smartesting®

Optimize your Test Center

Model-Based Testing for Information Systems

From Business Requirements to Test Repositories

Author: Bruno LEGEARD
Contact: legeard@smartesting.com

October 2011

Agenda

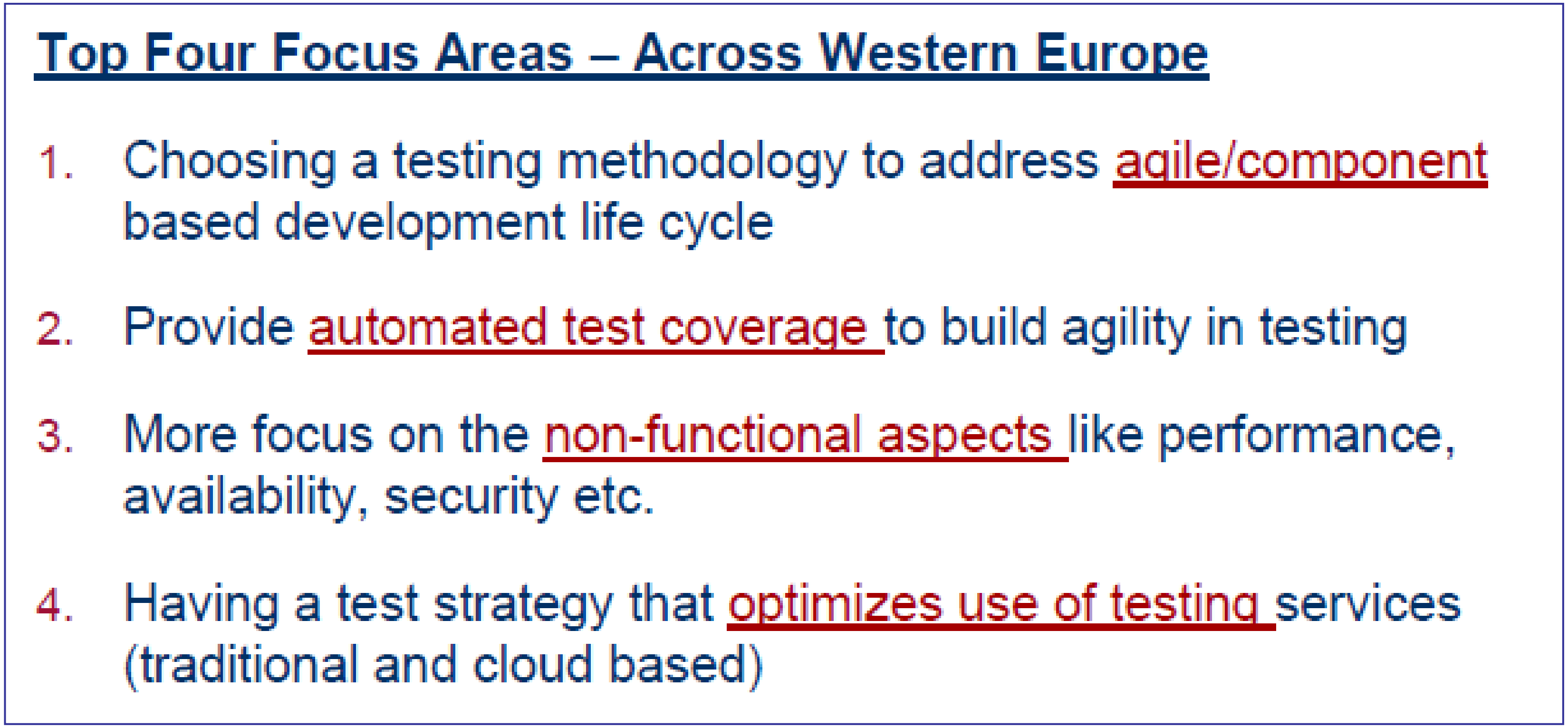
- Part 1: Principles of Model-based Testing for IT Systems…...... 3
  - Testing challenges in the context of Large-scale Enterprise Information Systems
  - Types of testing addressed by model-based testing
  - Composing Business Process models, Behavioral models and Test data models for automated test generation
- Part 2: Step-by-Step Example with Smartesting CertifyIt………14
  - Use BPMN to describe Business Processes for Test Generation
  - Link Business Processes, Business Rules and Logical Test data
  - Define Requirement Traceability information
  - Generate tests with Smartesting CertifyIt
  - Export your tests in the test repository for manual test execution
  - Generate scripts for automated test execution
- Part 3: How to start with MDT, adoption principles...……………. 44
  - Test team organization, roles and responsibilities
  - Agile and model-based testing, a perfect fit
  - Model-based testing and Risk-based testing

Large-scale Enterprise Information Systems

- **System of systems & Complex composite systems**
  - Multiple applications
    - Mix of Bespoke and Packaged applications
    - Mix of data-oriented and process-oriented applications
  - Multiple targeted platforms (PC, Smartphone, Pad)
- **Risk mitigation**
  - **Quality Assurance (Testing) ensures a key role for risk mitigation**
  - **Importance of compliance and regulation rules (SOX…)**
- **Software quality has been becoming a Must have**
  - **Users don't want to use buggy systems anymore**

Testing focus areas of IT organizations

QA/Test function maturity: shift from tactical ad hoc process to a more strategic & centralized approach



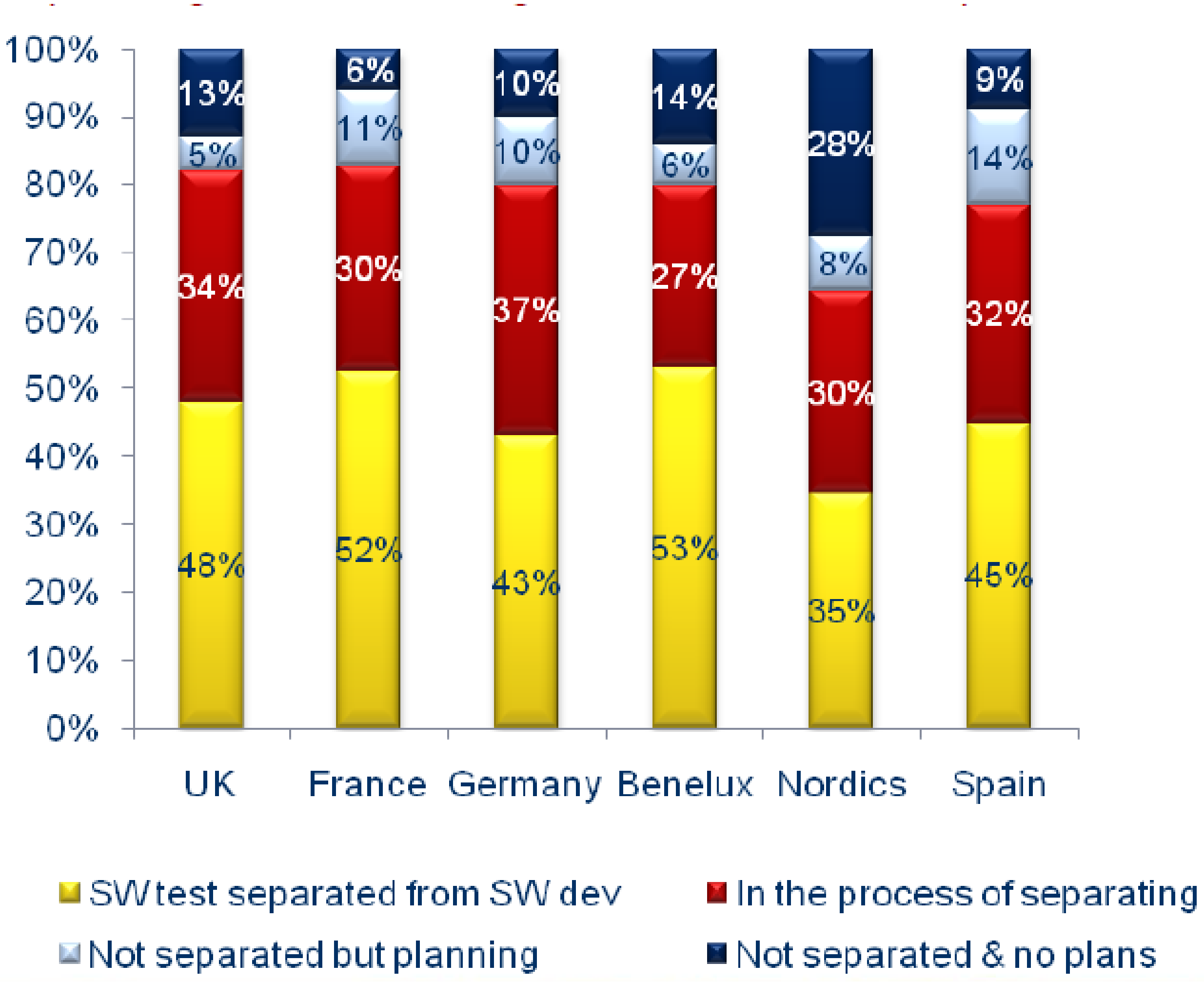
Top Four Focus Areas – Across Western Europe

1. Choosing a testing methodology to address agile/component based development life cycle
2. Provide automated test coverage to build agility in testing
3. More focus on the non-functional aspects like performance, availability, security etc.
4. Having a test strategy that optimizes use of testing services (traditional and cloud based)

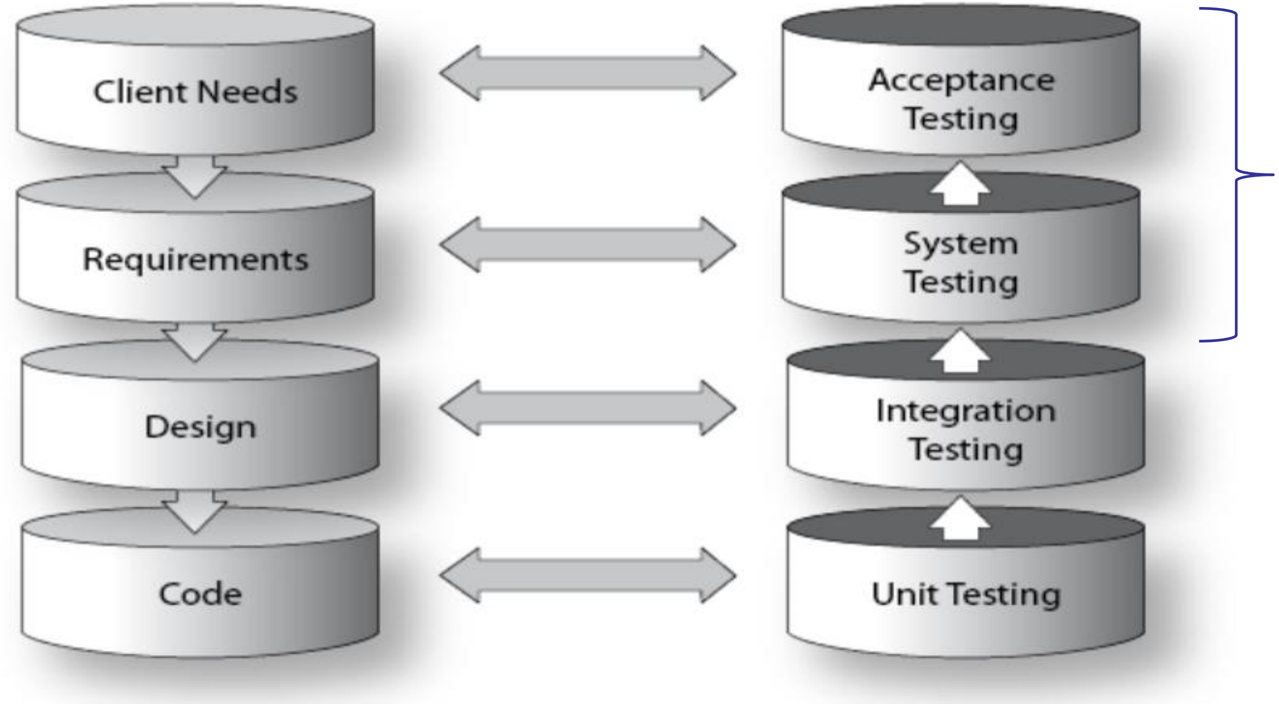
Source IDC - European Services, Enterprise Application Testing Survey, March 2011

