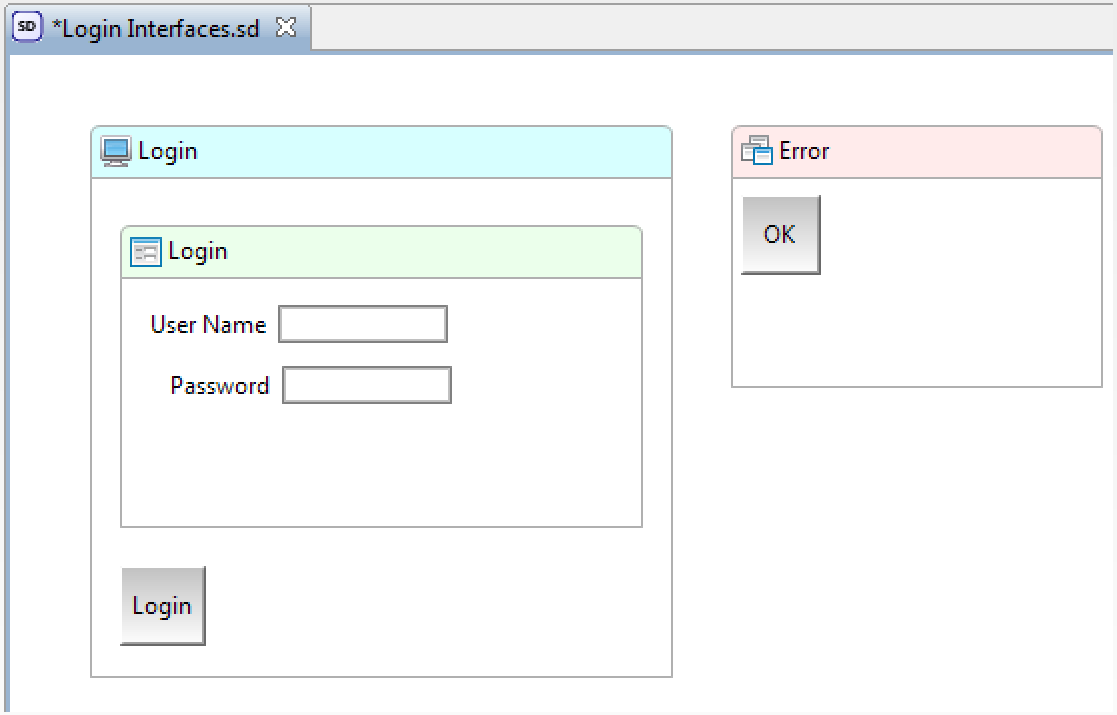
18) Click on the *Popup* symbol 🗔 from the modeling toolbar, then click anywhere in the canvas to place it.

19) Instead of typing the name directly let’s use the *Property View* to specify the name. Click <*ENTER*>. Type “*Error*” in the **Name** field in the *Property View* and click <*ENTER*>.

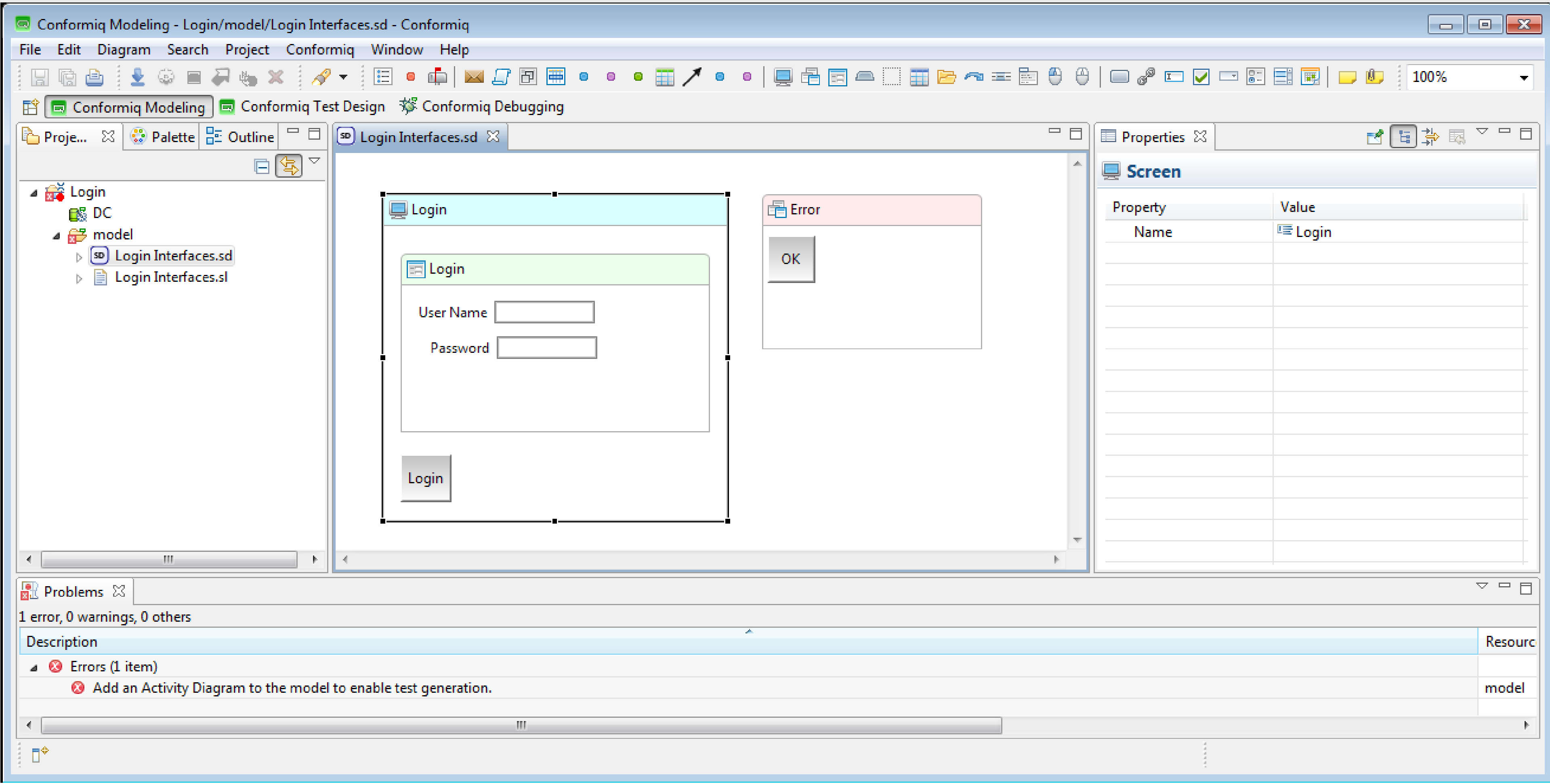


Property	Value
Name	Error
Default Text	

20) Select the *Button* widget from the tool bar and click in the popup.
21) Type in “OK” as the *Name*
22) A button named “OK” is now added to the *Popup* as shown below



23) Save your diagram. The *Structure Diagram* should now appear as below.



Property	Value
Name	Login

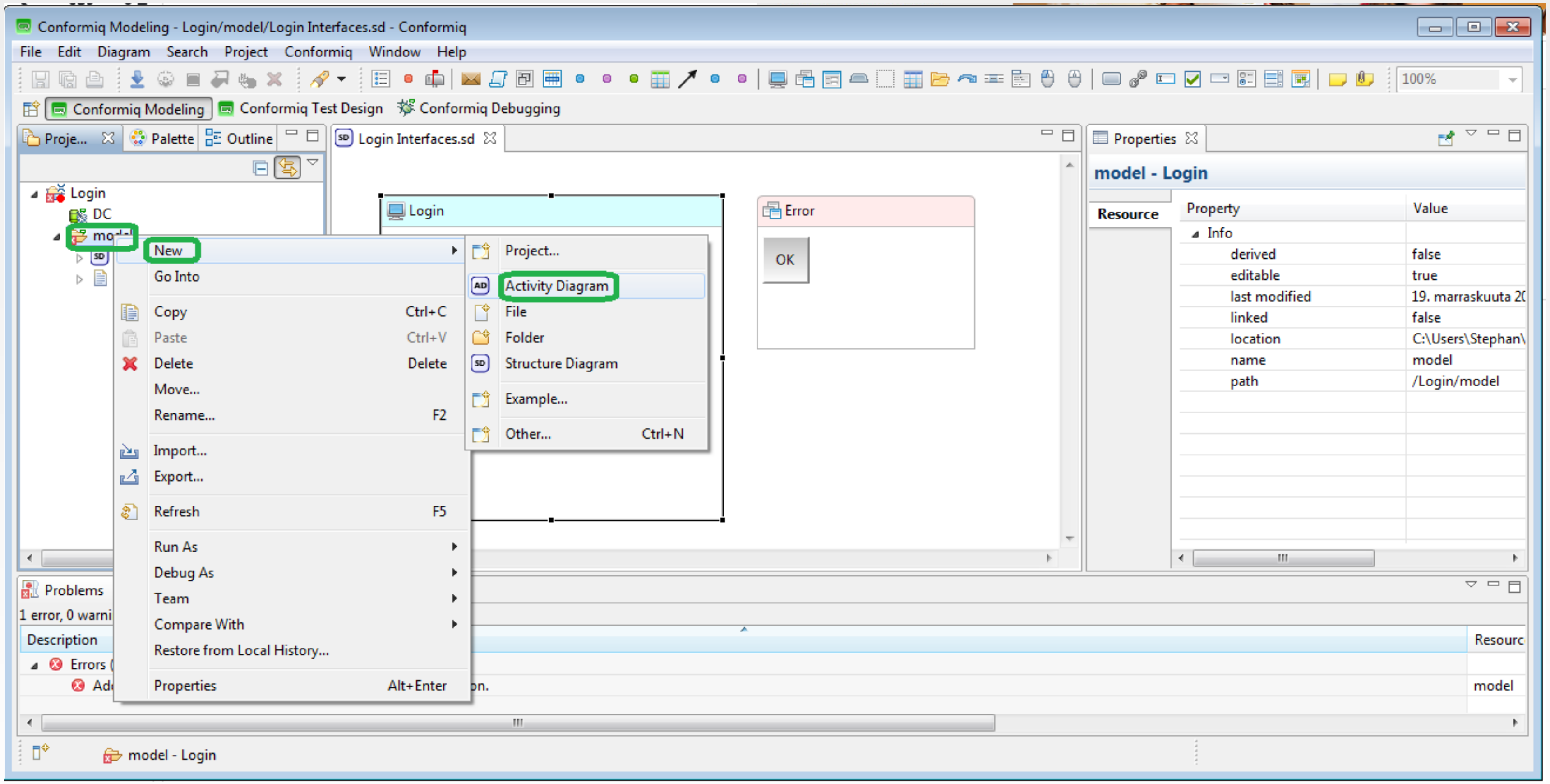
1 error, 0 warnings, 0 others

Description	Resourc
Errors (1 item)	
Add an Activity Diagram to the model to enable test generation.	model

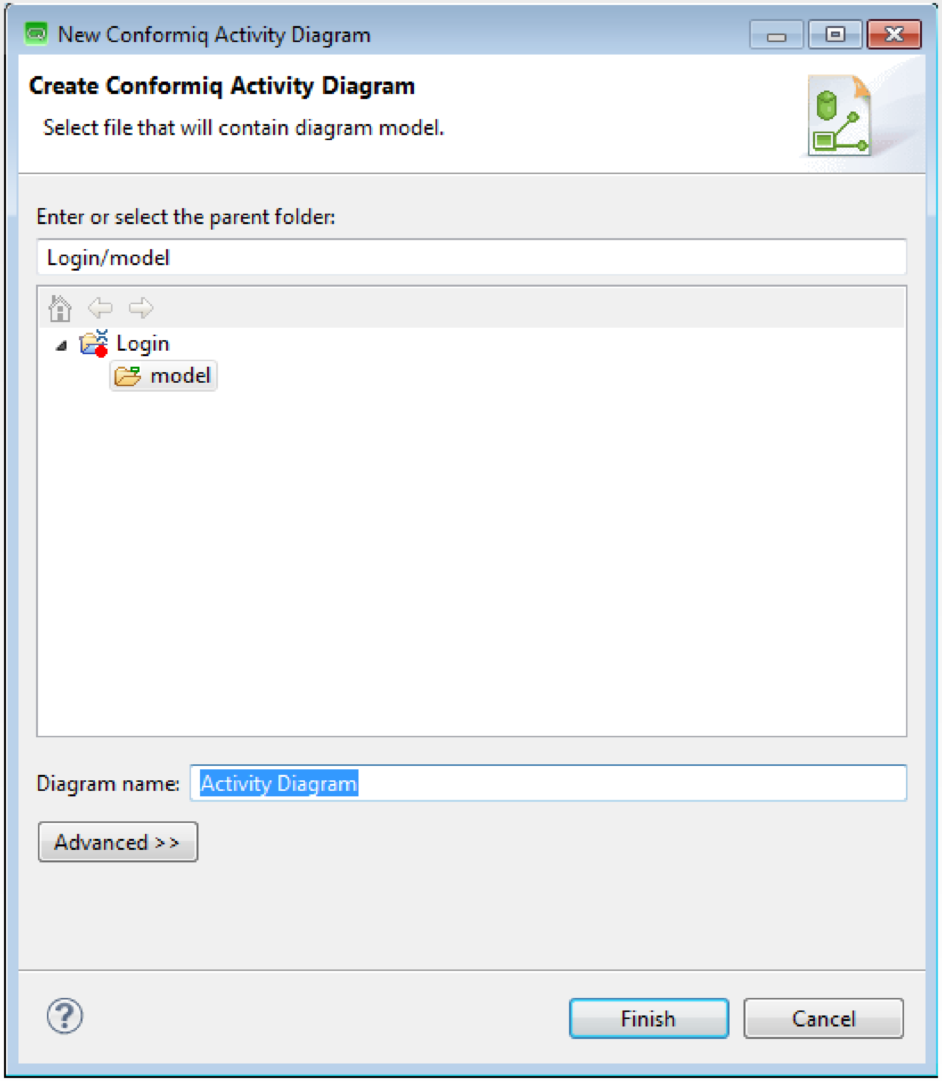
Creating an Activity Diagram

1) Now let's create Activity Diagram and refine it accordingly for Test Generation, using the Actions we've created previously.