Separate software testing from software development

Source IDC - European Services, Enterprise Application Testing Survey, March 2011

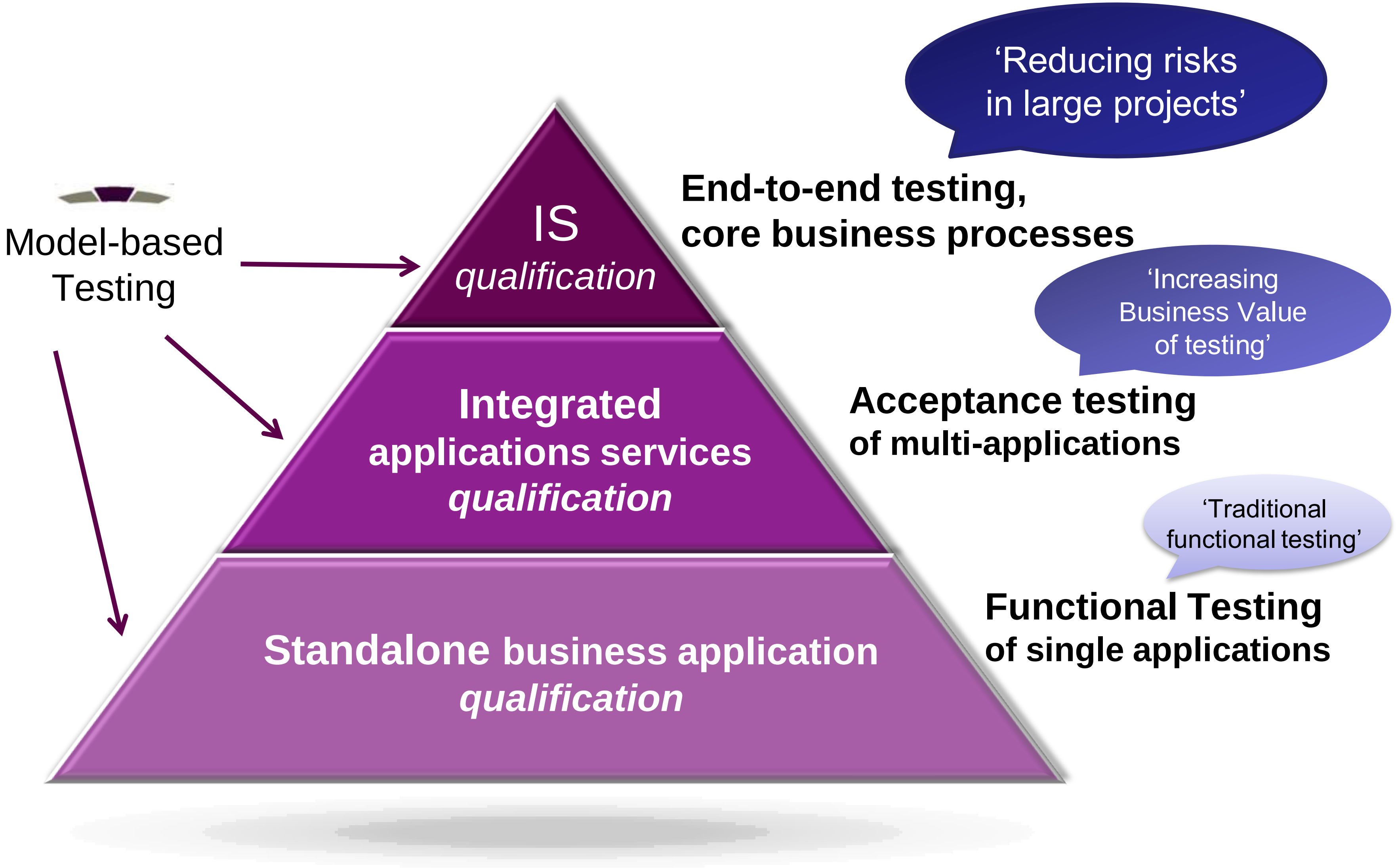


Testing levels



Model-Based Testing for IT Systems

Where Does It Fit?



‘Reducing risks in large projects’

End-to-end testing, core business processes

Model-based Testing

IS *qualification*

‘Increasing Business Value of testing’

Acceptance testing of multi-applications

Integrated applications services *qualification*

‘Traditional functional testing’

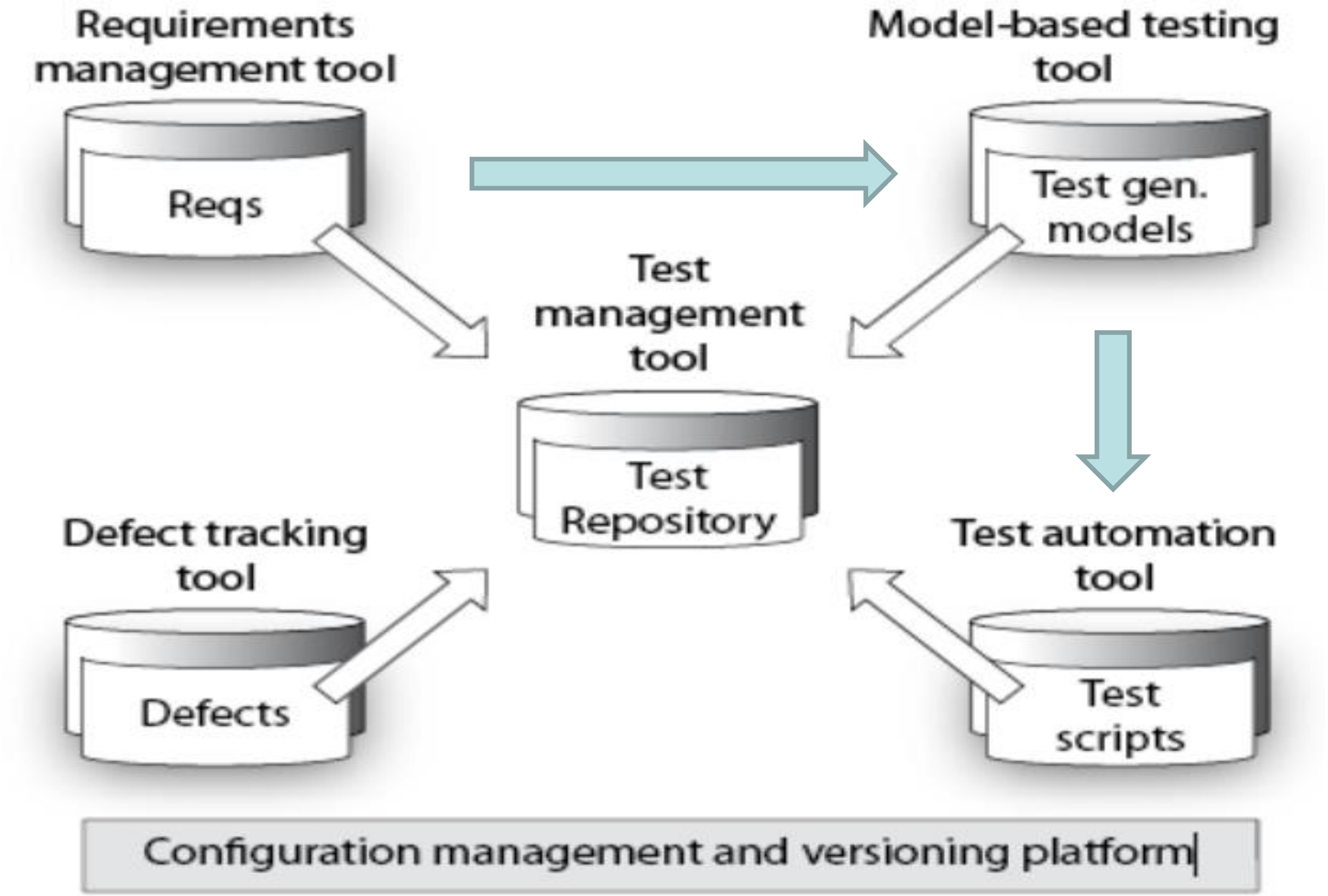
Functional Testing of single applications

Standalone business application *qualification*

Testing methodology

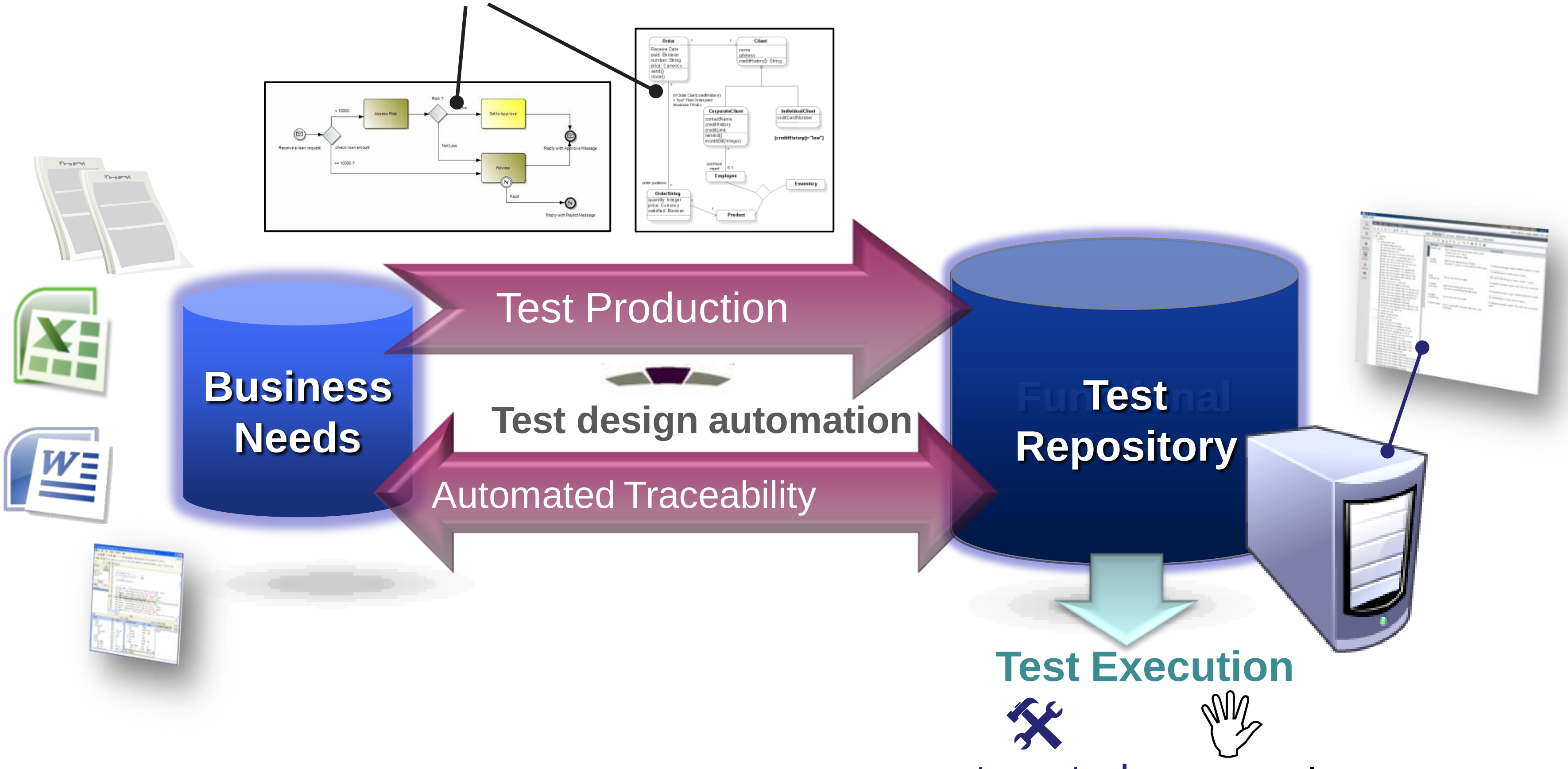
- ➲ RRBT – Risk & Requirements Based Testing
  - – Driving test effort on the basis of risk analysis & linking risks and requirements
  - – Several documented methodology inspired by RRBT:
    - Sogeti Tmap®
    - Logica RRBT
    - ISTQB / CFTL

Testing tools



Model-based testing for IT systems

Model Assets for automated test generation



Business Needs

Test Production

Test design automation

Automated Traceability

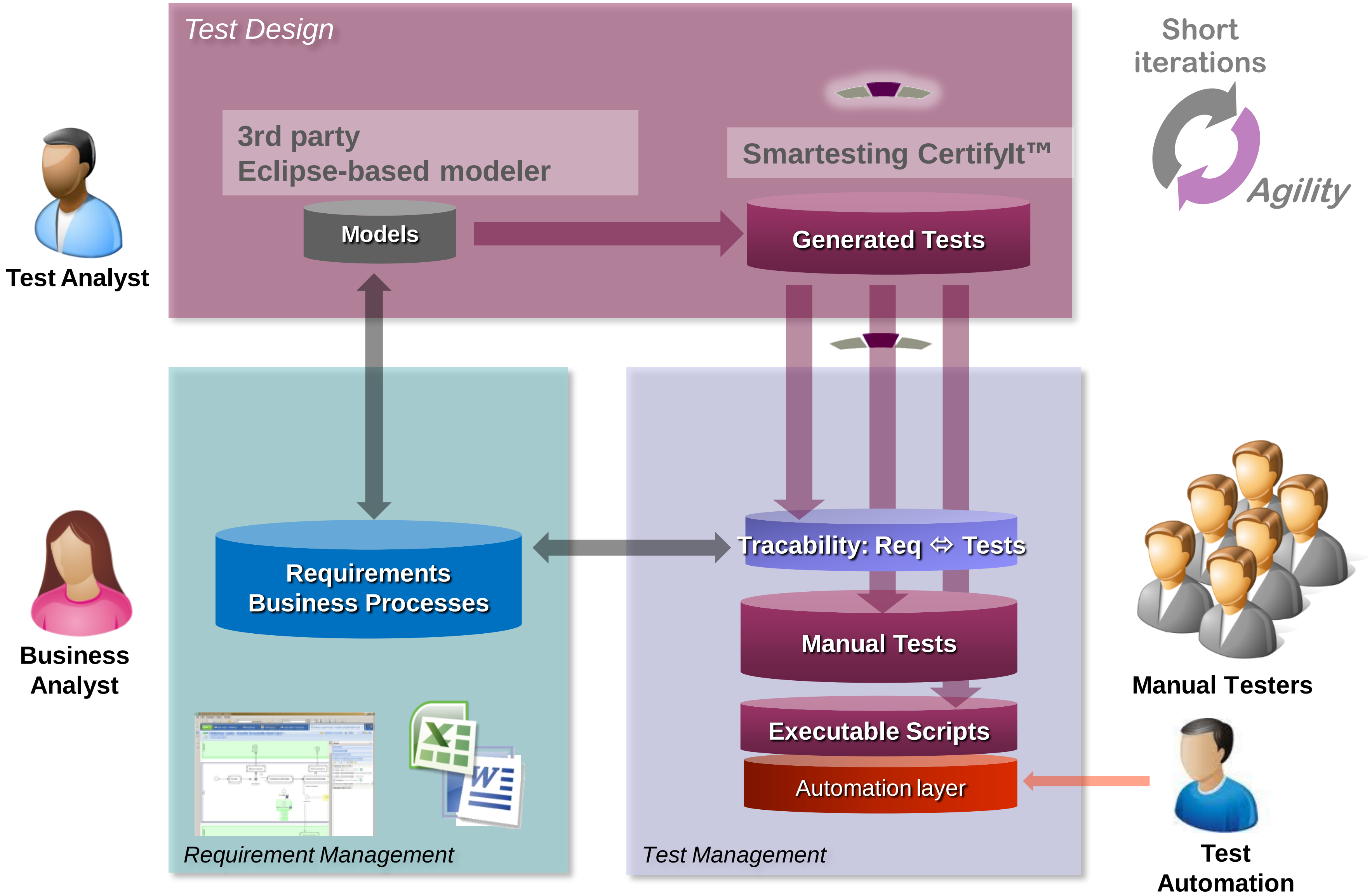
Test Repository

Test Execution

automated

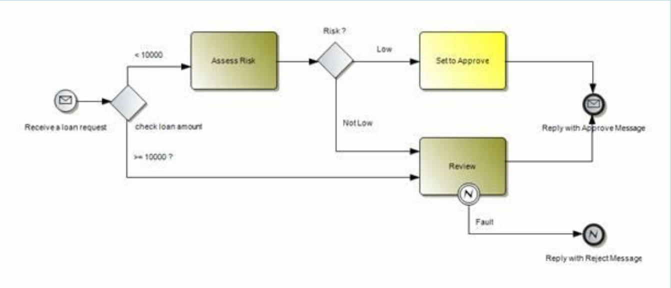
manual

Test Generation Process

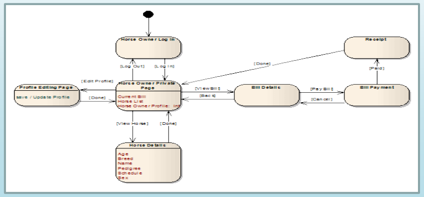


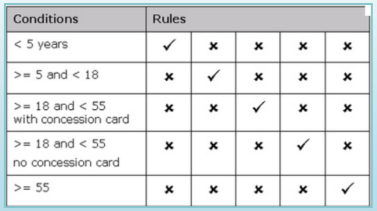
Models for Automated Test Generation

Business Process Model (BPMN)



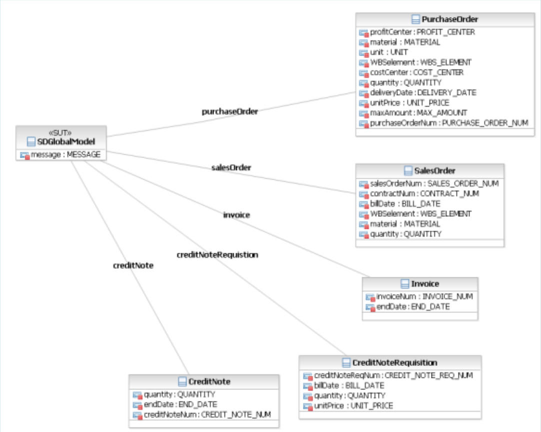
Business Rules and Behavioral Model (UML)





Conditions	Rules				
< 5 years	✓	×	×	×	×
>= 5 and < 18	×	✓	×	×	×
>= 18 and < 55 with concession card	×	×	✓	×	×
>= 18 and < 55 no concession card	×	×	×	✓	×
>= 55	×	×	×	×	✓

Business Entities and Logical Test Data (UML)



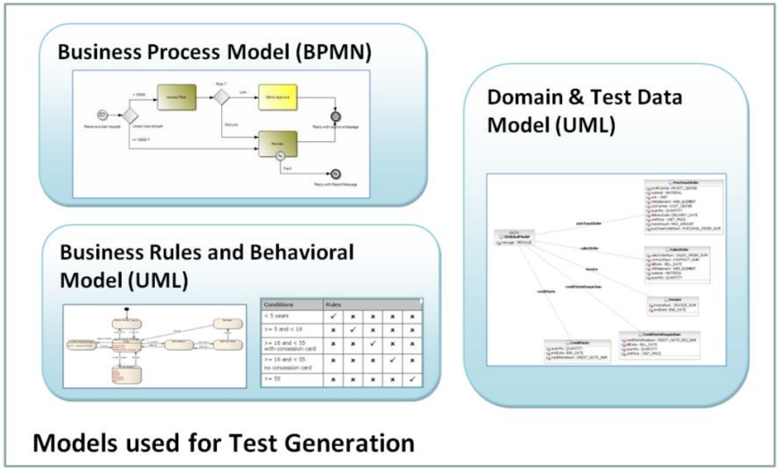
Models used with Smartesting

Agenda

- Part 1: Principles of Model-based Testing for IT Systems…..... 3
  - Testing challenges in the context of Large-scale Enterprise Information Systems
  - Types of testing addressed by model-based testing
  - Composing Business Process models, Behavioral models and Test data models for automated test generation
- Part 2: Step-by-Step Example with Smartesting CertifyIt………14
  - Use BPMN to describe Business Processes for Test Generation
  - Link Business Processes, Business Rules and Logical Test data
  - Define Requirement Traceability information
  - Generate tests with Smartesting CertifyIt
  - Export your tests in the test repository for manual test execution
  - Generate scripts for automated test execution
- Part 3: How to start with MDT, adoption principles...…………… 44
  - Test team organization, roles and responsibilities
  - Agile and model-based testing, a perfect fit
  - Model-based testing and Risk-based testing

Test Repository is fully driven from models + the testing strategy

What do you want to test?


Business Process Model (BPMN)
Domain & Test Data Model (UML)
Business Rules and Behavioral Model (UML)
Models used for Test Generation

- Expected behavior
- Observation point
- Processes and flows
- Business rules to be tested
- Documentation of actions

Test Analyst

How do you want to test it?