2) Right-click on *model* folder, and select *New > Activity Diagram* as shown



3) In the *Create Activity Diagram* dialog type in **Diagram Name** field "*Login App*"



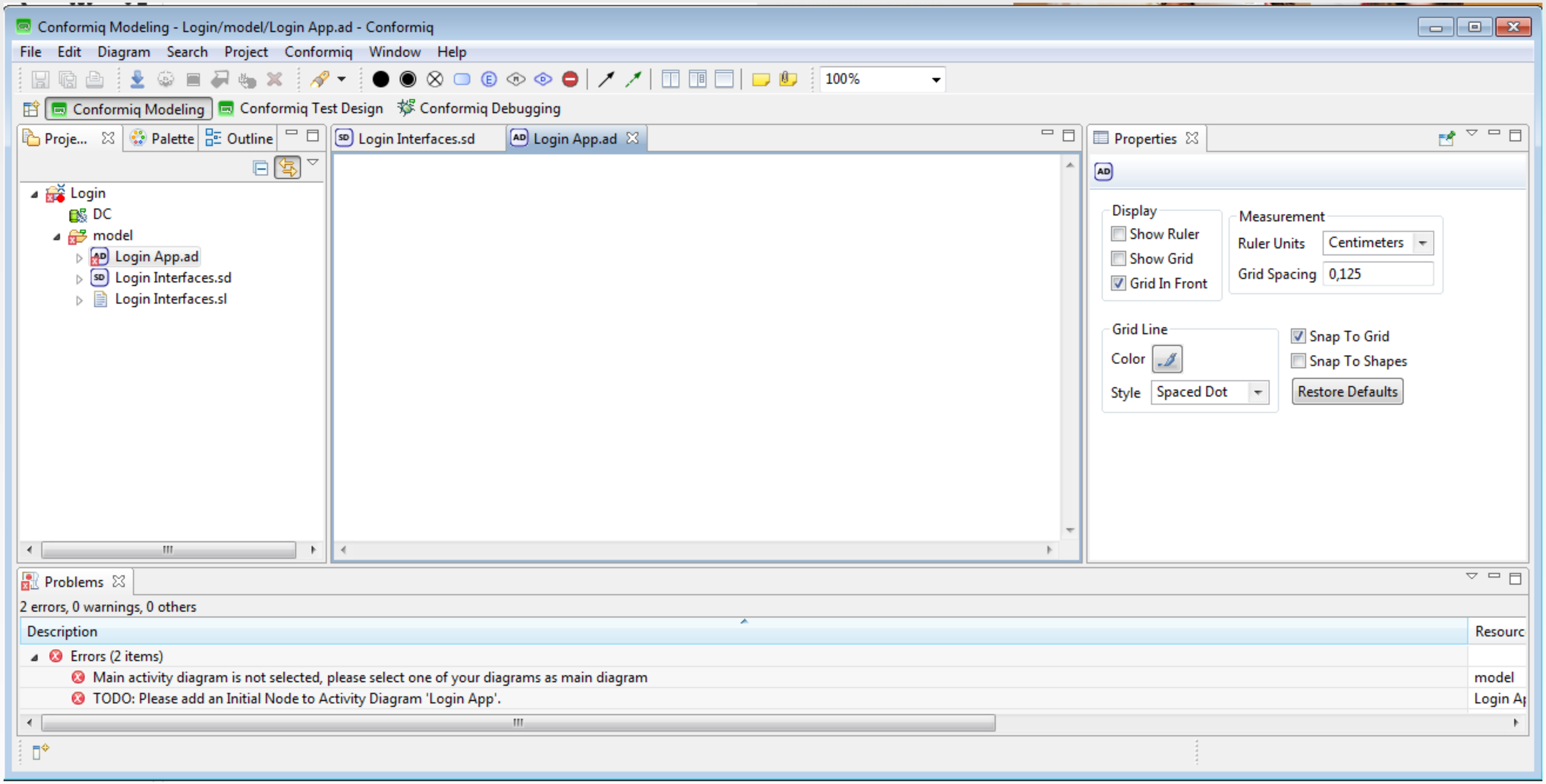
Create Conformiq Activity Diagram

Select file that will contain diagram model.

Enter or select the parent folder: Login/model

Diagram name: Activity Diagram

4) A new *Activity Diagram* should be opened as below in the *Conformiq Modeling Perspective*.



Conformiq Modeling - Login/model/Login App.ad - Conformiq

- Login
  - DC
  - model
    - Login App.ad
    - Login Interfaces.sd
    - Login Interfaces.sl

Display
- ☐ Show Ruler
- ☐ Show Grid
- ☑ Grid In Front

Measurement
- Ruler Units: Centimeters
- Grid Spacing: 0,125

Grid Line
- Color
- Style: Spaced Dot

- ☑ Snap To Grid
- ☐ Snap To Shapes

Restore Defaults

Problems

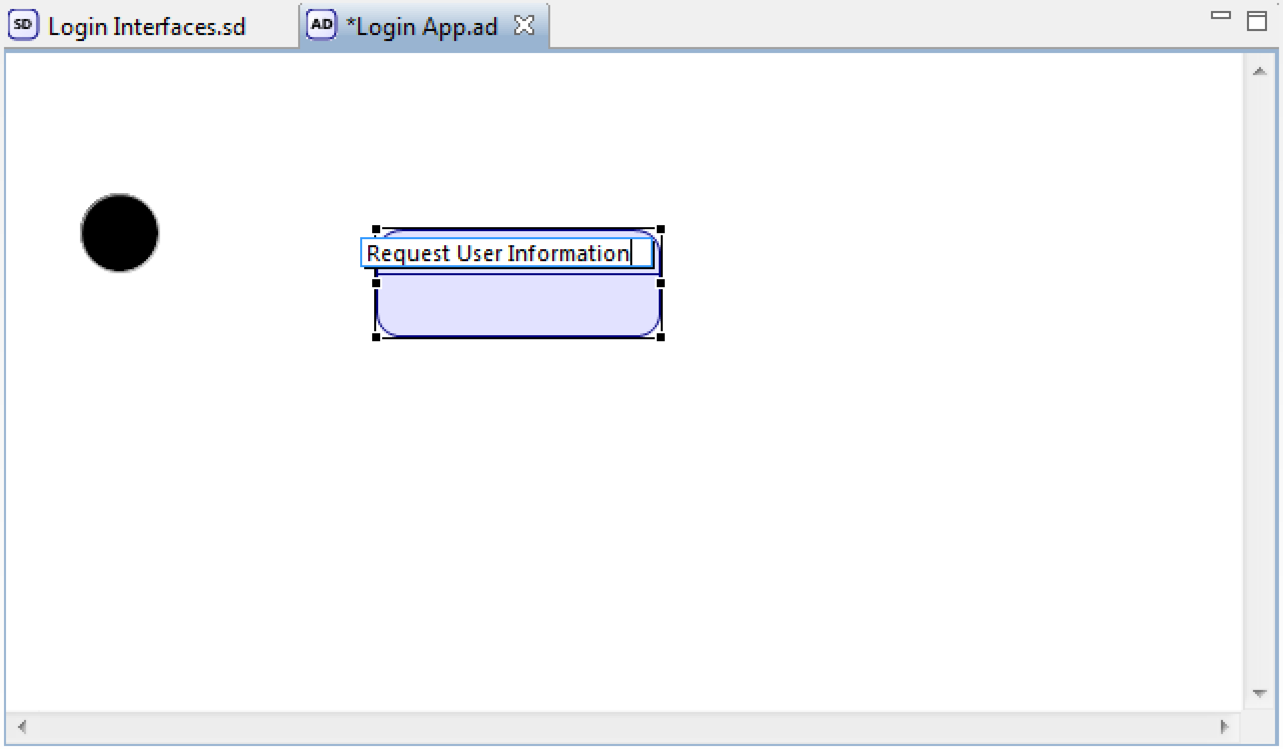
2 errors, 0 warnings, 0 others

Description	Resourc
Errors (2 items)	
Main activity diagram is not selected, please select one of your diagrams as main diagram	model
TODO: Please add an Initial Node to Activity Diagram 'Login App'.	Login Ap

5) First, add an Initial Node, by clicking on the Initial Node ● icon in the Creator modeling toolbar, and then clicking anywhere on the canvas to place it.

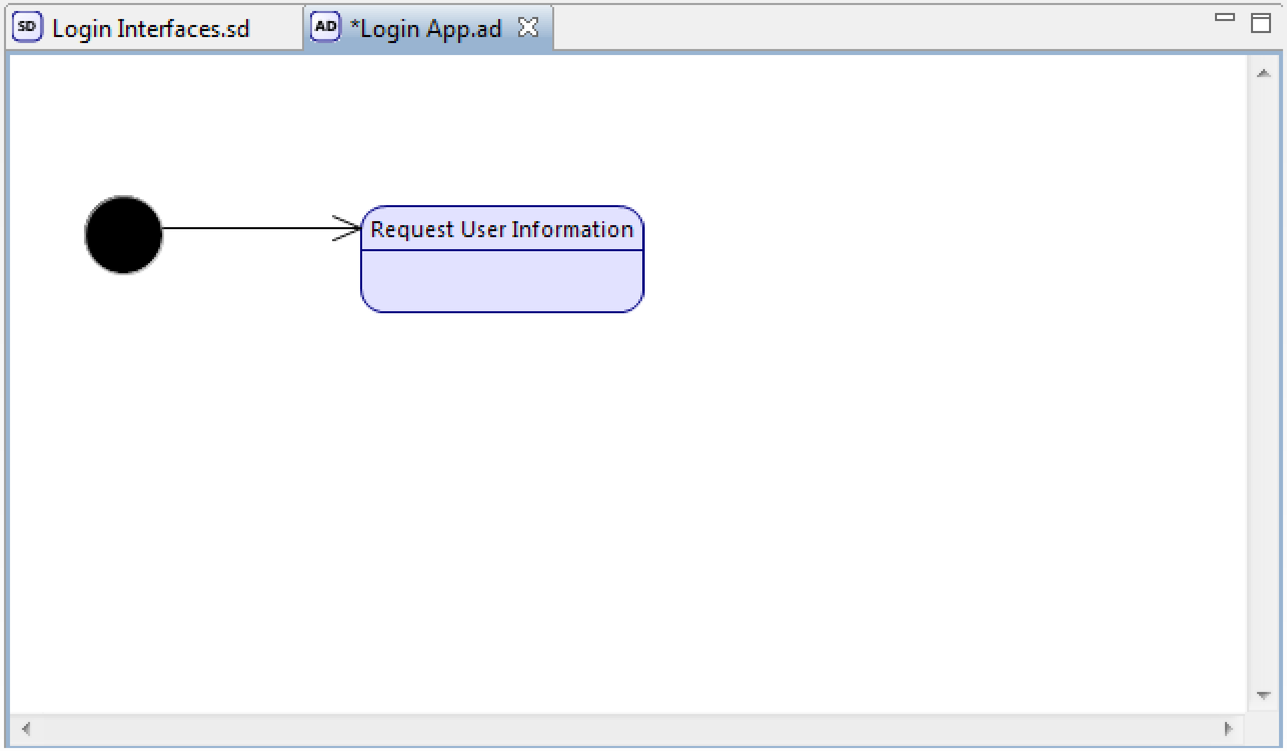
6) Next, add an Activity Node, by clicking on the Activity Node ▭ symbol in the symbol toolbar, and then clicking anywhere on the canvas to place it.

7) Type: “Request User Information”, as this is the first activity that takes place in the Login process. Click <*ENTER*>.



8) Next, connect the *Initial Node* to the *Activity Node*, by clicking on the *Control Flow* symbol ↗ from the Creator modeling toolbar, then click inside the *Initial Node*, and finally, drag it inside the *Activity* box.

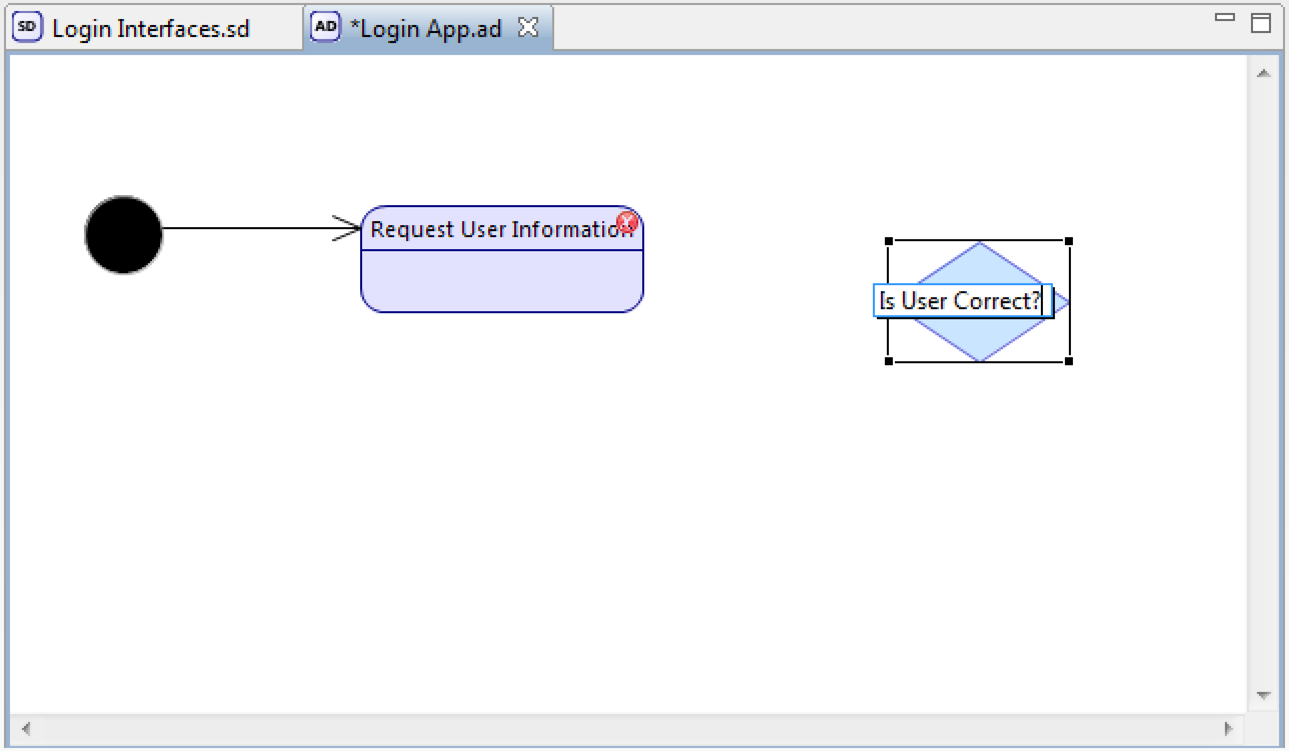
9) The *Initial Node* and the *Activity Node* should now be connected as below.



10) Next, add a *Decision Node*. Click the *Decision Node* D symbol on the Creator modeling toolbar, and then click anywhere on the canvas to place it.