Testing Strategy

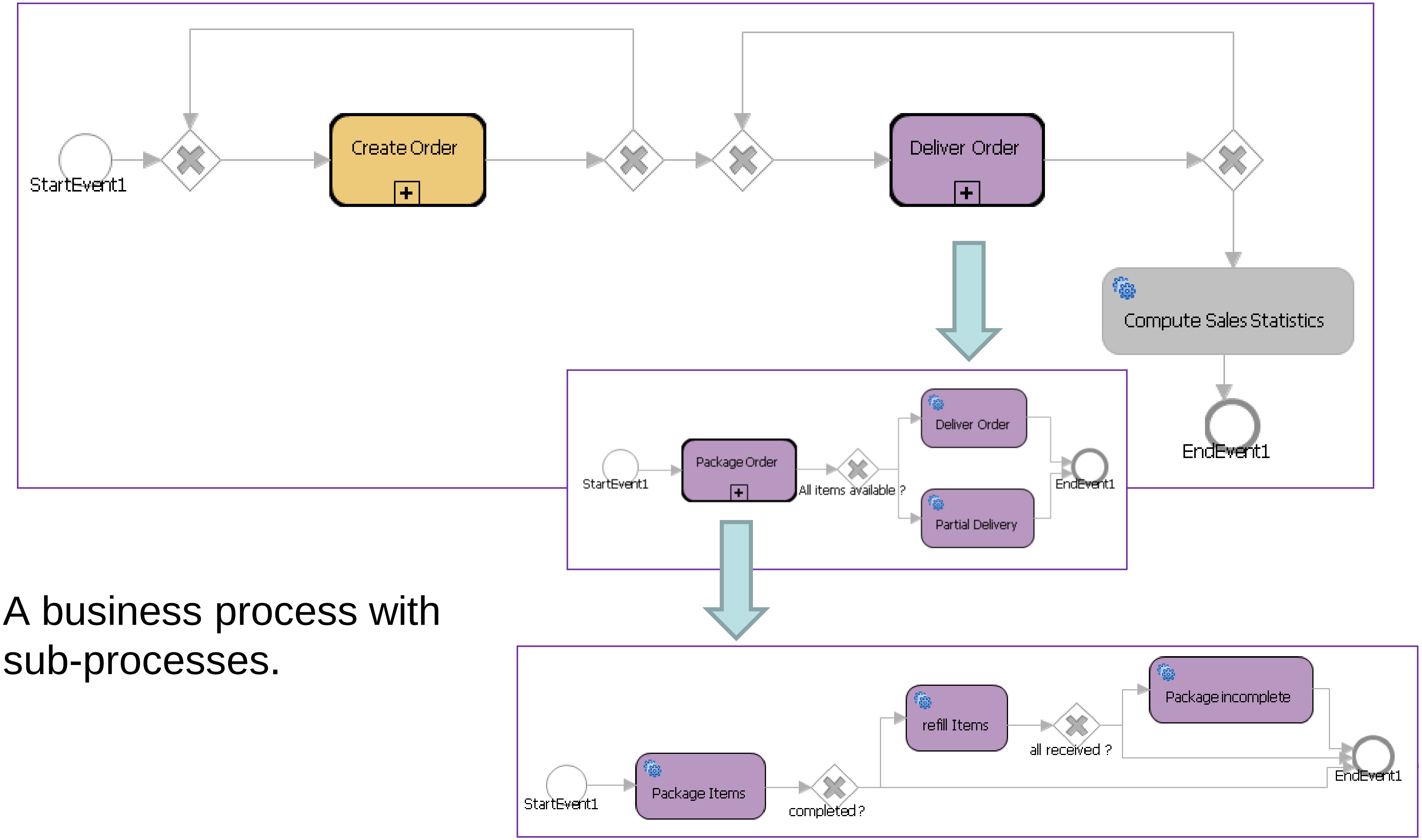
- Model coverage
- Configuration
- Initial state

Automated Test Generation

Models of Business Processes structure the Smartesting MBT solution for IT

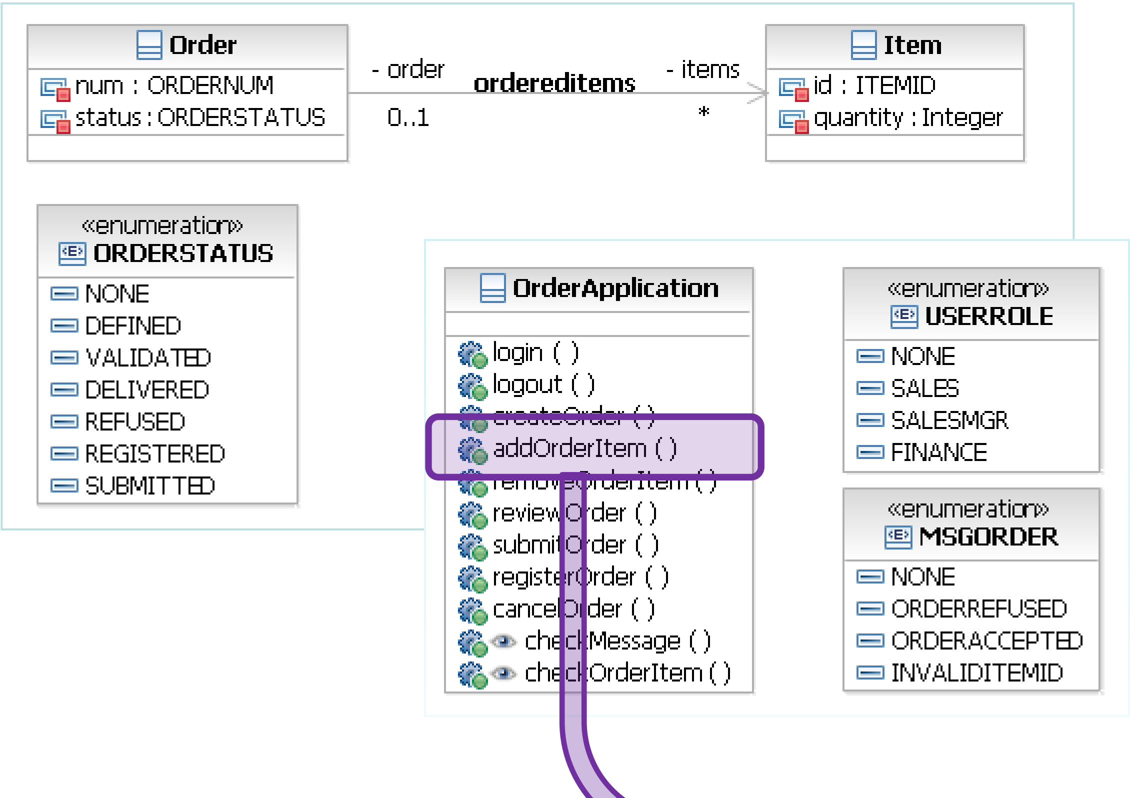
- Business Process models formalize the business workflow to be tested
  - Facilitating the communication between QA team and Business Analysts
  - Simplifying modeling activities for Test Analysts (Business Processes + Business Rules + Logical Test Data)
- Business Process models may be reuse from upfront activities
  - From Business Process Analysis
  - From Requirements Elicitation
- Business Process modeling is based on standard BPMN notation from OMG

Modeling Business Processes with BPMN



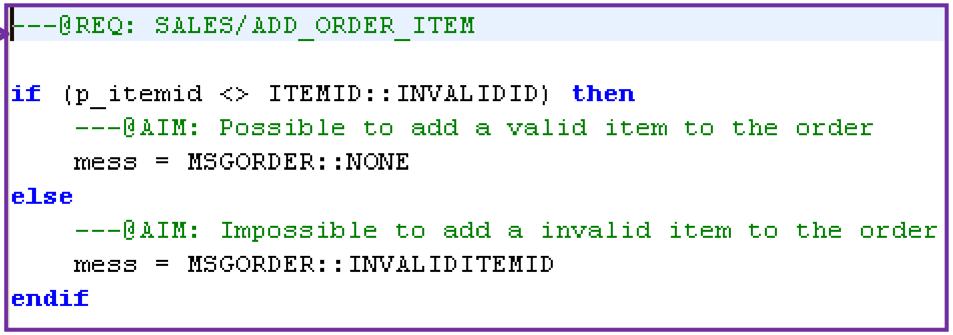
A business process with sub-processes.

Modeling Business Entities and Expected Behavior



Business entities and boundaries are described:

- at corporate level (shared by all apps)
- at application level



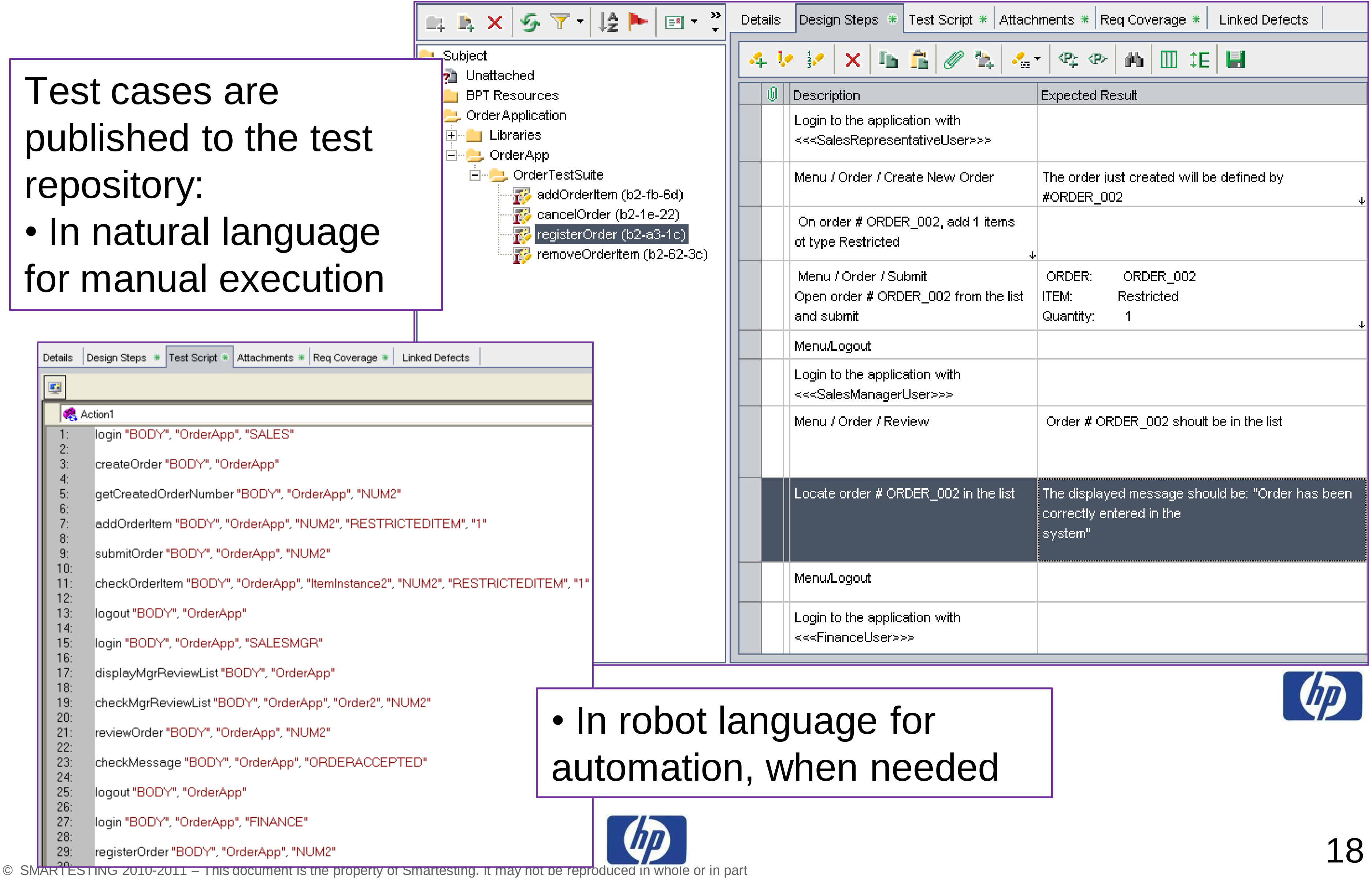
```
---@REQ: SALES/ADD_ORDER_ITEM

if (p_itemid <> ITEMID::INVALIDID) then
    ---@AIM: Possible to add a valid item to the order
    mess = MSGORDER::NONE
else
    ---@AIM: Impossible to add a invalid item to the order
    mess = MSGORDER::INVALIDITEMID
endif
```

A precise description of the requirements and business rules define the **expected behavior**

Publication to the Test Repository for Test Execution

Test cases are published to the test repository:

- In natural language for manual execution



Description	Expected Result
Login to the application with <<<SalesRepresentativeUser>>>	
Menu / Order / Create New Order	The order just created will be defined by #ORDER_002
On order # ORDER_002, add 1 items of type Restricted	
Menu / Order / Submit Open order # ORDER_002 from the list and submit	ORDER: ORDER_002 ITEM: Restricted Quantity: 1
Menu/Logout	
Login to the application with <<<SalesManagerUser>>>	
Menu / Order / Review	Order # ORDER_002 shoult be in the list
Locate order # ORDER_002 in the list	The displayed message should be: "Order has been correctly entered in the system"
Menu/Logout	
Login to the application with <<<FinanceUser>>>	

Action1

```
login "BODY", "OrderApp", "SALES"

createOrder "BODY", "OrderApp"

getCreatedOrderNumber "BODY", "OrderApp", "NUM2"

addOrderItem "BODY", "OrderApp", "NUM2", "RESTRICTEDITEM", "1"

submitOrder "BODY", "OrderApp", "NUM2"

checkOrderItem "BODY", "OrderApp", "ItemInstance2", "NUM2", "RESTRICTEDITEM", "1"

logout "BODY", "OrderApp"

login "BODY", "OrderApp", "SALESMGR"

displayMgrReviewList "BODY", "OrderApp"

checkMgrReviewList "BODY", "OrderApp", "Order2", "NUM2"

reviewOrder "BODY", "OrderApp", "NUM2"

checkMessage "BODY", "OrderApp", "ORDERACCEPTED"

logout "BODY", "OrderApp"

login "BODY", "OrderApp", "FINANCE"

registerOrder "BODY", "OrderApp", "NUM2"
```

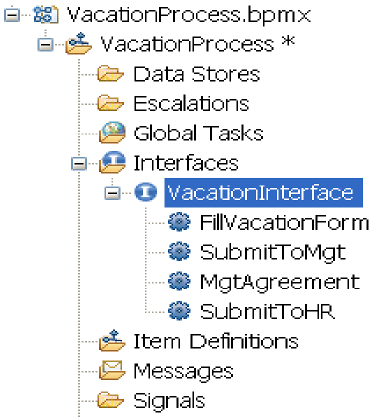
- In robot language for automation, when needed

Composing BPMN models with behavioral models

- Business Processes described with BPMN
- Each "Service Task" in BPMN is linked to a Smartesting UML operation
- The Smartesting simplified UML stereotype is used to:
  - Simplify modeling for testing
  - Make the Business Scenarios issued from Business Processes executable (manual or auto)
  - Capture the business rules and expected behaviors
  - Provide requirements coverage

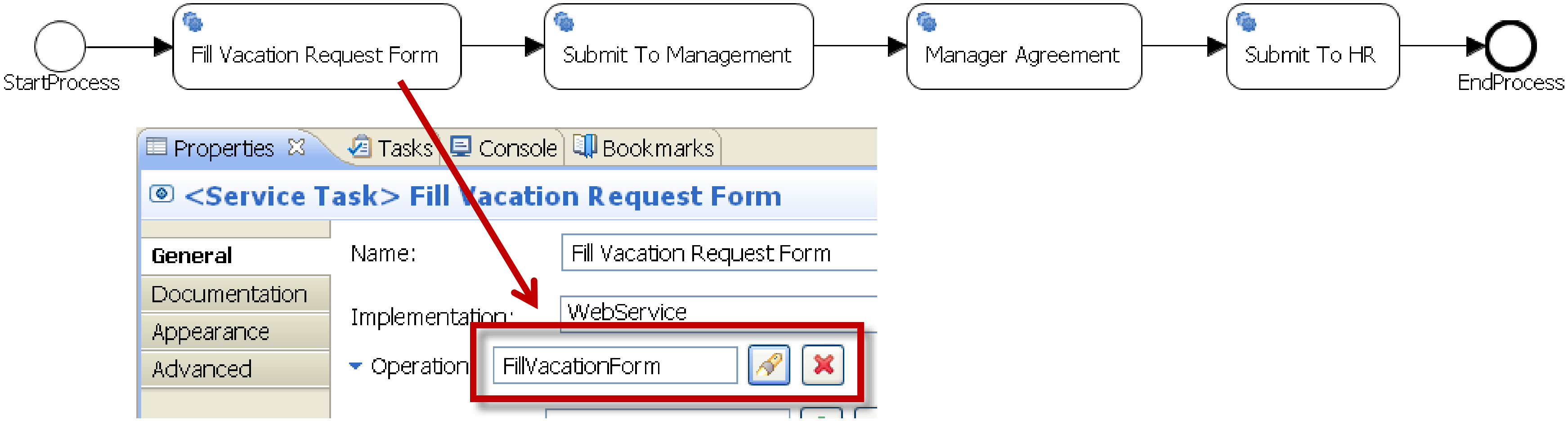
How to link BPMN with Smartesting UML *(BPMN Side)*

1. Define Interface Operations for each Service Task



VacationProcess.bpmx
- VacationProcess *
  - Data Stores
  - Escalations
  - Global Tasks
  - Interfaces
    - VacationInterface
      - FillVacationForm
      - SubmitToMgt
      - MgtAgreement
      - SubmitToHR
  - Item Definitions
  - Messages
  - Signals

2. Associate each Service Task with Interface Operations



<Service Task> Fill Vacation Request Form

Name: Fill Vacation Request Form

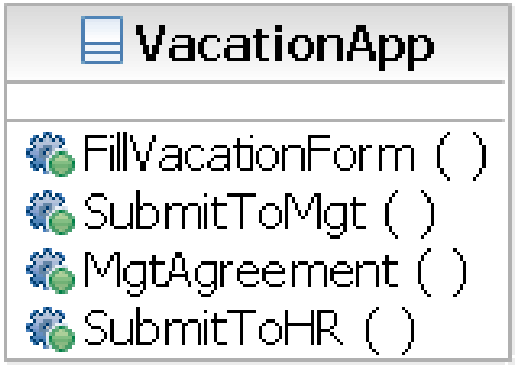
Implementation: WebService

Operation: FillVacationForm

How to link BPMN with Smartesting UML

(UML Side)

- Declare UML operations for each Interface Operation



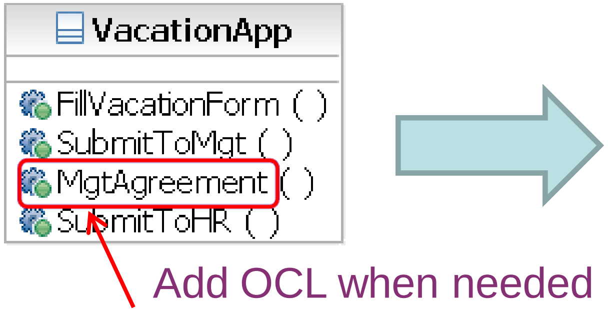
VacationApp

FillVacationForm ( )
SubmitToMgt ( )
MgtAgreement ( )
SubmitToHR ( )

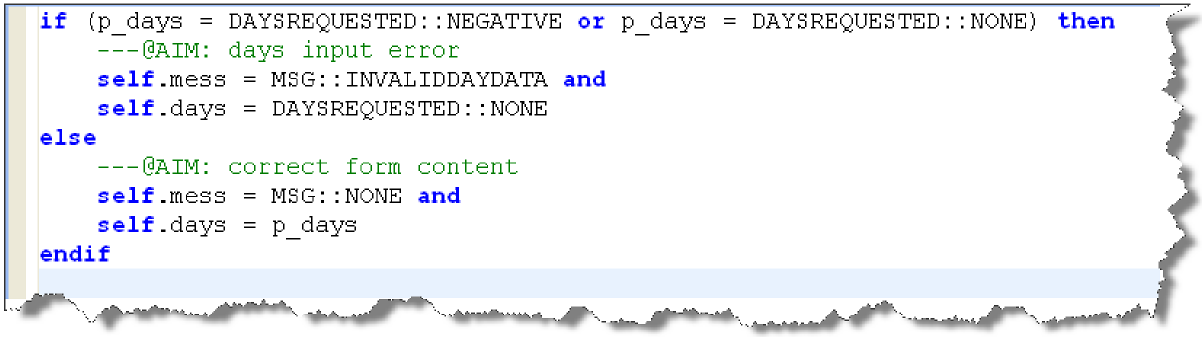
How to define detailed business rules

With OCL:

- OCL stands for Object Constraint Language
- Its goal is to express constraints on UML elements to overcome the limitations inherent to any graphical representation
- It manipulates objects and collections of objects
- Operations can be called in OCL
- OCL scripts are always in the context of a class
- OCL is used to express both the conditions under which an action is possible and the effects of this action

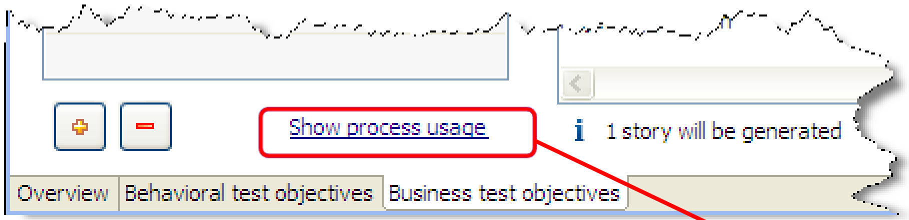


Add OCL when needed for each operation

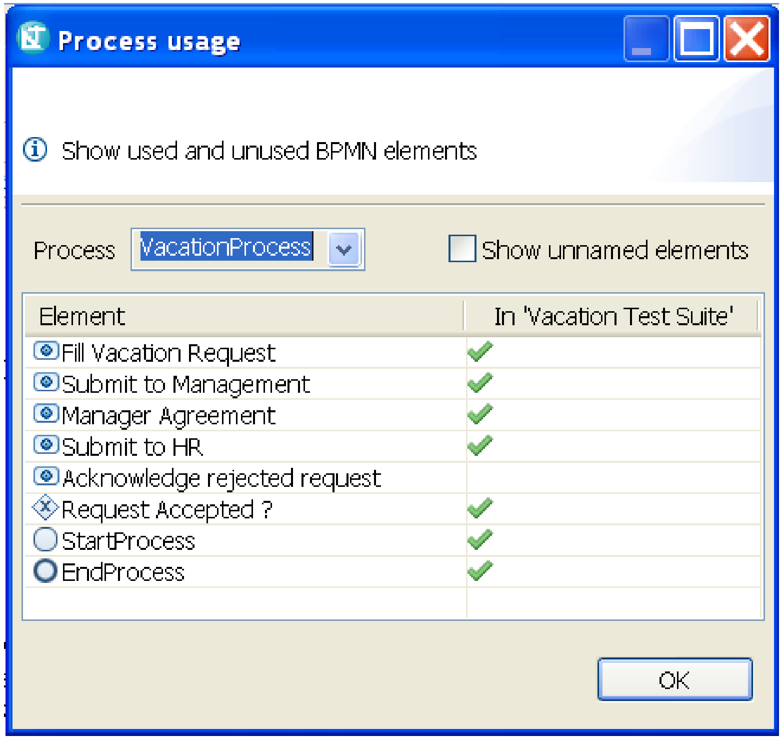


```
if (p_days = DAYSREQUESTED::NEGATIVE or p_days = DAYSREQUESTED::NONE) then
    ---@AIM: days input error
    self.mess = MSG::INVALIDDAYDATA and
    self.days = DAYSREQUESTED::NONE
else
    ---@AIM: correct form content
    self.mess = MSG::NONE and
    self.days = p_days
endif
```