11) Type "*Is User Correct?*" and then click <*ENTER*>

12) A *Decision Node* is added to the *Activity Diagram* as shown ➔



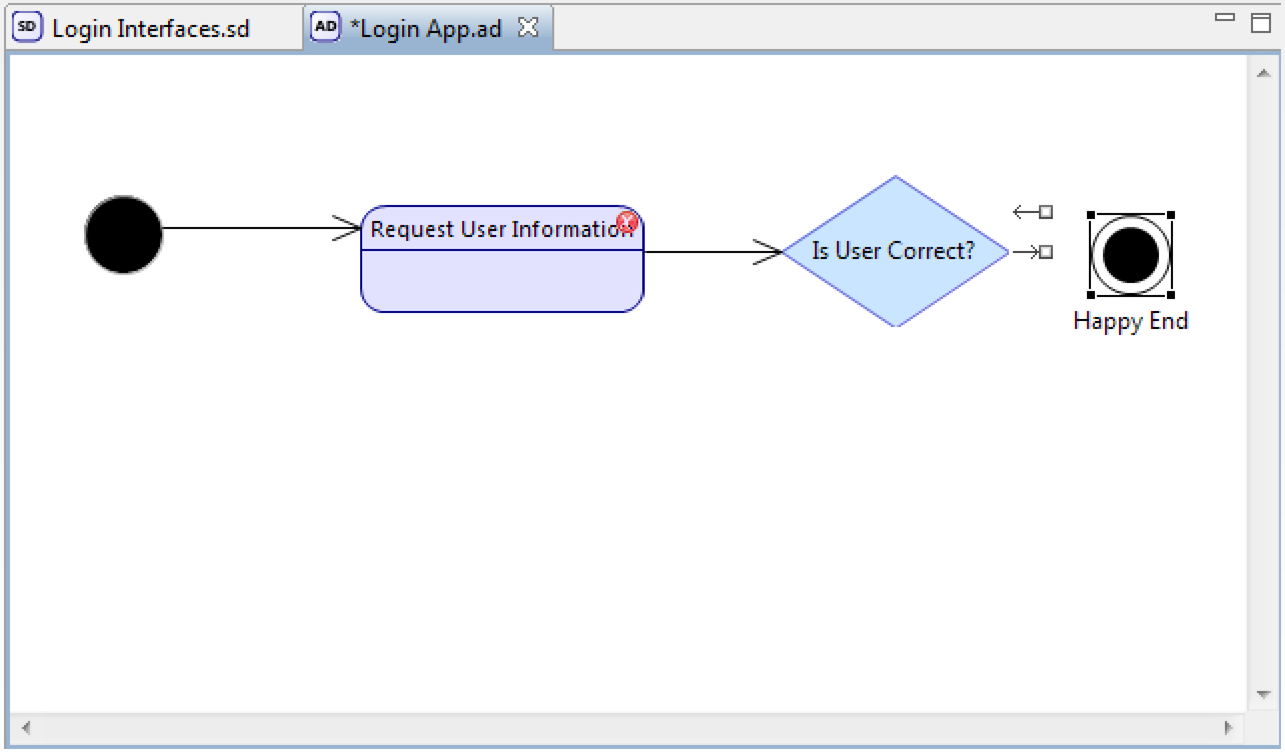
13) Connect the *Activity Node* and the *Decision Node*, using the *Control Flow* symbol ↗

14) Next, we'll add two *Final Nodes* coming out of the *Decision*: "*Happy End*" & "*Error*".

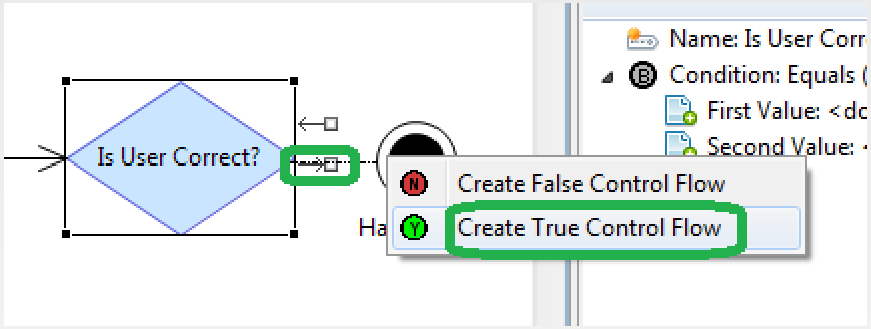
To do so:

15) Click on the *Final Node* symbol ◉ in the symbol toolbar, then on the canvas to place it there.

16) Type "*Happy End*" in the **Name** field, and click *OK*. A *Final Node* titled "*Happy End*" should be added to the *Activity Diagram*.



17) Now connect the *Control Flow Node* to the *Final Node*, by dragging the anchor displayed when hovering above the "Is User Correct?" Decision Node. Since you are trying to connect a Flow *Control Node*, you'd get the following dialog:



18) For this "*Happy End*" path, select the *Yes* Y option. The *Yes* Y control flow should be added to the *Activity Diagram*.

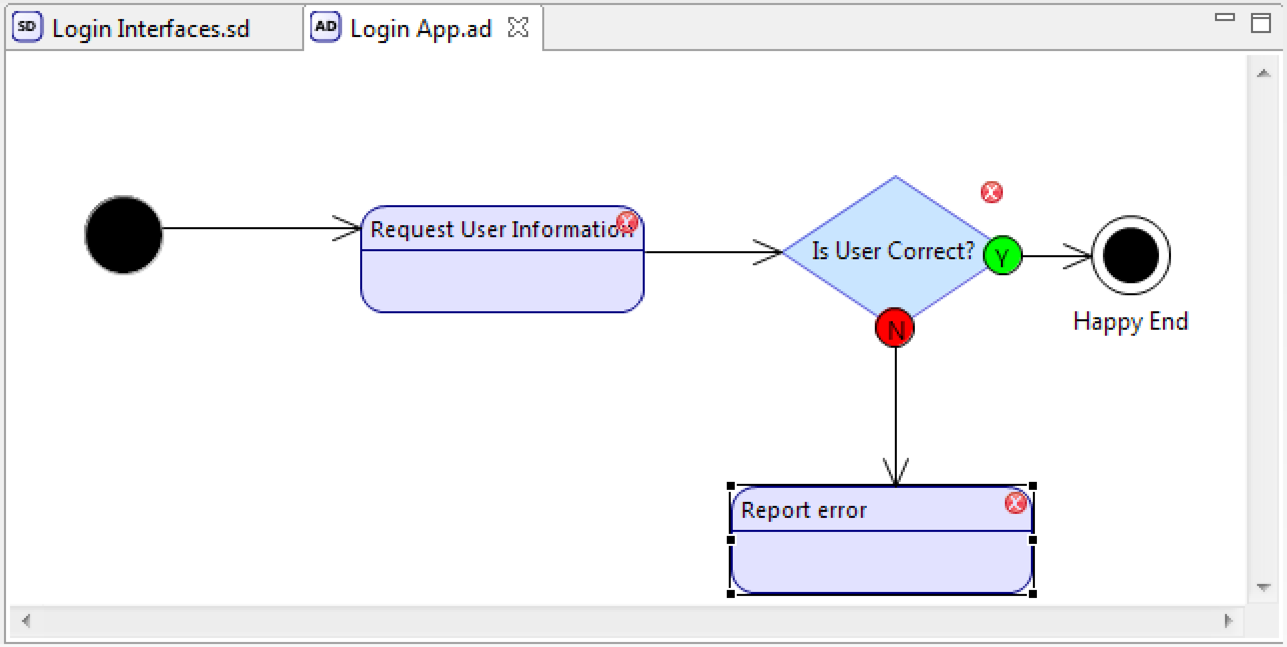
19) In case that the user is incorrect, lets report an error.

20) Add a new *Activity Node* titled "*Report error*". Look at previous steps if you need to refresh your memory.

21) "*Report error*" *Activity Node* should be added to the *Activity Diagram*.

22) Next, connect the "*Is User Correct?" Decision Node* to the "*Report error*" *Activity Node*. This time, pick the *No* Ⓝ option. The *No* Ⓝ control flow should be added to the *Activity Diagram*.

23) This is how your *Activity Diagram* should look so far:



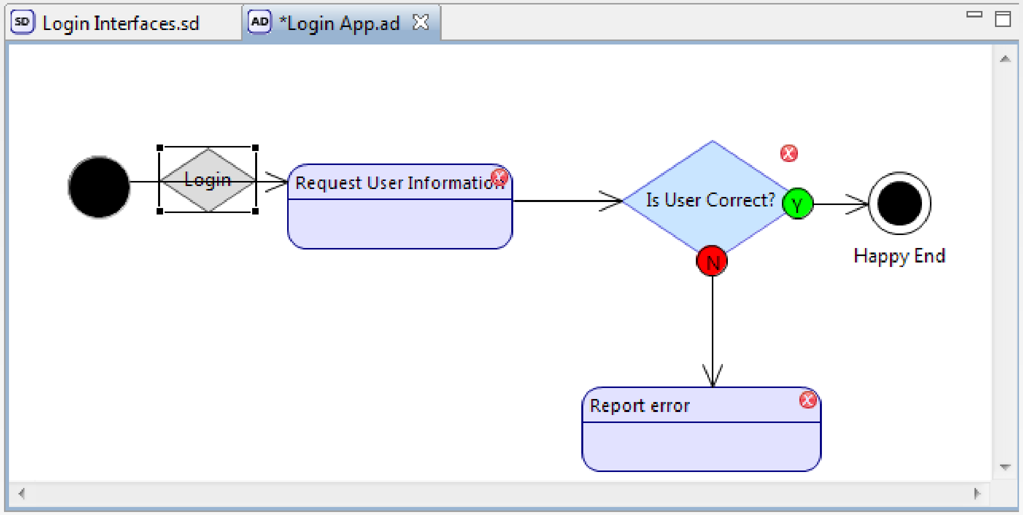
24) According to the AUT (Application under Test), if the user provides invalid login credentials, and an error has been reported, he or she is redirected to retry to login.

25) Therefore, let's loop back the "*Report error*" *Activity Node* back to the "*Request User Information*" *Activity Node*.

26) In this example we'll make the merging of *Control Flows* explicit, by using the *Merge* node.

27) Click on the *Merge* symbol from the modeling toolbar, and then click on the canvas to place it.

28) Type "*Login*" in the **Name** field, and then click <*ENTER*>. Click *Save*.
A *Merge Node* titled "*Login*" is now added to the *Activity Diagram*.



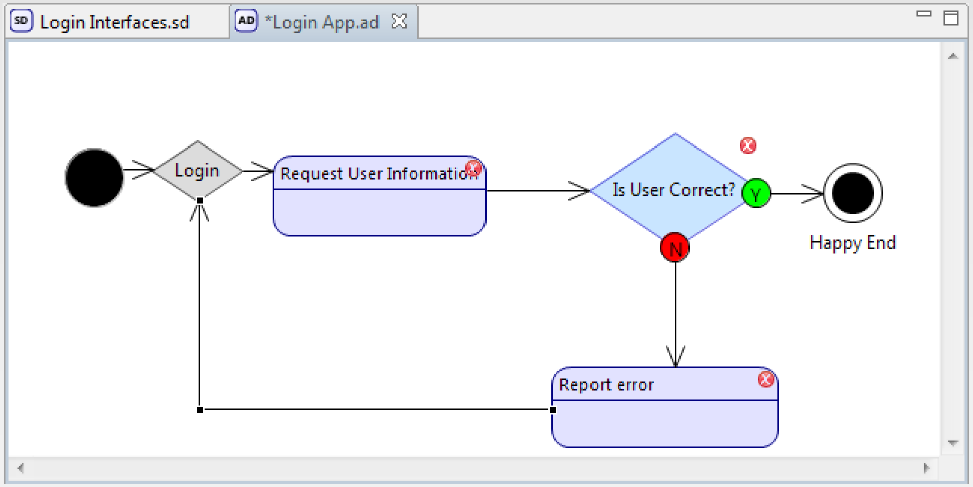
29) Now, we'll move and add some Flow Controls (connectors):

29.1) Move the existing Control Flow from the Initial Node to the "Request User Information" Activity - to "Login" Merge instead.

29.2) Draw a new Control Flow from "Login" Merge to "Request User Information" Activity.

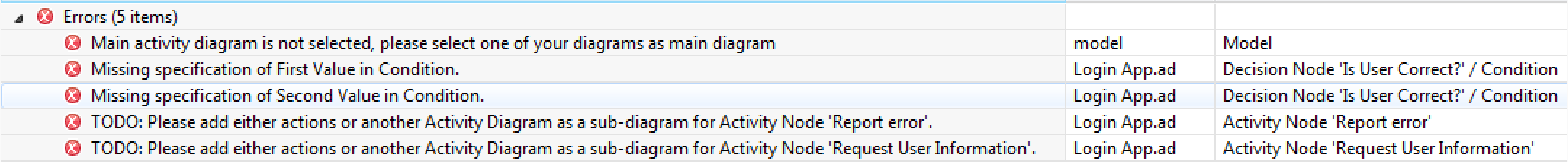
29.3) Draw a new Control Flow from "Report error" Activity to "Login" Merge.

30) You're *Activity Diagram* should look similar as below:



31) The TODO messages listed at the bottom need to go away before this *Activity Diagram* can actually generate test cases. We will discuss this topic in subsequent slides.

32) *Note that at this point we still have two* warnings / to-do's on this *Activity Diagram*:

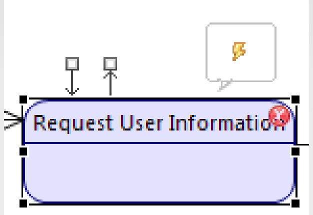


Errors (5 items)		
Main activity diagram is not selected, please select one of your diagrams as main diagram	model	Model
Missing specification of First Value in Condition.	Login App.ad	Decision Node 'Is User Correct?' / Condition
Missing specification of Second Value in Condition.	Login App.ad	Decision Node 'Is User Correct?' / Condition
TODO: Please add either actions or another Activity Diagram as a sub-diagram for Activity Node 'Report error'.	Login App.ad	Activity Node 'Report error'
TODO: Please add either actions or another Activity Diagram as a sub-diagram for Activity Node 'Request User Information'.	Login App.ad	Activity Node 'Request User Information'

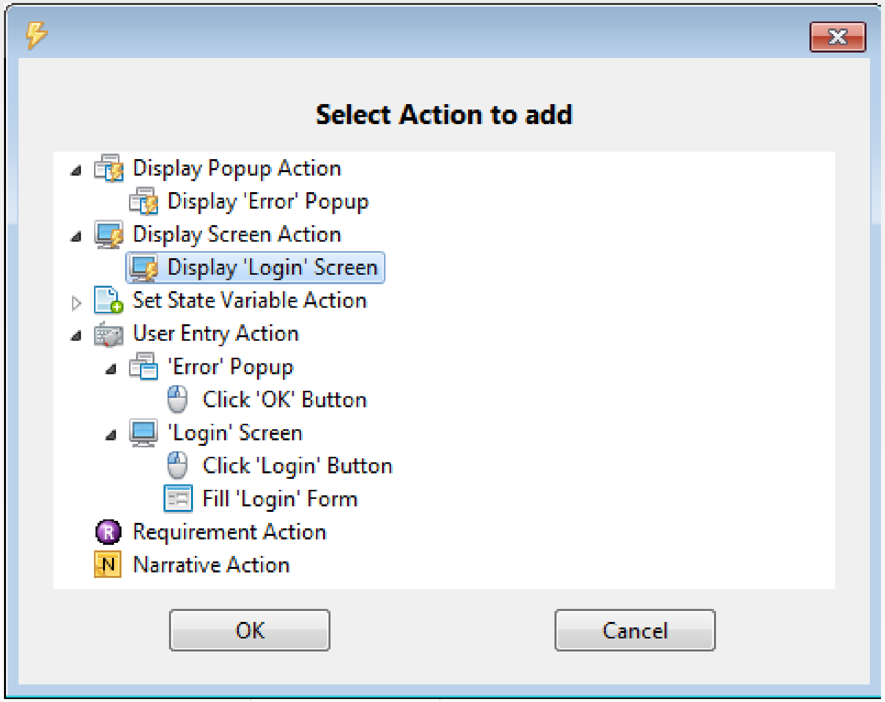
The above errors need to be removed before the *Activity Diagram* can automatically generate test cases.