How to determine the coverage of the Business Scenarios w.r.t. the BPMN model



ℹ 1 story will be generated



ⓘ Show used and unused BPMN elements

Process: VacationProcess

□ Show unnamed elements

Element	In 'Vacation Test Suite'
Fill Vacation Request	✓
Submit to Management	✓
Manager Agreement	✓
Submit to HR	✓
Acknowledge rejected request	
Request Accepted ?	✓
StartProcess	✓
EndProcess	✓

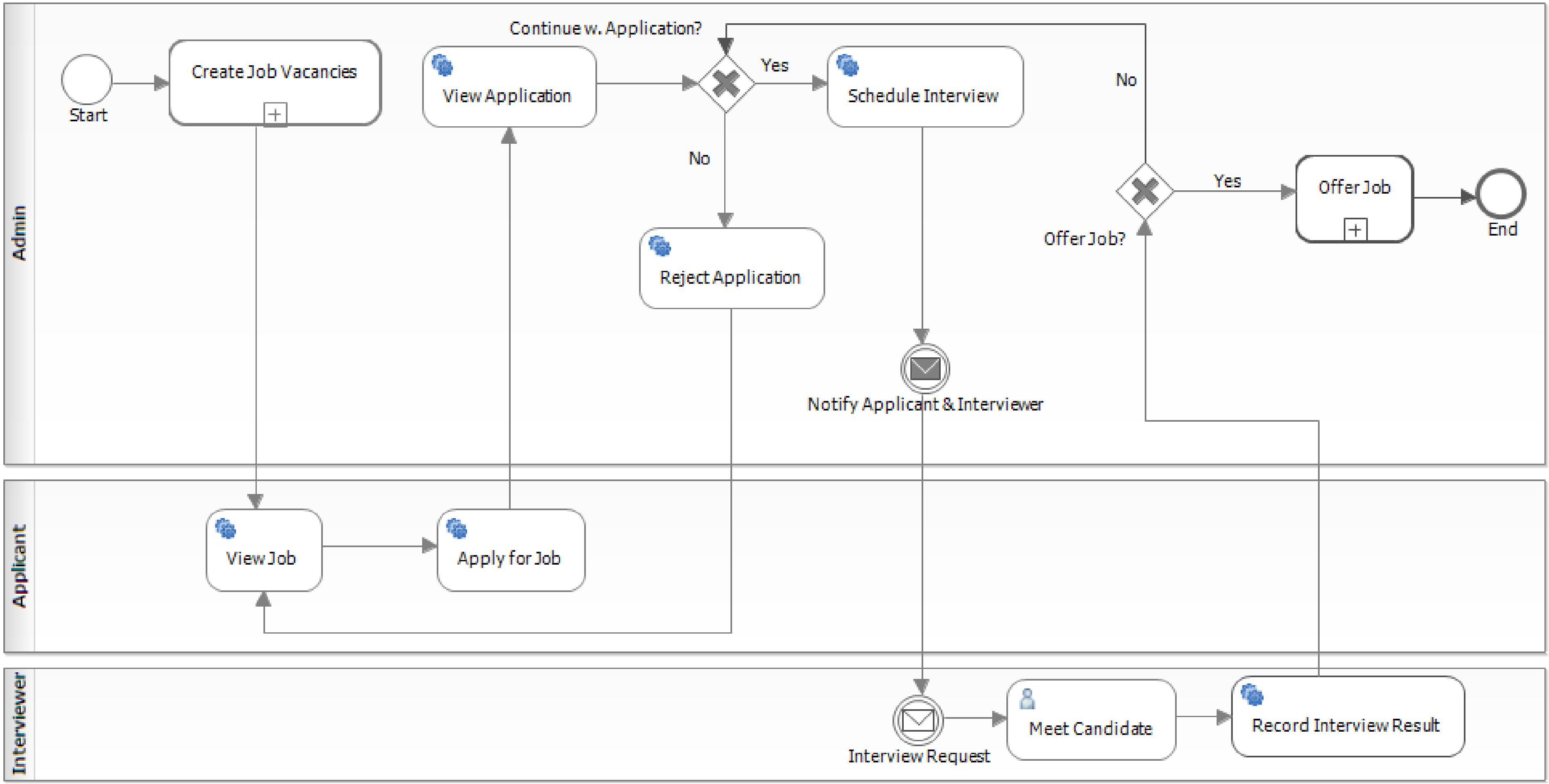
Process Usage provides an accurate view of what paths in the business process have been covered or not

A step-by-step example

- Objectives
  - How to develop test generation models from BPM and Smartesting UML diagrams
  - How to use test generation models to automatically create test cases
  - How to use the generated tests in the test management environment

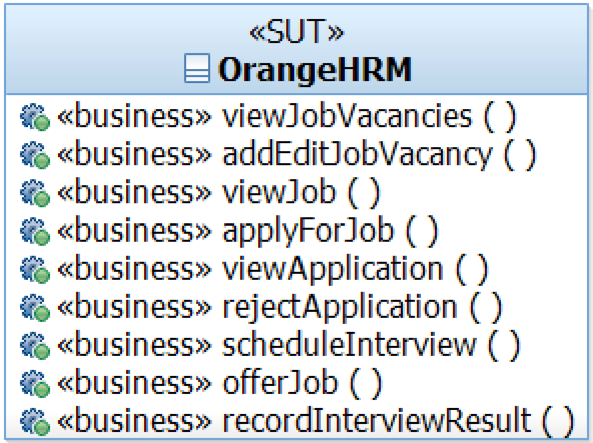
Analyze Requirements:

BPM Example: OrangeHRM Recruitment Process



Analyze/Refine Your Business Flows:
1 - Building a List of Keywords (1/2)

- Keywords as operations of the SUT
  - Operation signature
    - ➔ Keyword specification
      - ➔ Contract between business logic and technical implementation
  - Operation parameters can be used to model
    - Multiple choices/options (e.g. selecting a menu item)
    - User forms (e.g. login)



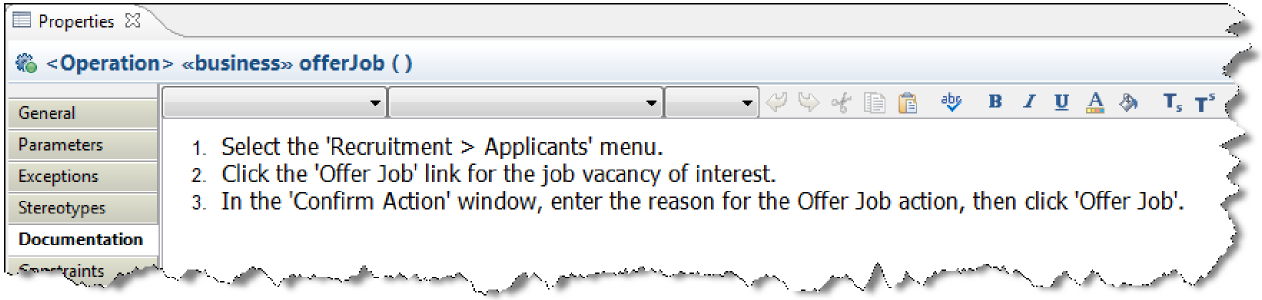
«SUT»
OrangeHRM

«business» viewJobVacancies ( )
«business» addEditJobVacancy ( )
«business» viewJob ( )
«business» applyForJob ( )
«business» viewApplication ( )
«business» rejectApplication ( )
«business» scheduleInterview ( )
«business» offerJob ( )
«business» recordInterviewResult ( )

(continues next slide)

Analyze/Refine Your Business Flows:
1 - Building a List of Keywords (2/2)

- Documenting the keywords
  - Natural language documentation of what the user should do to perform the action
  - Example: documentation of <<business>> offerJob()

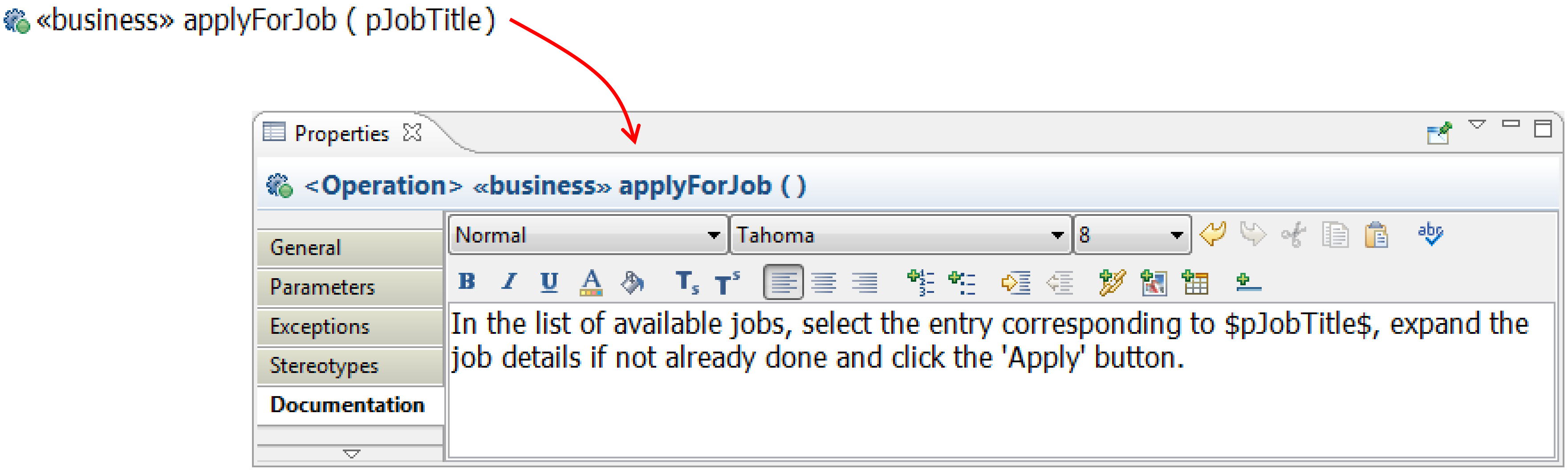


1. Select the 'Recruitment > Applicants' menu.
2. Click the 'Offer Job' link for the job vacancy of interest.
3. In the 'Confirm Action' window, enter the reason for the Offer Job action, then click 'Offer Job'.