Refining the *Activity Diagram* with Generated *Actions*

33) Hover over the “*Request User Information*” *Activity Node*, and then click on the lightning bolt icon.



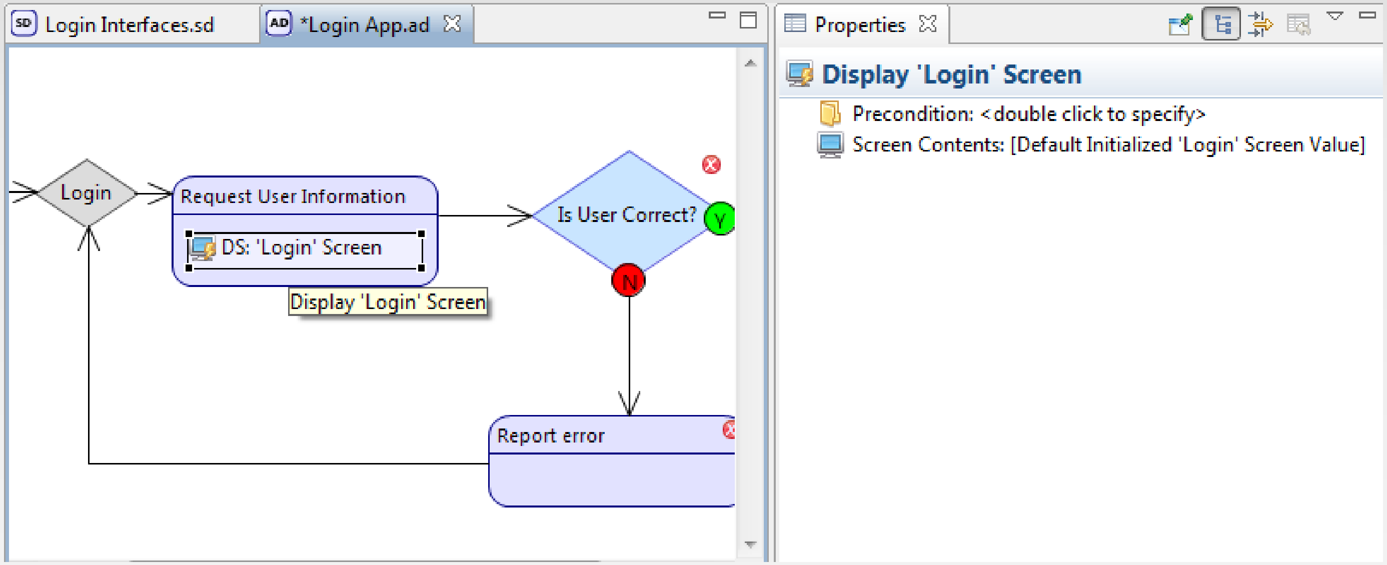
34) The *Action* dialog should appear as below:


Select Action to add

- Display Popup Action
  - Display 'Error' Popup
- Display Screen Action
  - Display 'Login' Screen
- Set State Variable Action
- User Entry Action
  - 'Error' Popup
    - Click 'OK' Button
  - 'Login' Screen
    - Click 'Login' Button
    - Fill 'Login' Form
- Requirement Action
- Narrative Action

35) The first *Action* in this *Activity* is that the system displays the Login screen to the user. Select the “*Display Login Screen*” *Action*, and click <*ENTER*>.

36) The *Property* View shows the "*Display Login Screen"* Action added.



Display 'Login' Screen

Precondition: <double click to specify>

Screen Contents: [Default Initialized 'Login' Screen Value]

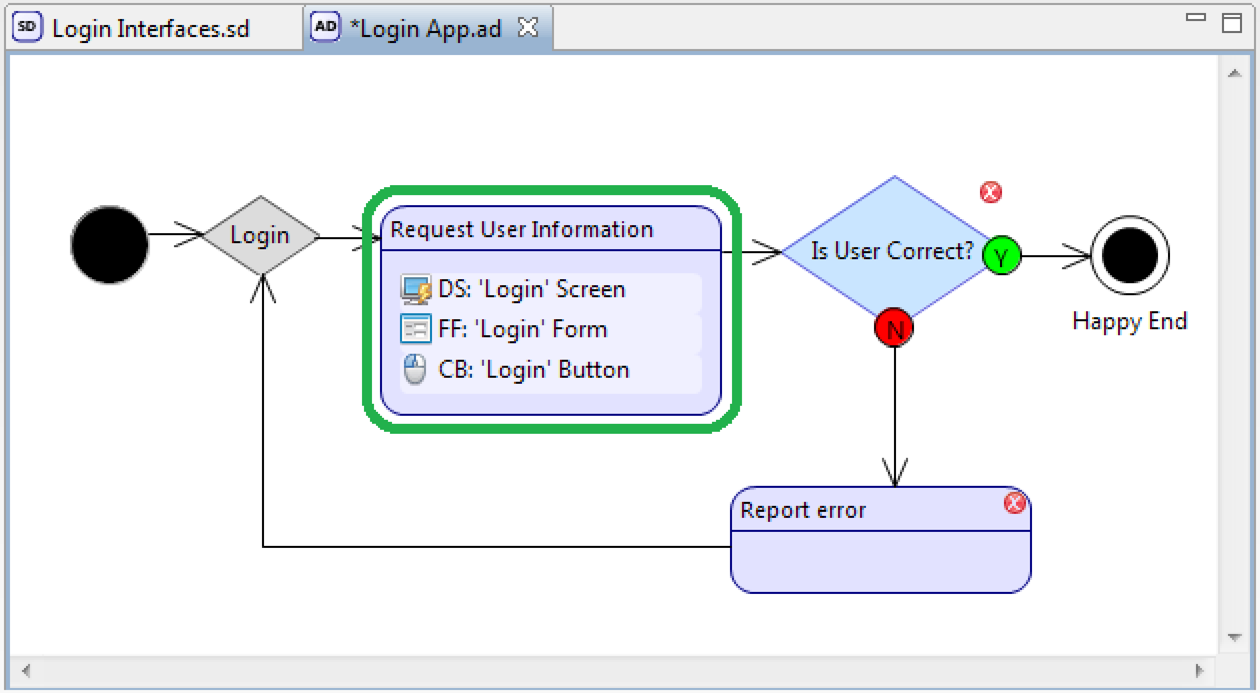
37) Next, hover again over the activity and click on the lightning bolt. The *Action* dialog should appear.

38) Select "*User Entry Action > Login Screen > Fill Login Form*" *Action* and click <*ENTER*>. The "*Fill Login Form [Login Screen]*" *Action* is now added to the list.

39) The last *Action* on this *Activity*, after the user fills out the Login Form, is for the user to click on the Login button. Let's add this *Action*.

40) Add the "*User Entry Action > Login Screen > Click Login Button*" *Action* similarly to the previous steps.

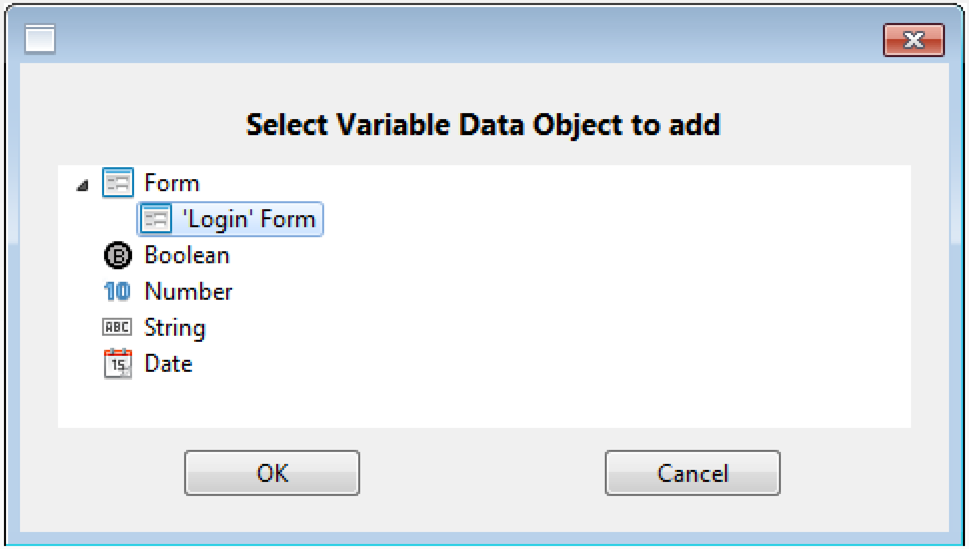
41) The “Request User Information” Activity should now look like this:



Note: *DS*, *FF* and *CB* are abbreviations used for *Display Screen*, *Fill Form*, and *Click Button*.

42) Next, you'll add some logic to the "Is User Correct?" Decision. How would you determine if the data the user entered is valid or not? We need to store the data somewhere and then compare it with some expected data.

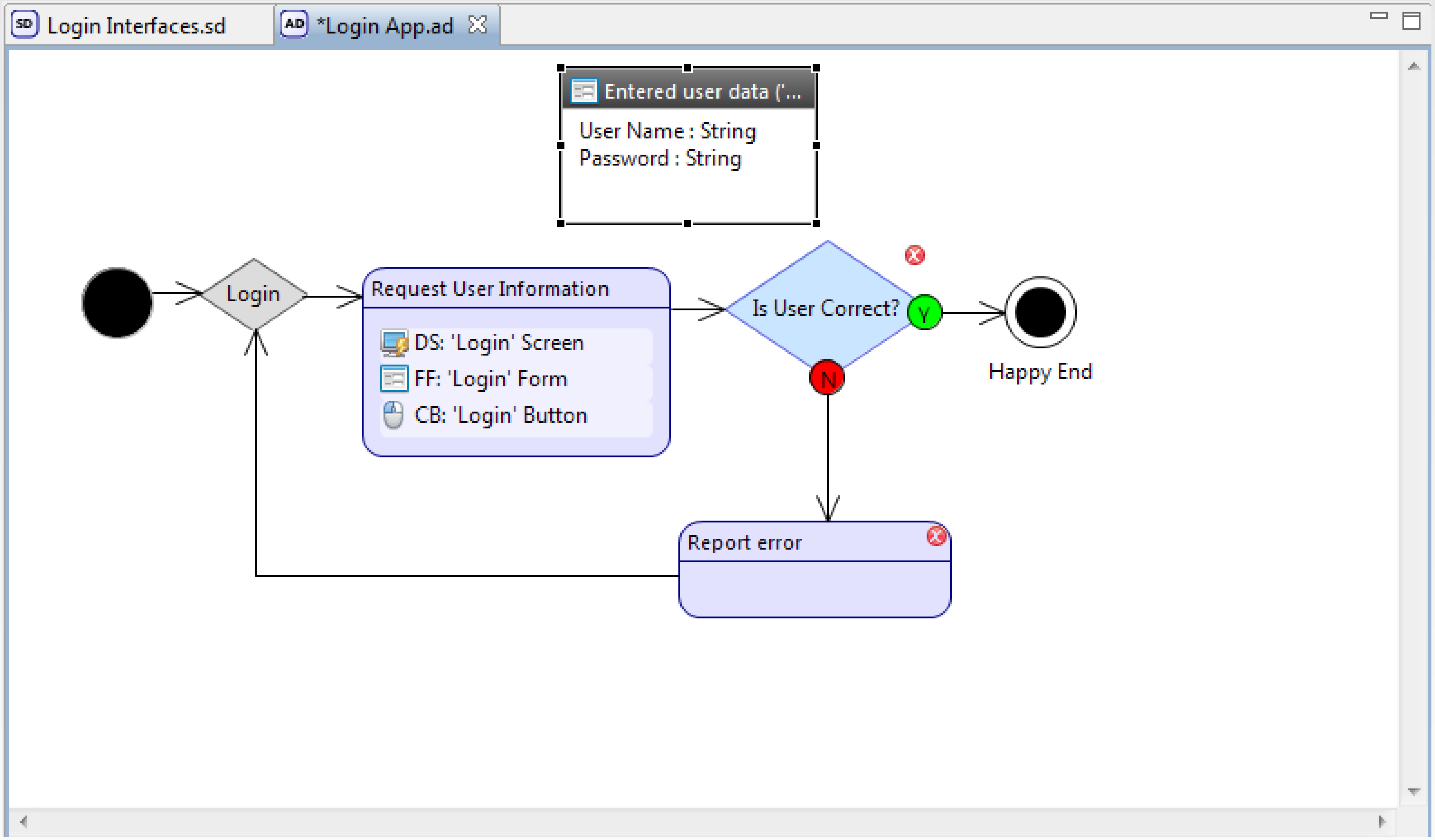
43) Let's add some variables, by clicking on the Variable Data Object symbol ▭ in the modeling toolbar, and then click inside the canvas to place it.
The Variable Data Object selection dialog should appear as below:



Select Variable Data Object to add

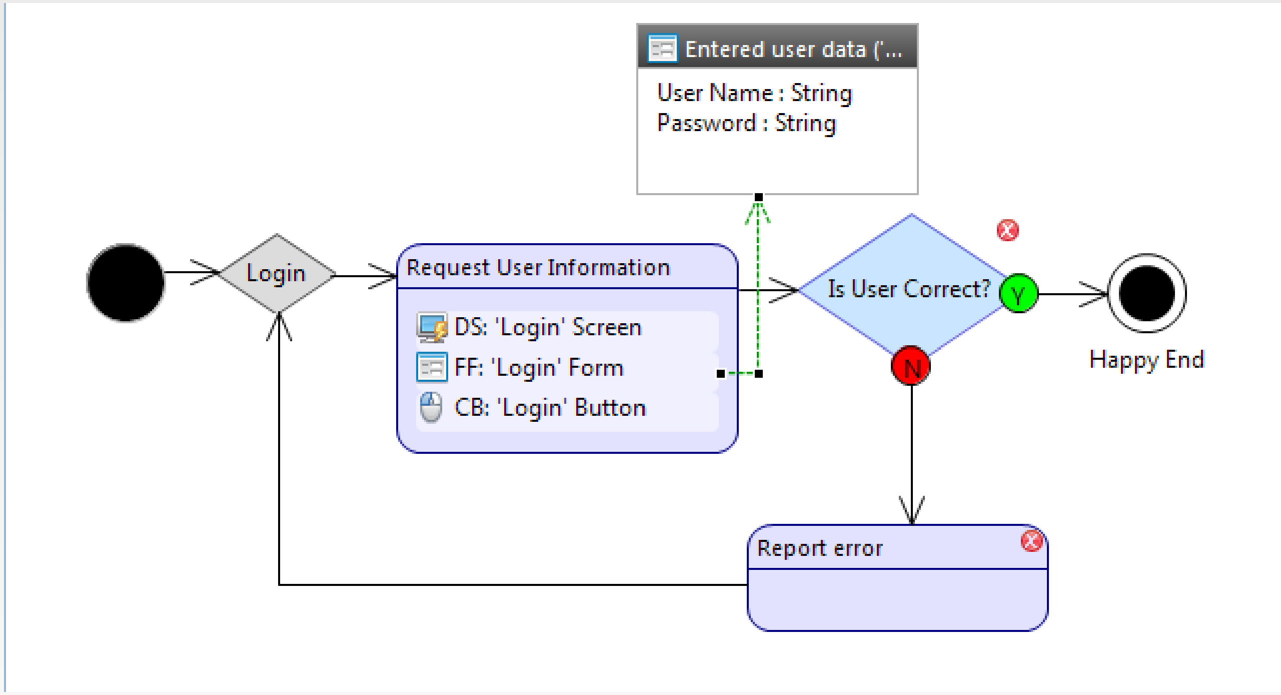
- Form
  - 'Login' Form
- Boolean
- Number
- String
- Date

44) Select "*Login Form Variable*". Type in the **Variable Name** field: "*Entered user data*", and click <*ENTER*>



45) Next, we need to point out where this *Data* object is produced. The form's *username* and *password* values should be stored when the user fills out the Login form.

46) Click on the *Data Flow symbol* , then click on the "*FF: Login*" area inside the "*Request User Information*" *Activity*, then click again at the top bar of the "*Entered user data*" variable *Data* object, to connect the two as shown below:



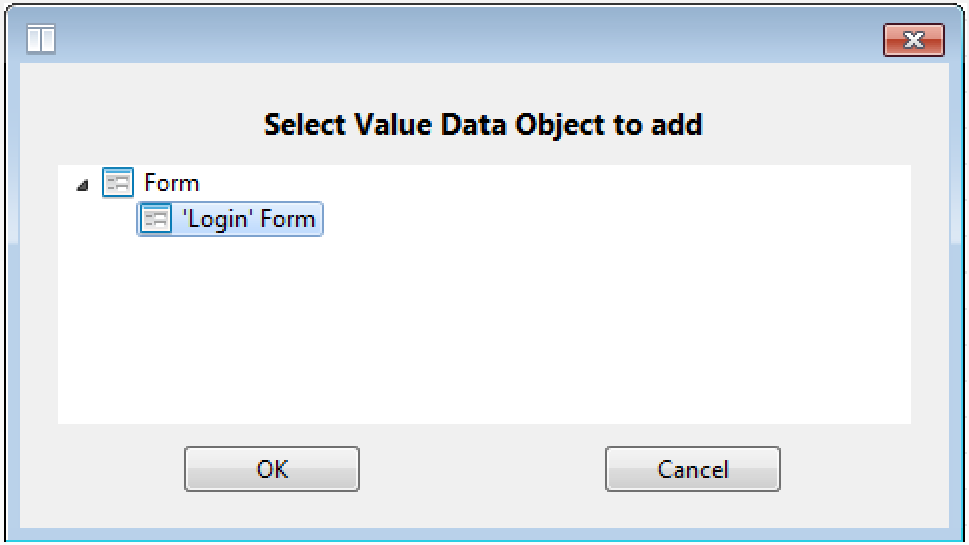
47) Next, we'll map the "*Entered user data*" variable data object to the "*Is User Correct?*" *Decision*. You can tie individual variables (*username* or *password* in this case) or the entire object (all variables).

48) Let's tie the entire object, for comparing both the *username* and *password* between the stored form data and the expected data for the decision making.

49) Click again on the *Data Flow symbol* ↗, then click inside the top bar of the "*Entered user data*" variable object, and then click inside the "*Is User Correct?*" *Decision* object.

50) Next, we'll define the expected data for the *username* and *password* to compare against in the *Decision*. In other words, in this example we are going to compare the stored data against some hard-coded values.

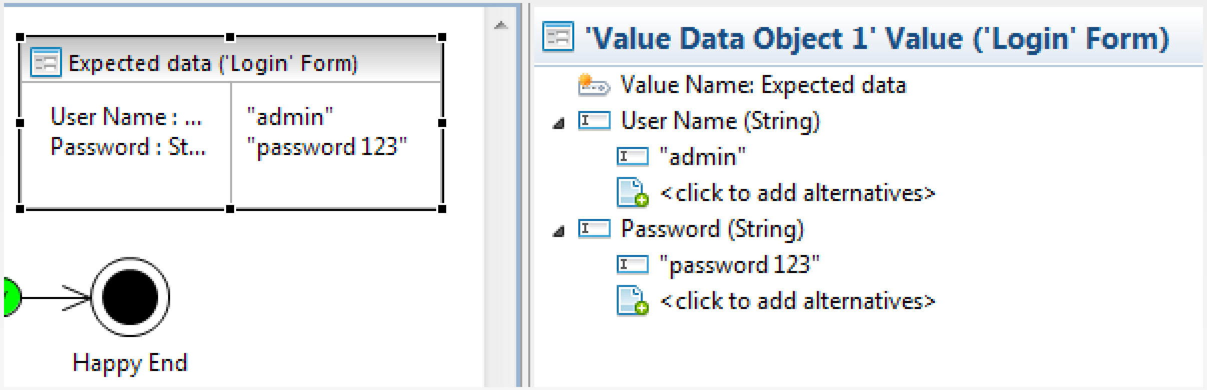
51) Click on the *Value Data* object symbol ◫ on the modeling toolbar, and then click inside the canvas. The *Value Data Object* dialog should open as below. A list of elements should appear.



Select Value Data Object to add

- Form
  - 'Login' Form

52) Select “*Login Form Value*”. Type in the **Value Name**: “*Expected data*”, type in the **User Name**:, “*admin*”, and **Password**: “*password123*”, and click *<ENTER>*.
53) A Login Form Value object is added to your *Activity Diagram as shown below*.


Expected data ('Login' Form)

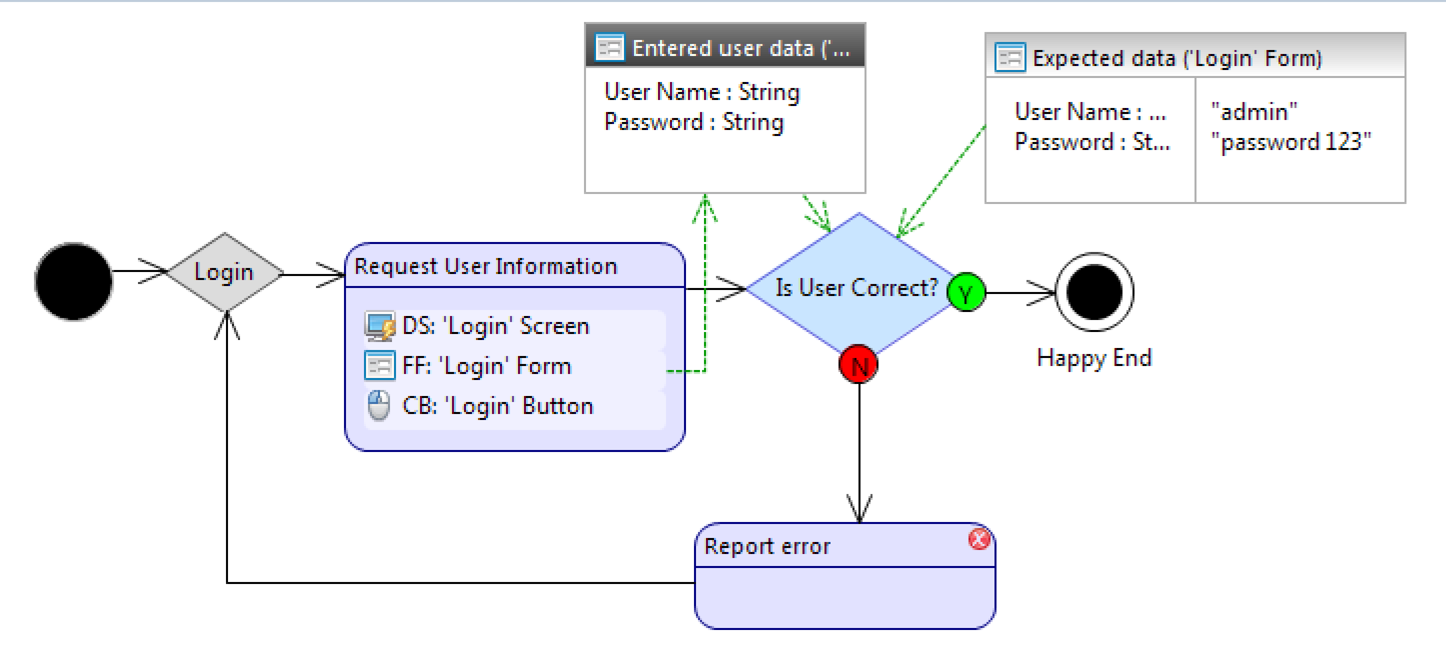
User Name : ... Password : St...	"admin" "password 123"

'Value Data Object 1' Value ('Login' Form)
- Value Name: Expected data
- User Name (String)
  - "admin"
  - <click to add alternatives>
- Password (String)
  - "password 123"
  - <click to add alternatives>

54) Using a *Data* Flow symbol ↗ connect the "*Expected data*" object to the "*Is User Correct?*" *Decision*. Click *Save*.

55) We've now defined a condition that if the *username* equals to "*admin*" and the *password* equals to "*password123*", then the *Decision* outcome is *Yes* Ⓨ and connects into the "*Happy End*" *Final Node*, otherwise the *Decision* outcome is *No* Ⓝ, and connects to the "*Report error*" *Activity*.

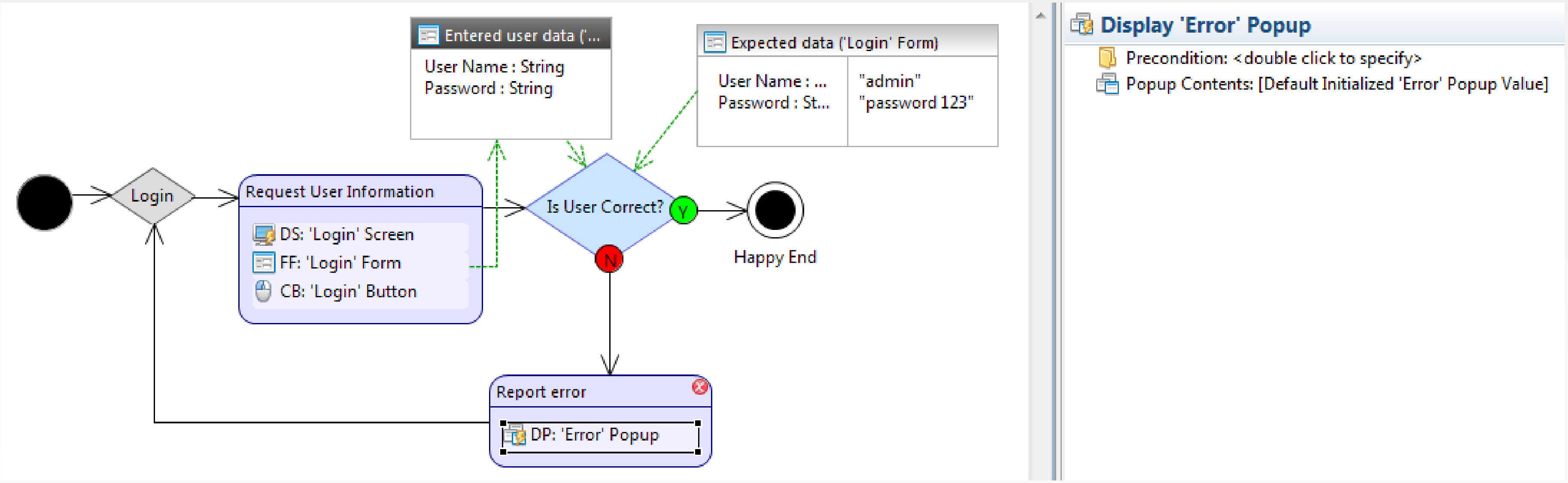
➔ At this point, your *Activity Diagram* should look similar to this:



56) As you can guess from the last TODO error in the *Problems View*, we still need to define an *Action* for the "*Report error*" *Activity*.

57) As before, in order to do so, hover the mouse over the "*Report error*" *Activity*, click on the lightning bolt.

58) Select "*Display Error Popup*" *Action*. A *"Display Error Popup" Action* should be added to the **Actions** in the "*Report error*" *Activity*.