- Keywords as the basis for automation
  - Signature of the keyword-operations = interface between models and technical implementation
  - Documented in the “Adaptation Layer Specification” for the test automation engineer

Develop Behavioral Model:
2 - Input Parameters for Business Actions

- Purpose: add variability to business actions
  - Contribute to the documentation of the business actions
    - Example: (parameter type not shown)



«business» applyForJob (pJobTitle)
Properties
<Operation> «business» applyForJob ( )
General
Parameters
Exceptions
Stereotypes
Documentation
Normal
Tahoma
8

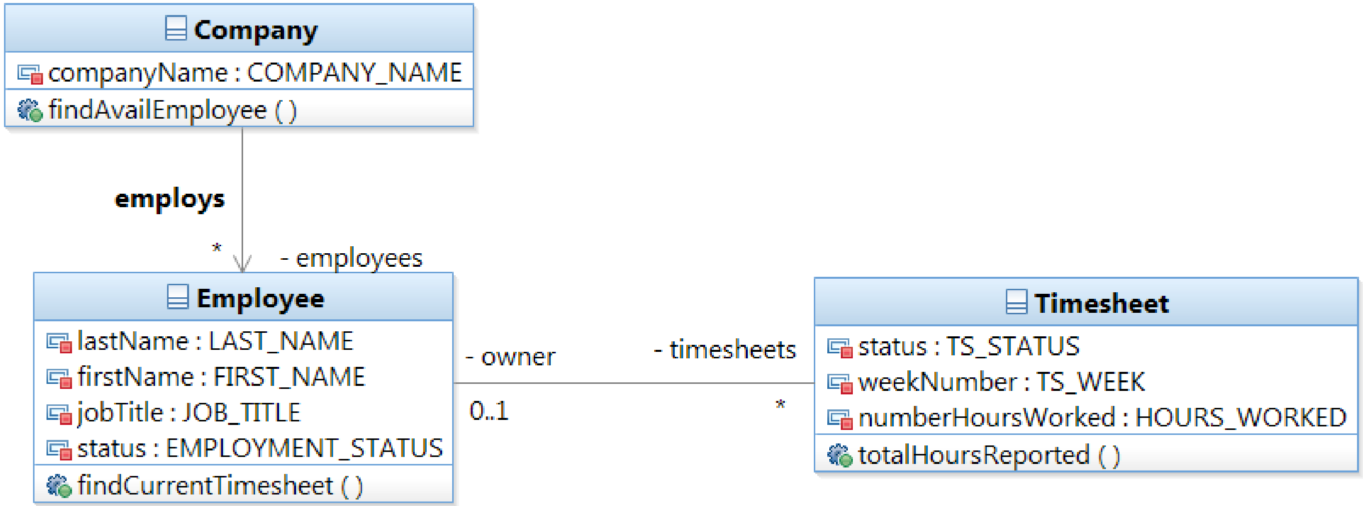
In the list of available jobs, select the entry corresponding to $pJobTitle$, expand the job details if not already done and click the 'Apply' button.

- Supported types:
  - Enumeration classes
  - Primitive types: Integers and Booleans

Develop Behavioral Model:
3 - Business Entities

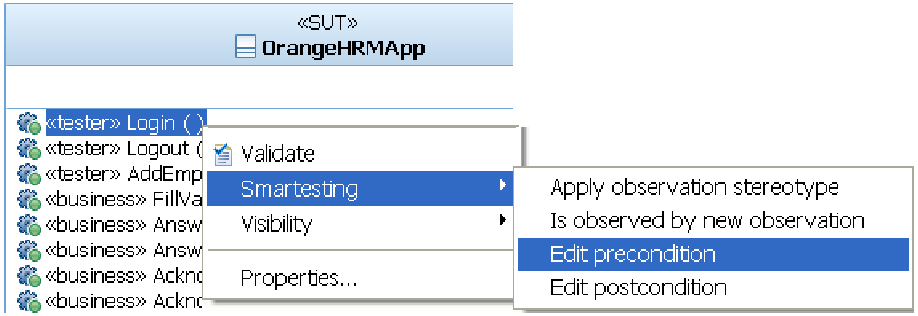
➲ Definition:

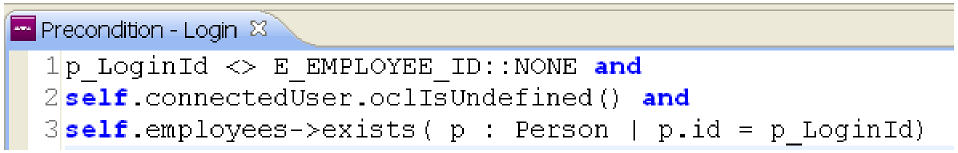
- Business entities correspond to the terms specific to the business domain (e.g. flight, traveler, reservation for an on-line flight reservation site) and are modeled as UML (entity) classes
- They have:
  - Characteristics modeled as UML attributes
  - Relationships with other classes modeled as UML associations
  - Behavior modeled as UML operations



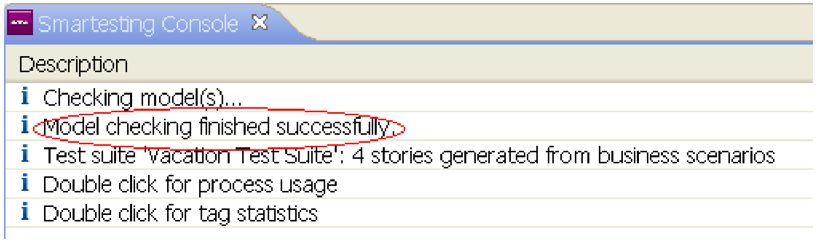
Develop Behavioral Model:
4 - Modeling Behavior

- OCL is used to express both the <u>conditions</u> under which an action is possible and the <u>effects</u> of this action
- OCL is attached to operations (*pre-* and *post-conditions*)





```
p_LoginId <> E_EMPLOYEE_ID::NONE and
self.connectedUser.oclIsUndefined() and
self.employees->exists( p : Person | p.id = p_LoginId)
```



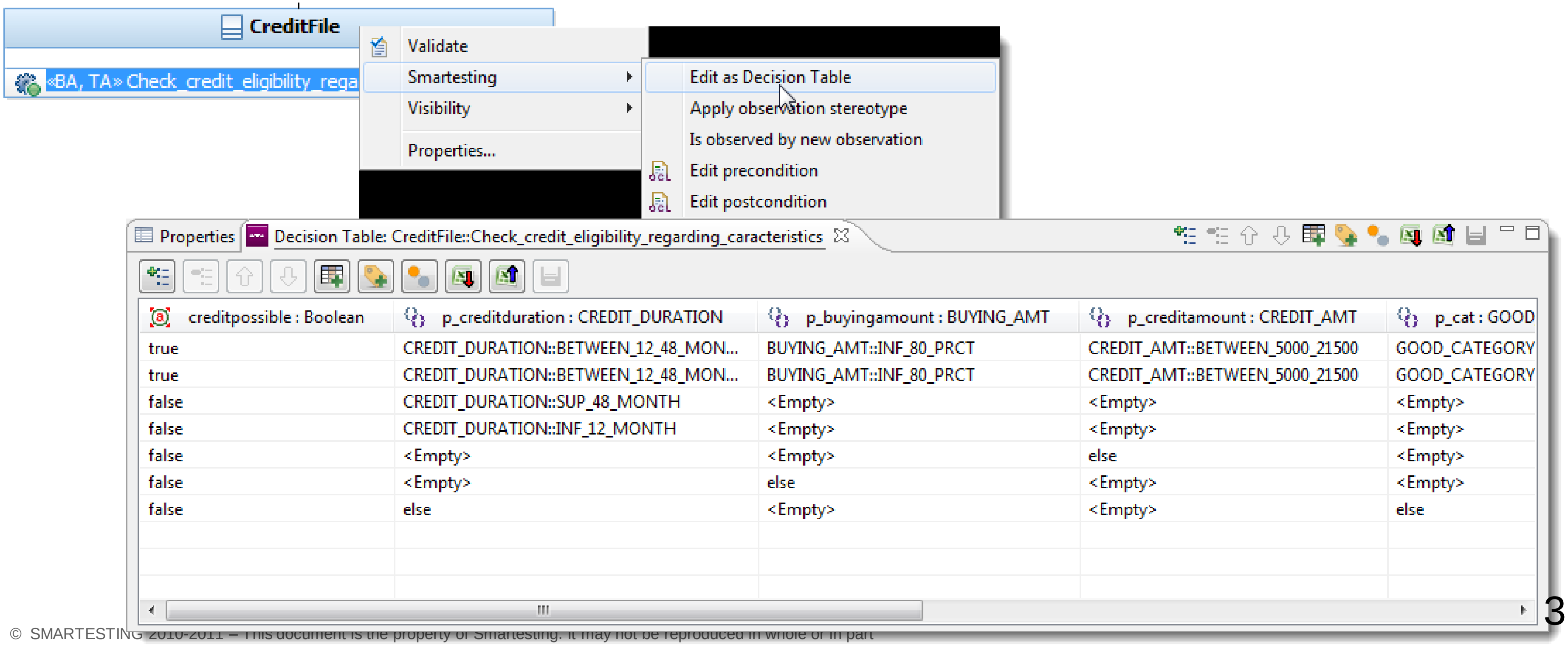
Description

- Checking model(s)...
- Model checking finished successfully
- Test suite 'Vacation Test Suite': 4 stories generated from business scenarios
- Double click for process usage
- Double click for tag statistics

Develop Behavioral Model:

5 - Expressing Conditions in Pre/Post-Conditions

- In post-conditions, conditions represent possible application behaviors (including error cases) that can be tested
- In pre-conditions, conditions are used to tune the model and filter out behaviors that are not possible
- The *Decision Table* can be used to capture easily the business rules


CreditFile
«BA, TA» Check_credit_eligibility_rega
Validate
Smartesting
Visibility
Properties...
Edit as Decision Table
Apply observation stereotype
Is observed by new observation
Edit precondition
Edit postcondition
Properties
Decision Table: CreditFile::Check_credit_eligibility_regarding_caracteristics

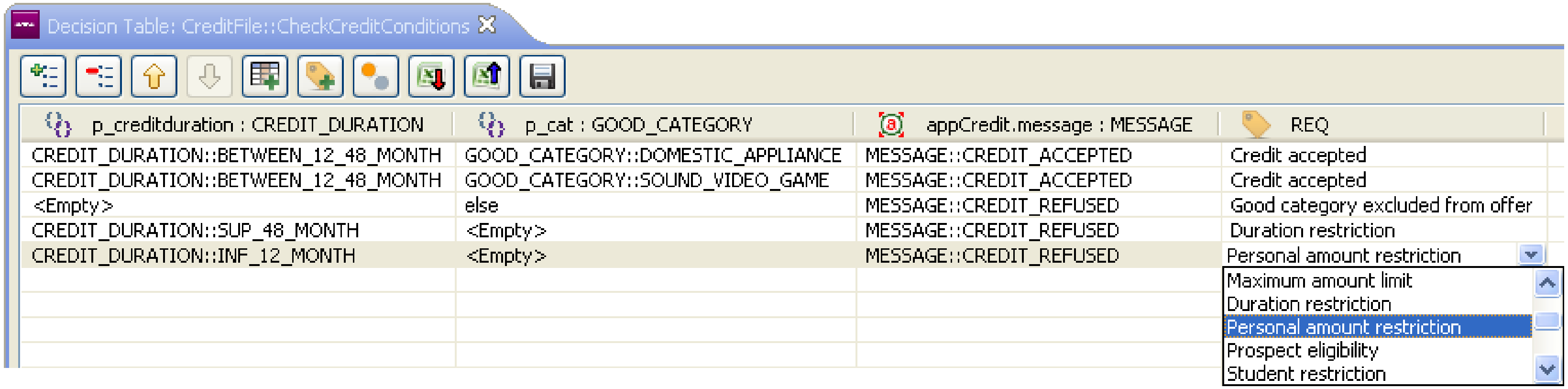
creditpossible : Boolean	p_creditduration : CREDIT_DURATION	p_buyingamount : BUYING_AMT	p_creditamount : CREDIT_AMT	p_cat : GOOD
true	CREDIT_DURATION::BETWEEN_12_48_MON...	BUYING_AMT::INF_80_PRCT	CREDIT_AMT::BETWEEN_5000_21500	GOOD_CATEGORY
true	CREDIT_DURATION::BETWEEN_12_48_MON...	BUYING_AMT::INF_80_PRCT	CREDIT_AMT::BETWEEN_5000_21500	GOOD_CATEGORY
false	CREDIT_DURATION::SUP_48_MONTH	<Empty>	<Empty>	<Empty>
false	CREDIT_DURATION::INF_12_MONTH	<Empty>	<Empty>	<Empty>
false	<Empty>	<Empty>	else	<Empty>
false	<Empty>	else	<Empty>	<Empty>
false	else	<Empty>	<Empty>	else