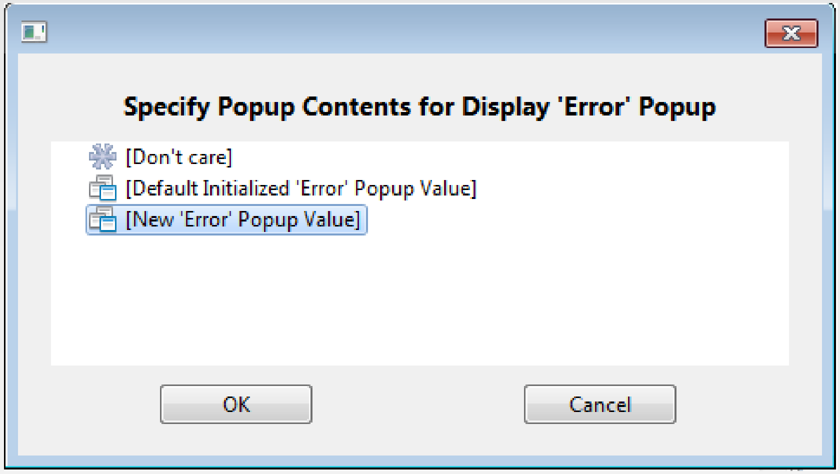


Display 'Error' Popup

Precondition: <double click to specify>

Popup Contents: [Default Initialized 'Error' Popup Value]

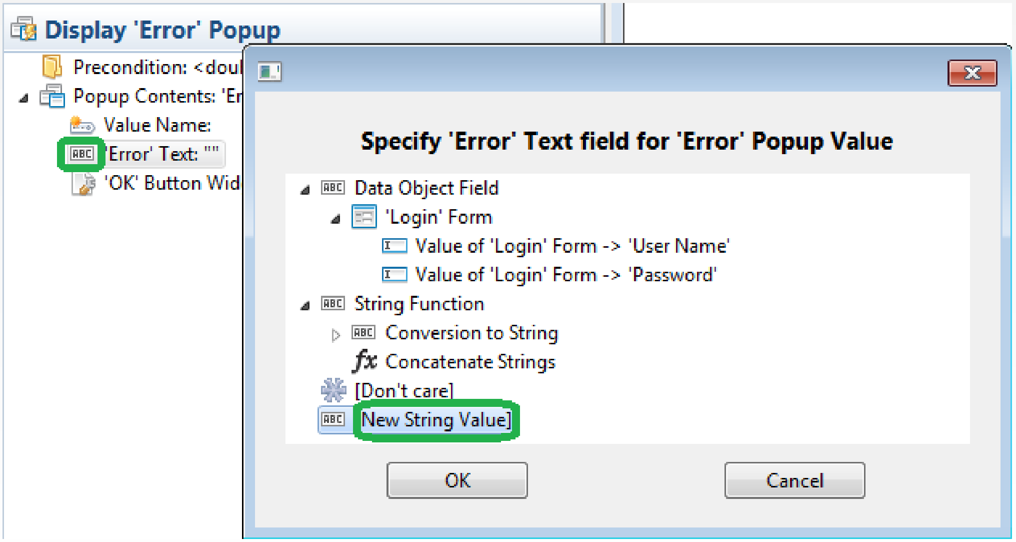
59) Click on the action in the activity diagrams. Double click on the Popup Contents Field in the property view. Select *New Error Popup Value*



Specify Popup Contents for Display 'Error' Popup

[Don't care]
[Default Initialized 'Error' Popup Value]
[New 'Error' Popup Value]

60) We could single click now on the **Error Text** field and simply type our text into the field. Instead let us double click and we get to a *Text Selection* dialog.

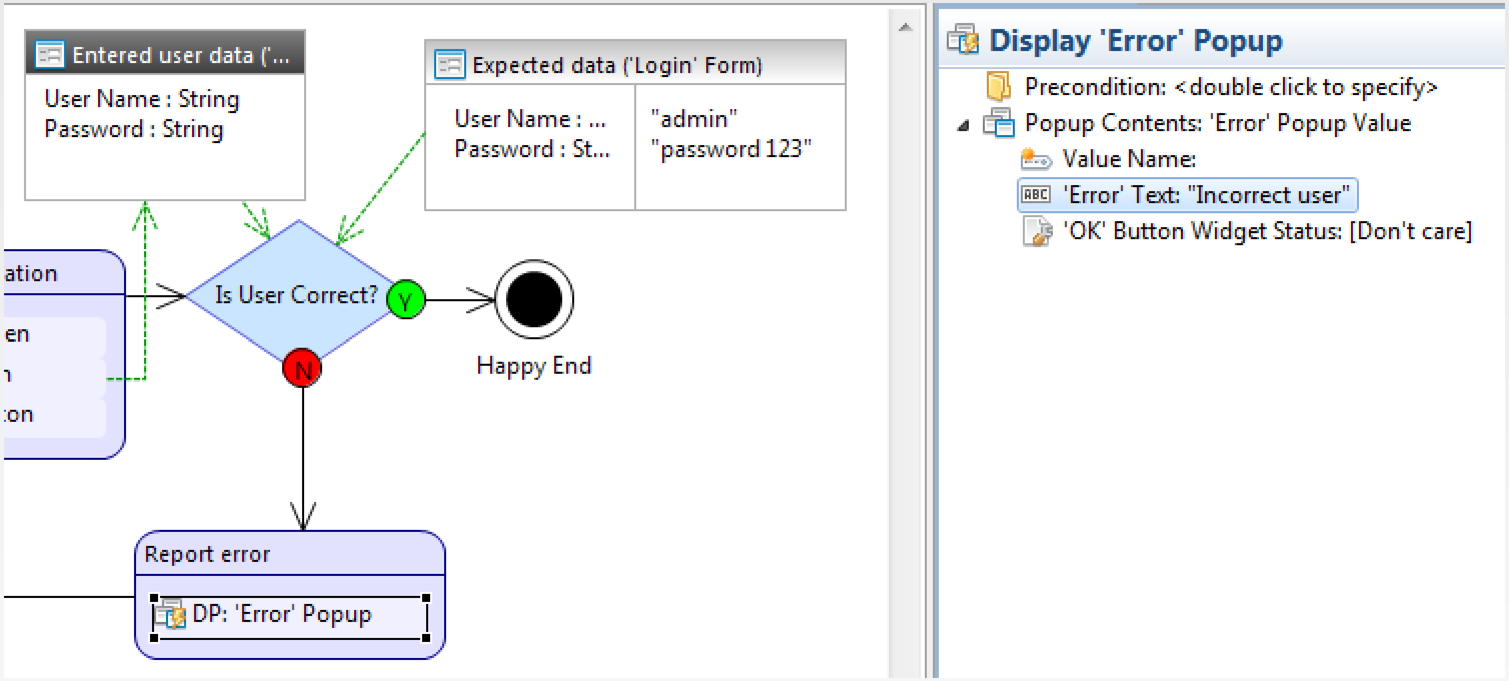


Display 'Error' Popup
Precondition: <doul
Popup Contents: 'Er
Value Name:
'Error' Text: ""
'OK' Button Wid

Specify 'Error' Text field for 'Error' Popup Value

Data Object Field
'Login' Form
Value of 'Login' Form -> 'User Name'
Value of 'Login' Form -> 'Password'
String Function
Conversion to String
Concatenate Strings
[Don't care]
[New String Value]

61) Select *String Value* and click *OK*. *String Value* element is now used for the **Error Text** property. Type: “*Incorrect user*”. Click *Save*.



Display 'Error' Popup

- Precondition: <double click to specify>
- Popup Contents: 'Error' Popup Value
  - Value Name:
  - 'Error' Text: "Incorrect user"
  - 'OK' Button Widget Status: [Don't care]

62) Notice that the last TODO message went away. You’ve just completed refining your *Activity Diagram* and almost ready to generate test cases.

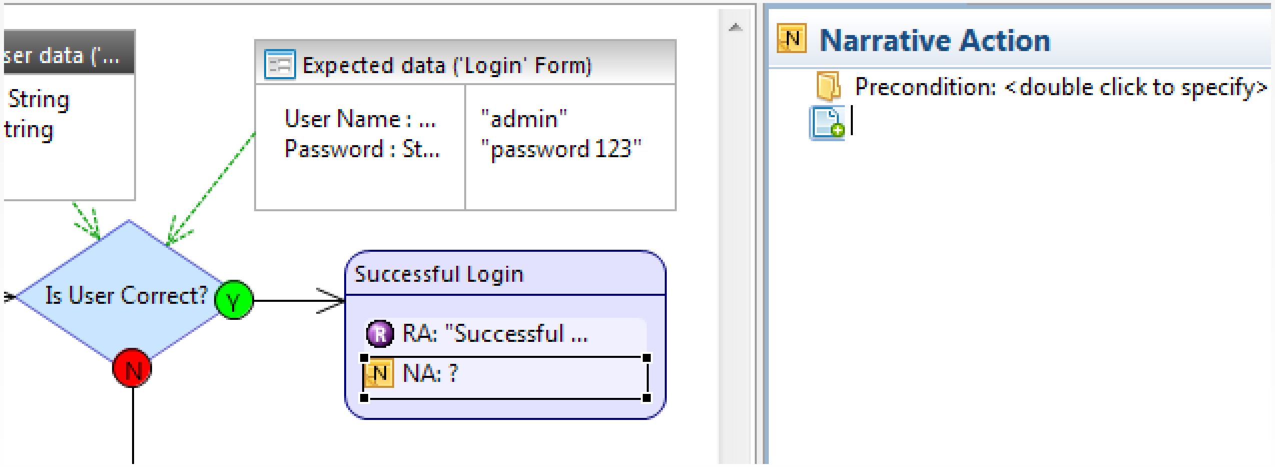
63) In order to ensure proper coverage and descriptive test cases, you need to add *Requirements* and *Narratives* to the model.

64) In our example, we want to generate test cases for both valid and invalid login attempts.

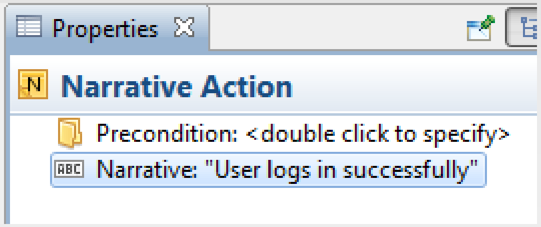
65) Replace the "Happy End" Final node with a "*Successful Login*" *Activity Node*, then click on the lightening bolt, …Select a *Requirement Action*.

66) In the Requirement Identifier edit field type "*Successful Login*", and then click <*ENTER*>.

67) Now add another *Action*, this time select a *Narrative Action*. Click on the **Narrative**: edit field.



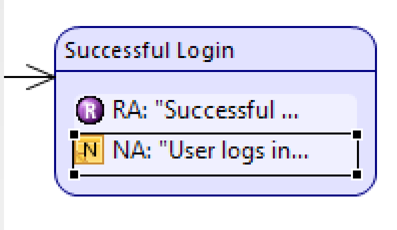
68) Type in the **Narrative** field: *“User logs in successfully”*.



Precondition: <double click to specify>

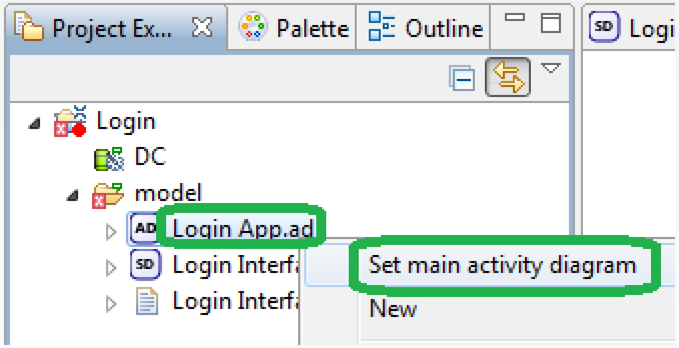
Narrative: "User logs in successfully"

69) Your new *Activity* Node should now have one *Requirement Action* and one *Narrative Action* ➔



70) Similarly, add a *Requirement Action* and a *Narrative Action* to the *Report Error Activity Node*.

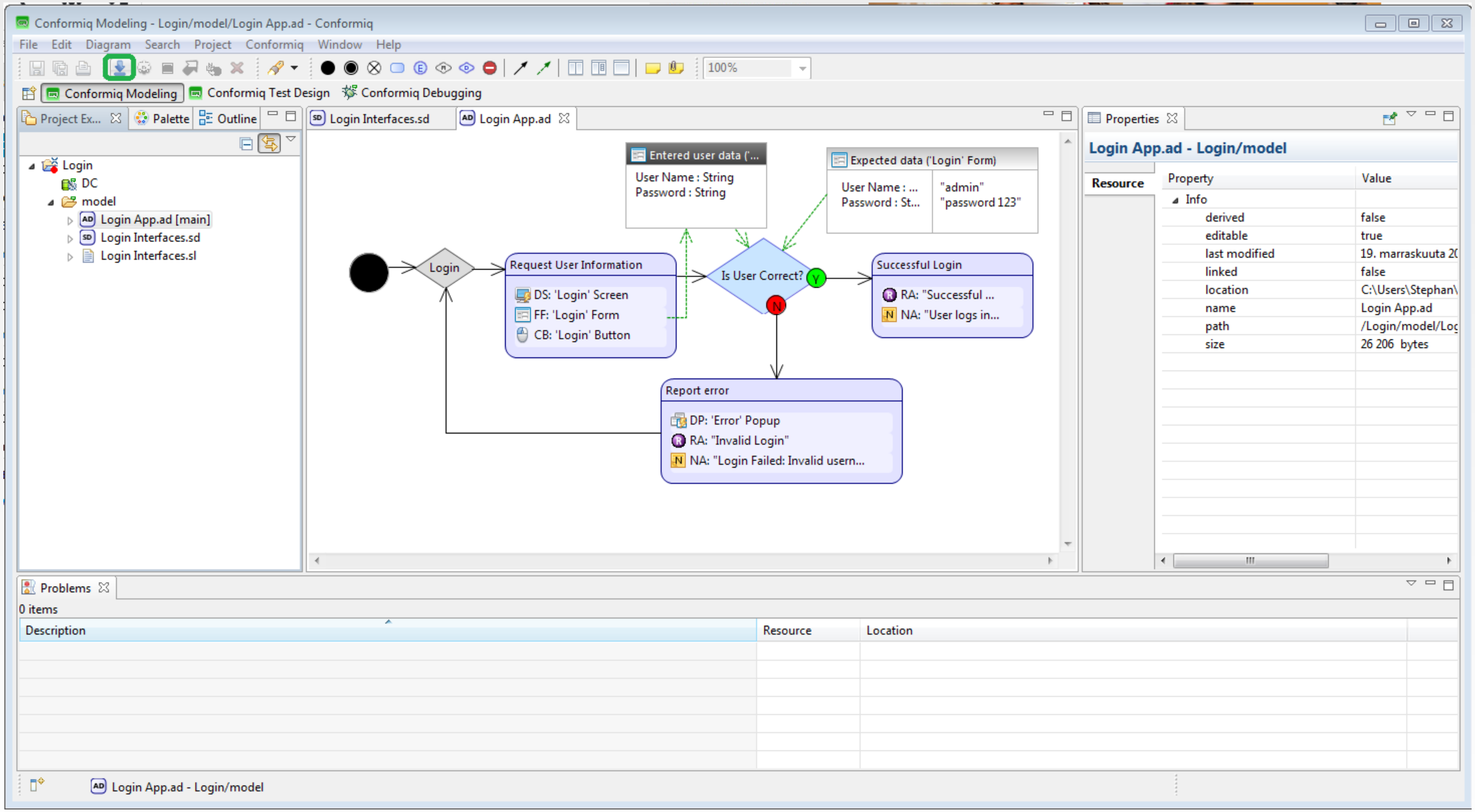
71) Finally set in the *Project Explorer* the ”Login App” as your ”*main activity diagram*”



71) Now you are ready to generate test cases from this Activity Diagram.

Generating *Test Cases* from an *Activity Diagram*

1) Click “Load Model” in the Creator tool bar.

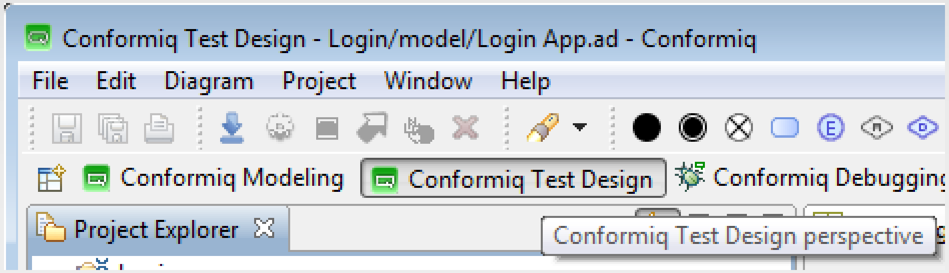


Property	Value
Info	
derived	false
editable	true
last modified	19. marraskuuta 2(
linked	false
location	C:\Users\Stephan\
name	Login App.ad
path	/Login/model/Log
size	26 206 bytes

0 items

Description	Resource	Location

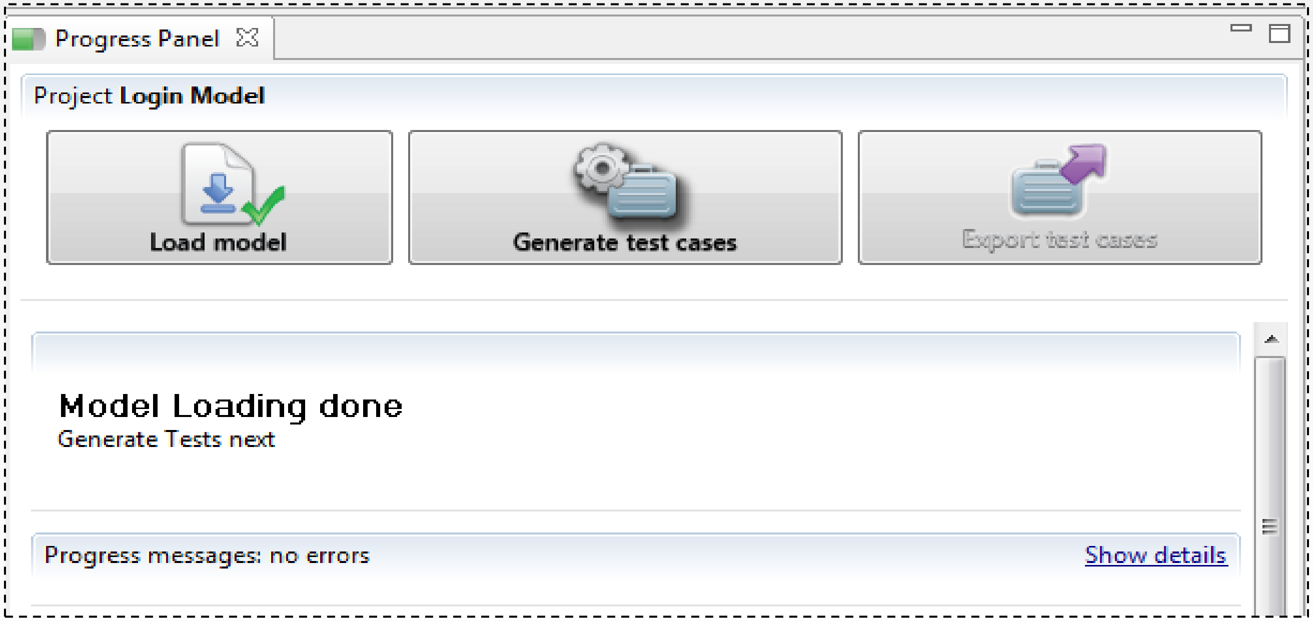
2) Switch to the *Conformiq Test Design* Perspective



Conformiq Test Design - Login/model/Login App.ad - Conformiq
File Edit Diagram Project Window Help
Conformiq Modeling
Conformiq Test Design
Conformiq Debugging
Project Explorer
Conformiq Test Design perspective

3) In the *Progress Panel*, make sure there are no errors detected and the model loading is completed.

4) The Tick mark in Green color is a sign for successful completion of model loading.

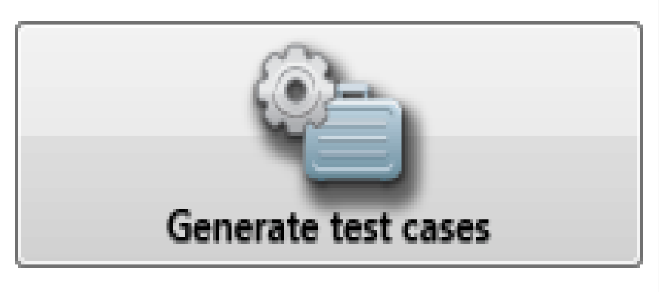


Model Loading done
Generate Tests next

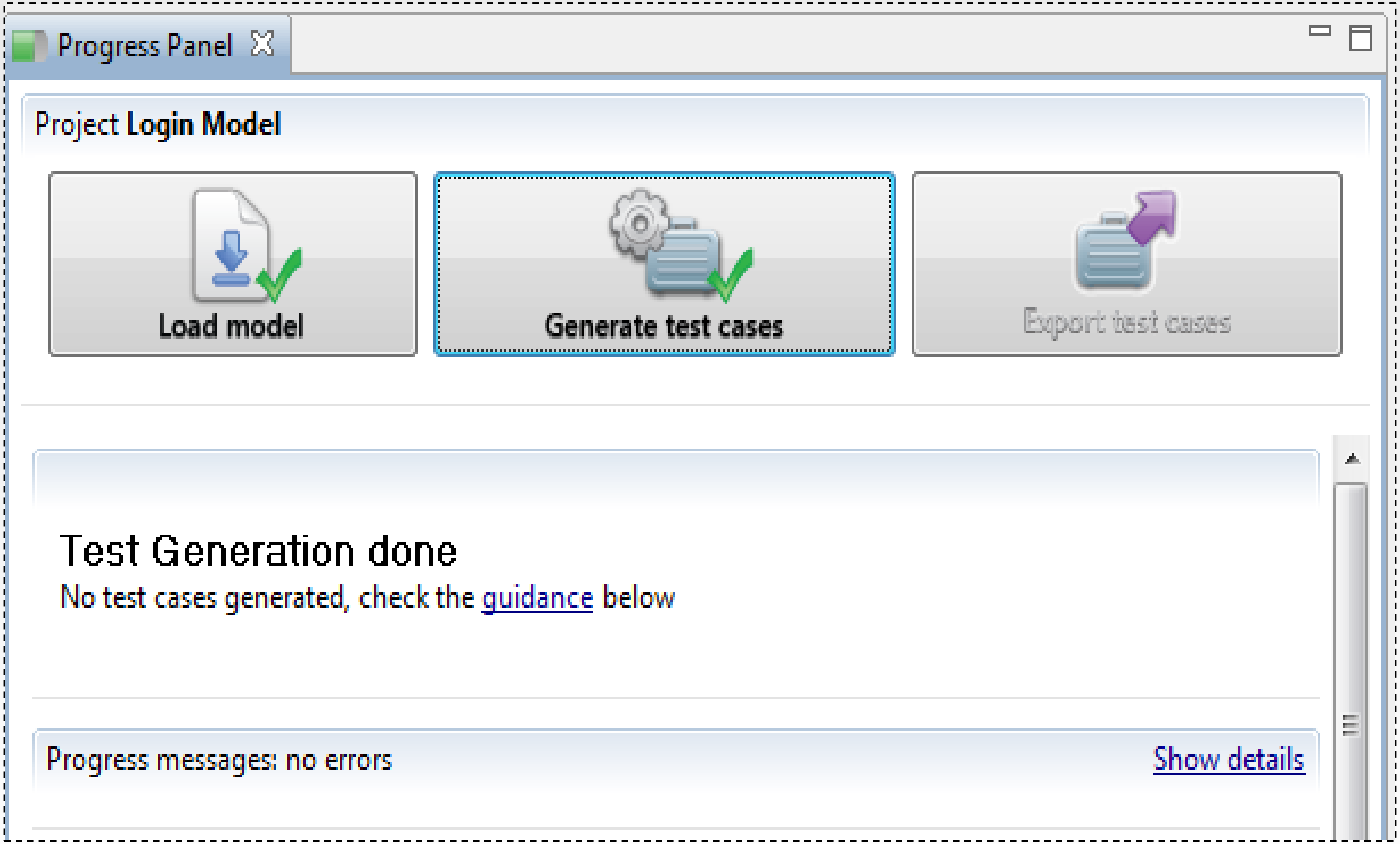
Progress messages: no errors

5) Next, you will have *Conformiq Creator* auto-generate test cases from this model

6) In the *Progress Panel*, click on **Generate test cases** button .



7) Make sure the *Progress Panel* did not detect any errors. Again, the Tick mark in Green color is a sign for successful completion of model loading.



Project **Login Model**

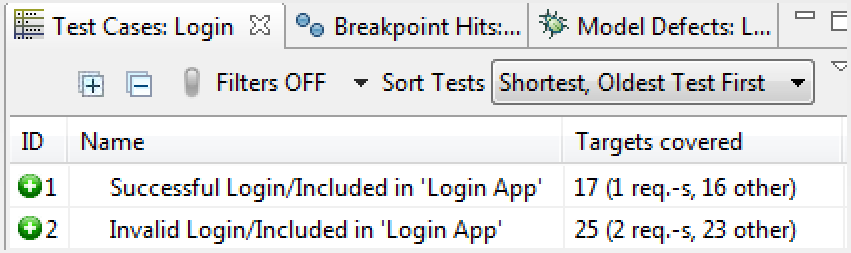
Test Generation done

No test cases generated, check the guidance below

Progress messages: no errors

Analyzing *Test Cases* in *Conformiq Designer*

1) Take a look at the *Test Case* view to see the test cases that were generated. You should see these:



Test Cases: Login | Breakpoint Hits:... | Model Defects: L...

Filters OFF | Sort Tests | Shortest, Oldest Test First

ID	Name	Targets covered
1	Successful Login/Included in 'Login App'	17 (1 req.-s, 16 other)
2	Invalid Login/Included in 'Login App'	25 (2 req.-s, 23 other)

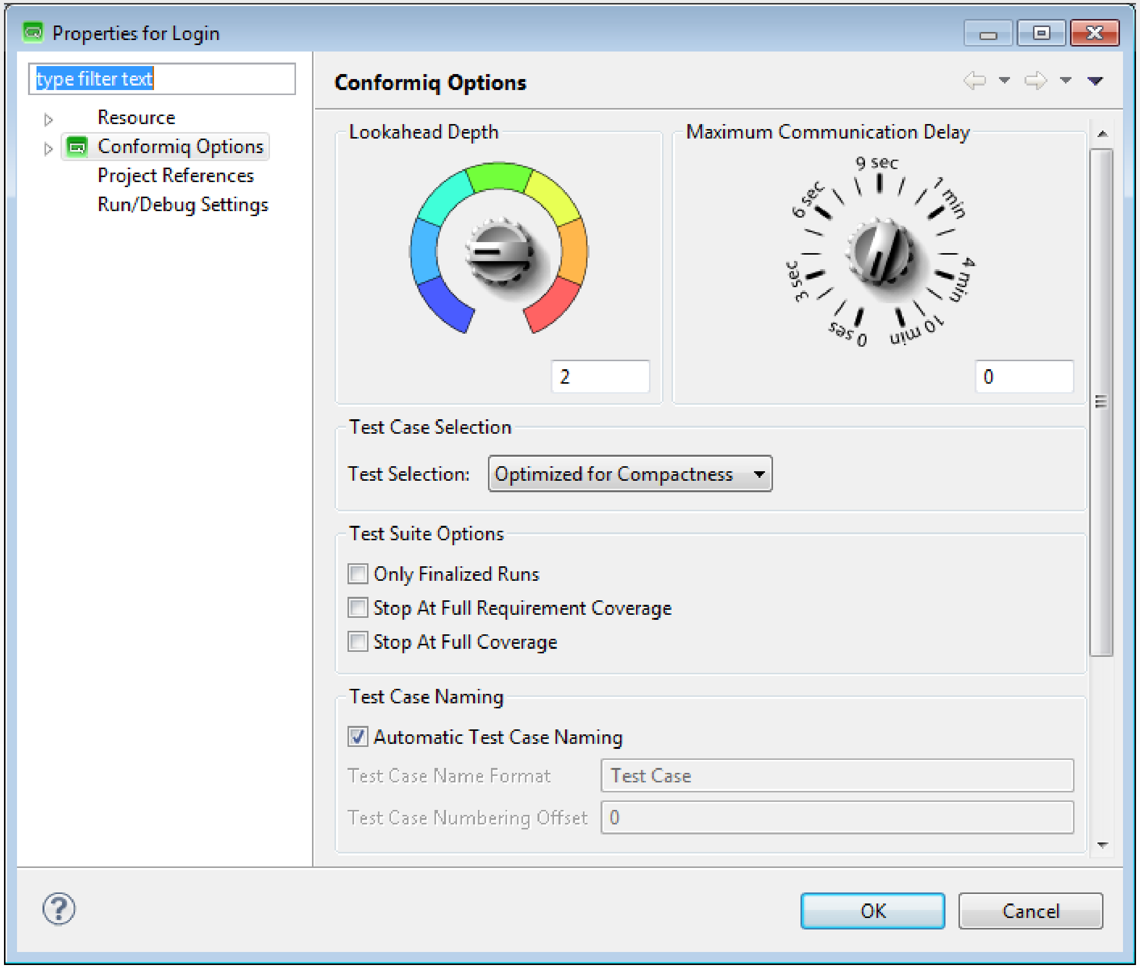
Note: Test case names were constructed from the *Requirement Actions* you've added to the model.

Tip: If you don't see test cases in the *Test Case* View, this can be due to some default settings. To fix that, in the *Project Explorer*, right-click on the model name "*Login Model*", and select *Properties*.

The *Properties* for the "*Login Model*" window should be displayed. Click on the *Conformiq Options* tab.

2) Change the ***Lookahead Depth*** value to 2 (as shown below), click *OK* to save and close the *Properties* Window.

3) Reload and *regenerate* test cases (refer to the steps above if you need assistance). Now you should be getting two test cases as shown above.



Properties for Login

type filter text

Conformiq Options

Lookahead Depth: 2

Maximum Communication Delay: 0

Test Case Selection

Test Selection: Optimized for Compactness

Test Suite Options

☐ Only Finalized Runs
☐ Stop At Full Requirement Coverage
☐ Stop At Full Coverage

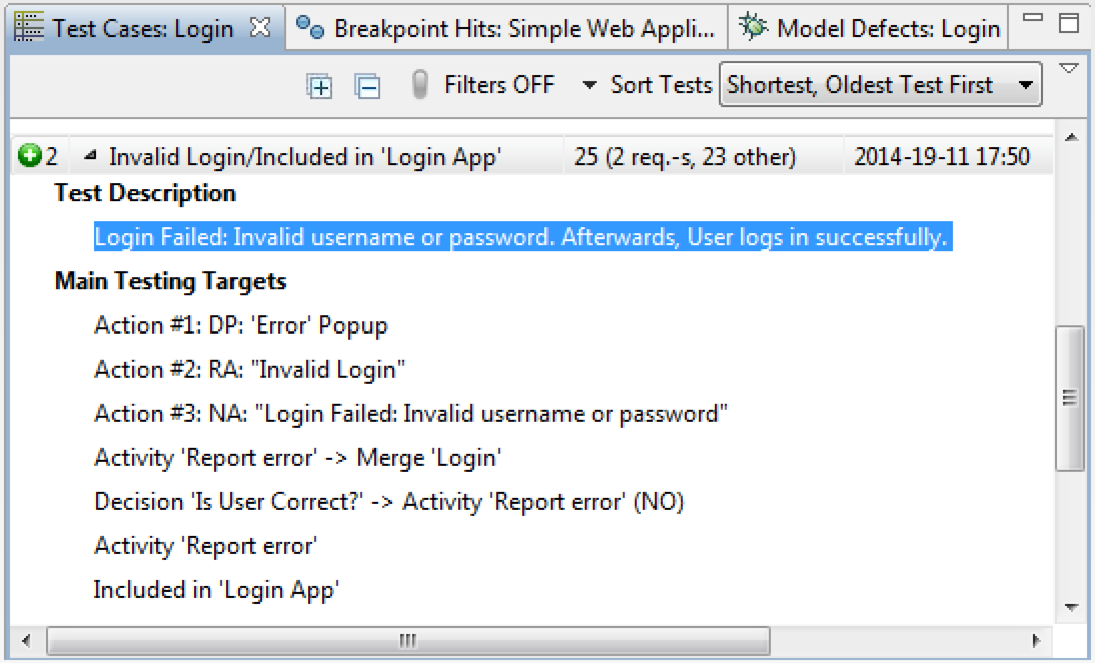
Test Case Naming

☑ Automatic Test Case Naming

Test Case Name Format: Test Case

Test Case Numbering Offset: 0

4) Click the expand arrow next to the test case names. You should see a description associated with the test cases as below:


Test Cases: Login
Breakpoint Hits: Simple Web Appli...
Model Defects: Login
Filters OFF
Sort Tests
Shortest, Oldest Test First

2 Invalid Login/Included in 'Login App' 25 (2 req.-s, 23 other) 2014-19-11 17:50

Test Description

Login Failed: Invalid username or password. Afterwards, User logs in successfully.

Main Testing Targets

Action #1: DP: 'Error' Popup

Action #2: RA: "Invalid Login"

Action #3: NA: "Login Failed: Invalid username or password"

Activity 'Report error' -> Merge 'Login'

Decision 'Is User Correct?' -> Activity 'Report error' (NO)

Activity 'Report error'

Included in 'Login App'

Modifying a *Model*

1) Test case 1 “Successful Login”, as its name suggests, covers a successful login, while

2) Test case 2 “Invalid Login” covers an unsuccessful login. In test case “Invalid Login”, Conformiq generated the following data values:

- User name: “admin”
- Password: “”

3) The reason Conformiq generated this set of values (and not others) is because we did not specify in the model any info or data constraints on what scenarios we would like to cover for an unsuccessful login. This test case only covers a valid username and an invalid password.

4) Let’s say, for example, that we are also interested in a case where the username is invalid and a password is valid.

5) Let’s make some changes to the Activity Diagram as detailed in next slide.

6) Switch to the *Conformiq Modeling* Perspective. First, remove the *Control Flow* (connector) between the "*Is User Correct?*" *Decision* and the "*Report error*" *Activity*. Select on the *Control Flow*, and select *Delete from Model*. The *Control Flow* should be deleted.

7) Add a new *Decision* object, named "*Bad Data*". Refer to previous steps in you need to refresh your memory.

8) Next, add a *Control Flow* between the "*Is User Correct?*" and the "*Bad Data*" *Decision* objects.

9) Select the "*No*" *Control Flow* type from the dialog.

10) Next, add a new *Control Flow* between the "*Bad Data*" *Decision* and the "*Report error*" *Activity*, this time choosing the *Yes Control Flow* type.

11) Here we will reuse the user data ("*Entered user data*") *Data* object, so add a *Data Flow* (connector) from the "*Entered user data*" Data object to the "*Bad Data*" *Decision*.

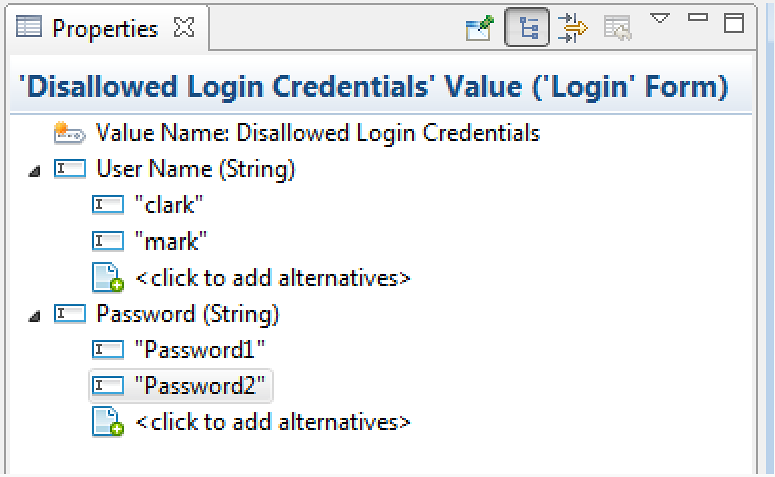
12) Next, let's define the expected values to be compared with. This time, we'll use a value with alternative values.

13) Create a new *Value Data* object, and select “*Login Form Value*” as shown below.

14) In the **Value Name** field, type “*Disallowed Login Credentials*”.

15) In the **User Name** field, add first “*clark” and then as an alternative value “mark*”

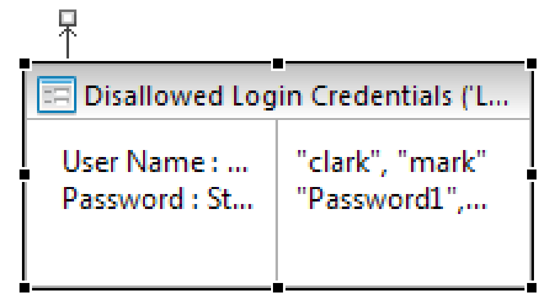
16) In the **Password** field, add values “*Password1” and “Password2*”



'Disallowed Login Credentials' Value ('Login' Form)

- Value Name: Disallowed Login Credentials
- User Name (String)
  - "clark"
  - "mark"
  - <click to add alternatives>
- Password (String)
  - "Password1"
  - "Password2"
  - <click to add alternatives>

17) A “*Disallowed Login Credentials*” Data object appears in the *Activity Diagram as follows*.



Disallowed Login Credentials ('L...

User Name : ... "clark", "mark"
Password : St... "Password1",...

18) Add a *Data Flow* (connector) from the newly created *Data* object to the “*Bad Data*” *Decision* object.

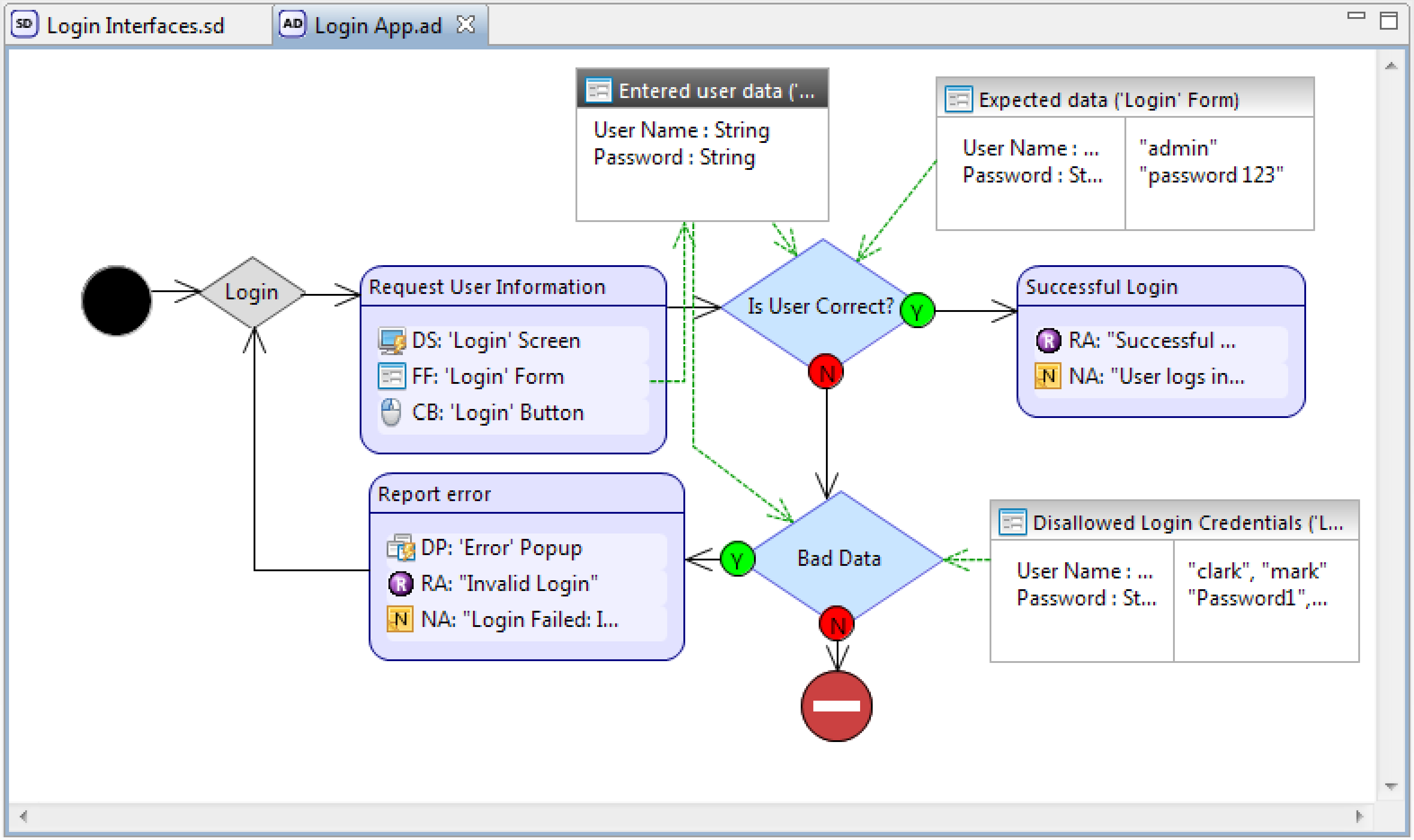
19) Add a *YES control flow* from the “*Bad Data*” *Decision to* “*Report error*” *Activity.*

21) We do not care about the *NO control flow*. Lets draw a *Block Node* and connect it with a *NO control flow*

22) The logic we just added to our *Activity Diagram* is that now the model does not allow a user to login using a username “*clark*” or “mark” and password “*Password1*” or “*Password2*”.

This completes the changes to the model.

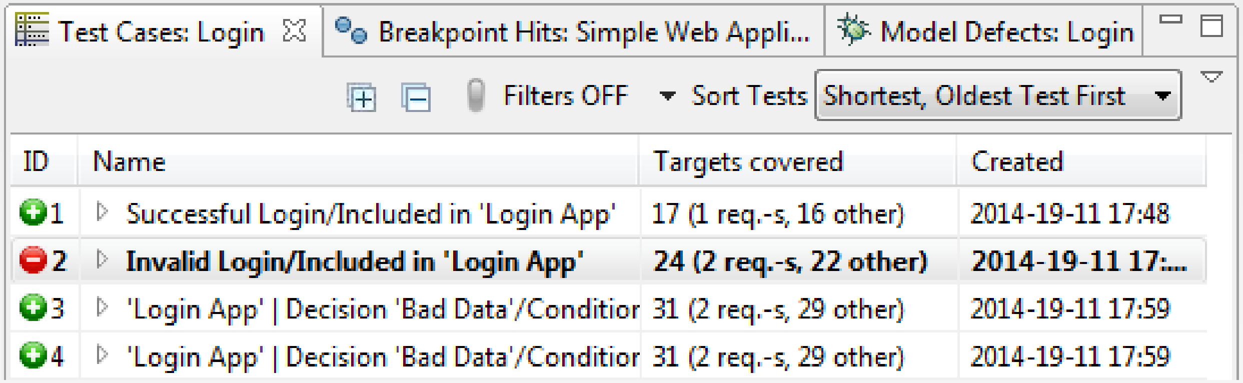
Your modified *Activity Diagram* should look similar to this:



Re-Generating *Test Cases* from modified *Activity Diagram*

1) In the *Conformiq Test Design* perspective , load and generate test cases (refer to section "Generating Test Cases from an *Activity Diagram*" if you need assistance).

2) You should get the following display in the *Test Case View*



Test Cases: Login | Breakpoint Hits: Simple Web Appli... | Model Defects: Login

Filters OFF ▾ Sort Tests: Shortest, Oldest Test First ▾

ID	Name	Targets covered	Created
⊕ 1	▷ Successful Login/Included in 'Login App'	17 (1 req.-s, 16 other)	2014-19-11 17:48
⊖ 2	▷ **Invalid Login/Included in 'Login App'**	**24 (2 req.-s, 22 other)**	**2014-19-11 17:...**
⊕ 3	▷ 'Login App' | Decision 'Bad Data'/Conditior	31 (2 req.-s, 29 other)	2014-19-11 17:59
⊕ 4	▷ 'Login App' | Decision 'Bad Data'/Conditior	31 (2 req.-s, 29 other)	2014-19-11 17:59

3) Test case 2 is now invalidated because a case where the *password* is empty is no longer implemented.

4) Test cases 3 and 4 were added, with "*clark/Password1*" and "*mark/Password2*" sets of user data, reflecting the model changes.

- ❑ ***You have completed the exercise successfully!***
- ❑ ***Try for more such exercises!!***

Ideabytes™
Innovation is Business

- Koti & Narasimha

THANK YOU

Corporate Office, Canada
Ideabytes Inc.
Ottawa, Ontario
Phone : +1 613 692 9908

Development Centre, India
Ideabytes Software India Pvt. Ltd
Jayabheri Enclave, Hyderabad
Phone : +91 40 6453 5959

South East Asia
Kuala Lumpur, Malaysia
Phone: +60 16 220 1365

USA
3389, Napoli pl,
San Jose, CA - 951 35
Phone: +1 408 600 1439

www.ideabytes.com
contact@ideabytes.com