Develop Behavioral Model:
6 - Requirement Traceability and Test Aims

- Purpose:
  - Provide traceability links between test cases and requirements
  - Keep track of the atomic behaviors covered by each test case
- In practice, tags are added to the model:
  - @REQ tags for requirements and @AIM tags for test aims
  - In the effect of operations only

```
--- @REQ: example TO DO
--- @AIM:
```

- In most cases it's easier to use the Smartesting tabular view:

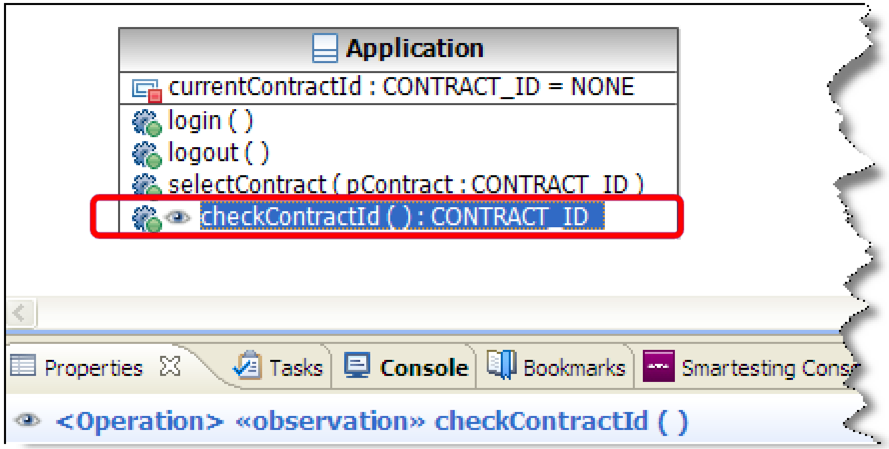


Decision Table: CreditFile::CheckCreditConditions

p_creditduration : CREDIT_DURATION	p_cat : GOOD_CATEGORY	appCredit.message : MESSAGE	REQ
CREDIT_DURATION::BETWEEN_12_48_MONTH	GOOD_CATEGORY::DOMESTIC_APPLIANCE	MESSAGE::CREDIT_ACCEPTED	Credit accepted
CREDIT_DURATION::BETWEEN_12_48_MONTH	GOOD_CATEGORY::SOUND_VIDEO_GAME	MESSAGE::CREDIT_ACCEPTED	Credit accepted
<Empty>	else	MESSAGE::CREDIT_REFUSED	Good category excluded from offer
CREDIT_DURATION::SUP_48_MONTH	<Empty>	MESSAGE::CREDIT_REFUSED	Duration restriction
CREDIT_DURATION::INF_12_MONTH	<Empty>	MESSAGE::CREDIT_REFUSED	Personal amount restriction

Develop Behavioral Model:
7 - Observation points

- Observation points are operations stereotyped <<observation>>



Application

currentContractId : CONTRACT_ID = NONE

login ( )

logout ( )

selectContract (pContract : CONTRACT_ID)

checkContractId ( ) : CONTRACT_ID

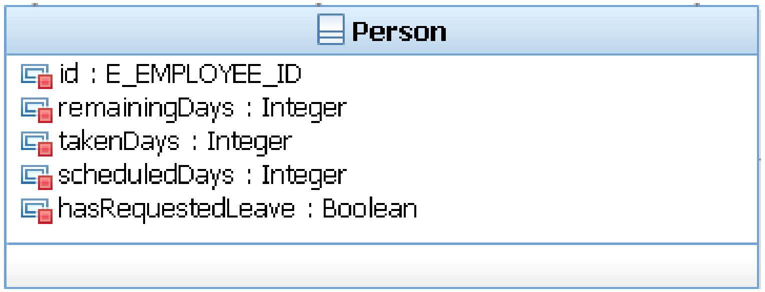
<Operation> «observation» checkContractId ( )

- Two approaches:
  - When the watched data has changed OR
  - When specific operations are called
- Like other operations, they can be documented and they can have OCL conditions (typically to further limit the situations when they are triggered)

Build Test Data:

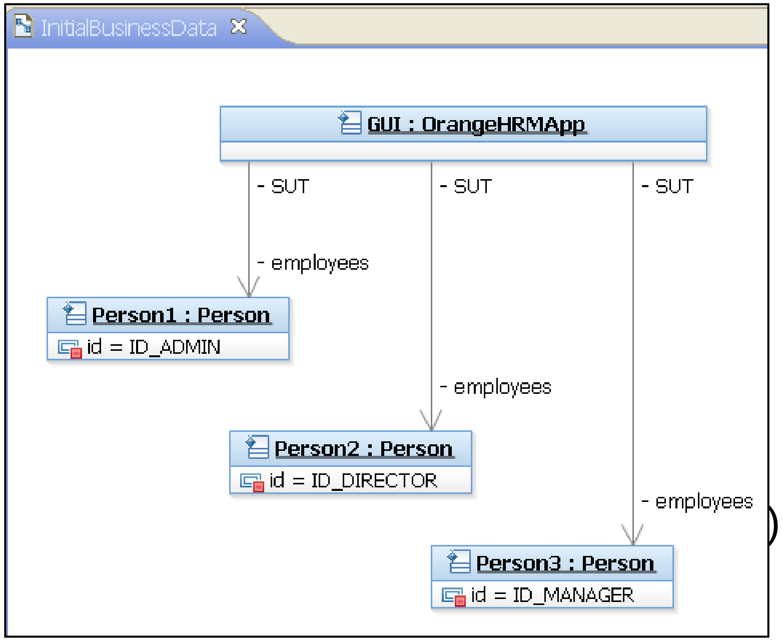
8 - Defining the initial state of the system

- Modeled as a package containing the actual objects (instances of classes) to use in the test generation
  - Possible to define different sets of objects
  - Generated tests may differ from one set of objects to another
  - Impact of the test strategy in the object definition (see next slide)
  - Mapped to the physical objects during test execution



Person

id : E_EMPLOYEE_ID
remainingDays : Integer
takenDays : Integer
scheduledDays : Integer
hasRequestedLeave : Boolean



InitialBusinessData

GUI : OrangeHRMApp

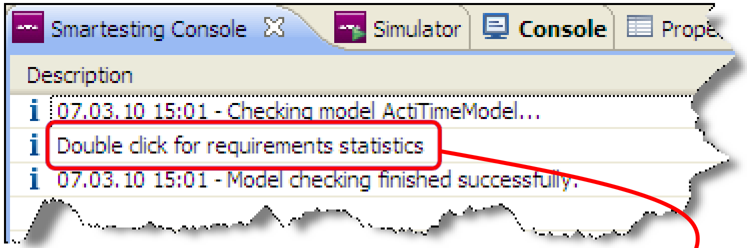
Person1 : Person
id = ID_ADMIN

Person2 : Person
id = ID_DIRECTOR

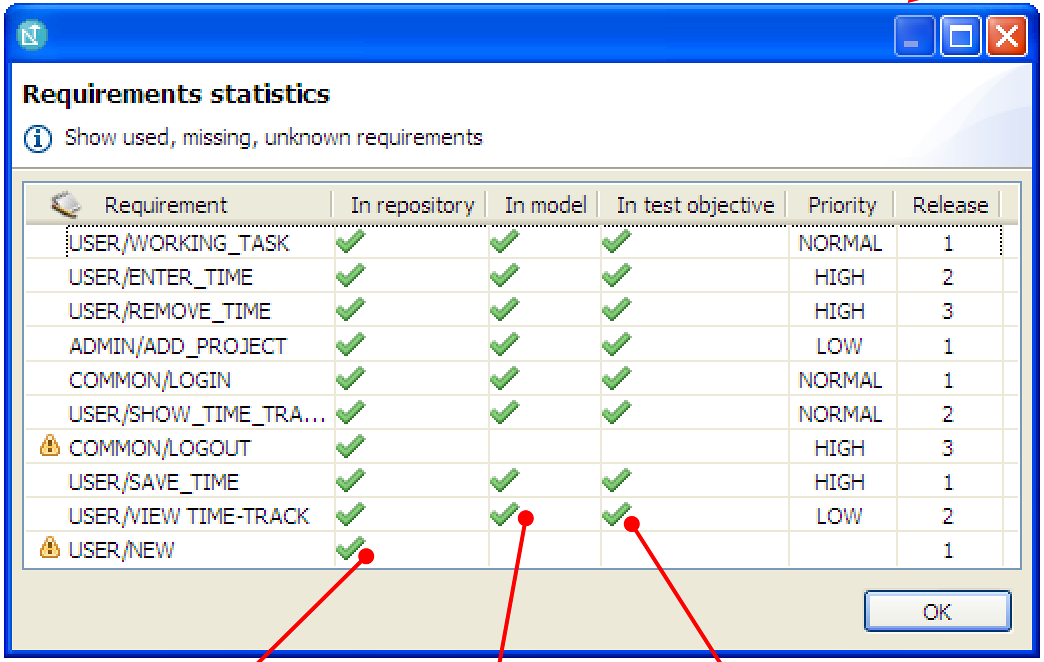
Person3 : Person
id = ID_MANAGER

Validate Model:
9 - Requirement coverage

- Tag statistics in the modeling tool
- Traceability links
  - In Smartesting CertifyIt
  - In the target test management tool



i 07.03.10 15:01 - Checking model ActiTimeModel...
i Double click for requirements statistics
i 07.03.10 15:01 - Model checking finished successfully.



Requirements statistics

ⓘ Show used, missing, unknown requirements

Requirement	In repository	In model	In test objective	Priority	Release
USER/WORKING_TASK	✓	✓	✓	NORMAL	1
USER/ENTER_TIME	✓	✓	✓	HIGH	2
USER/REMOVE_TIME	✓	✓	✓	HIGH	3
ADMIN/ADD_PROJECT	✓	✓	✓	LOW	1
COMMON/LOGIN	✓	✓	✓	NORMAL	1
USER/SHOW_TIME_TRA...	✓	✓	✓	NORMAL	2
⚠ COMMON/LOGOUT	✓			HIGH	3
USER/SAVE_TIME	✓	✓	✓	HIGH	1
USER/VIEW TIME-TRACK	✓	✓	✓	LOW	2
⚠ USER/NEW	✓				1

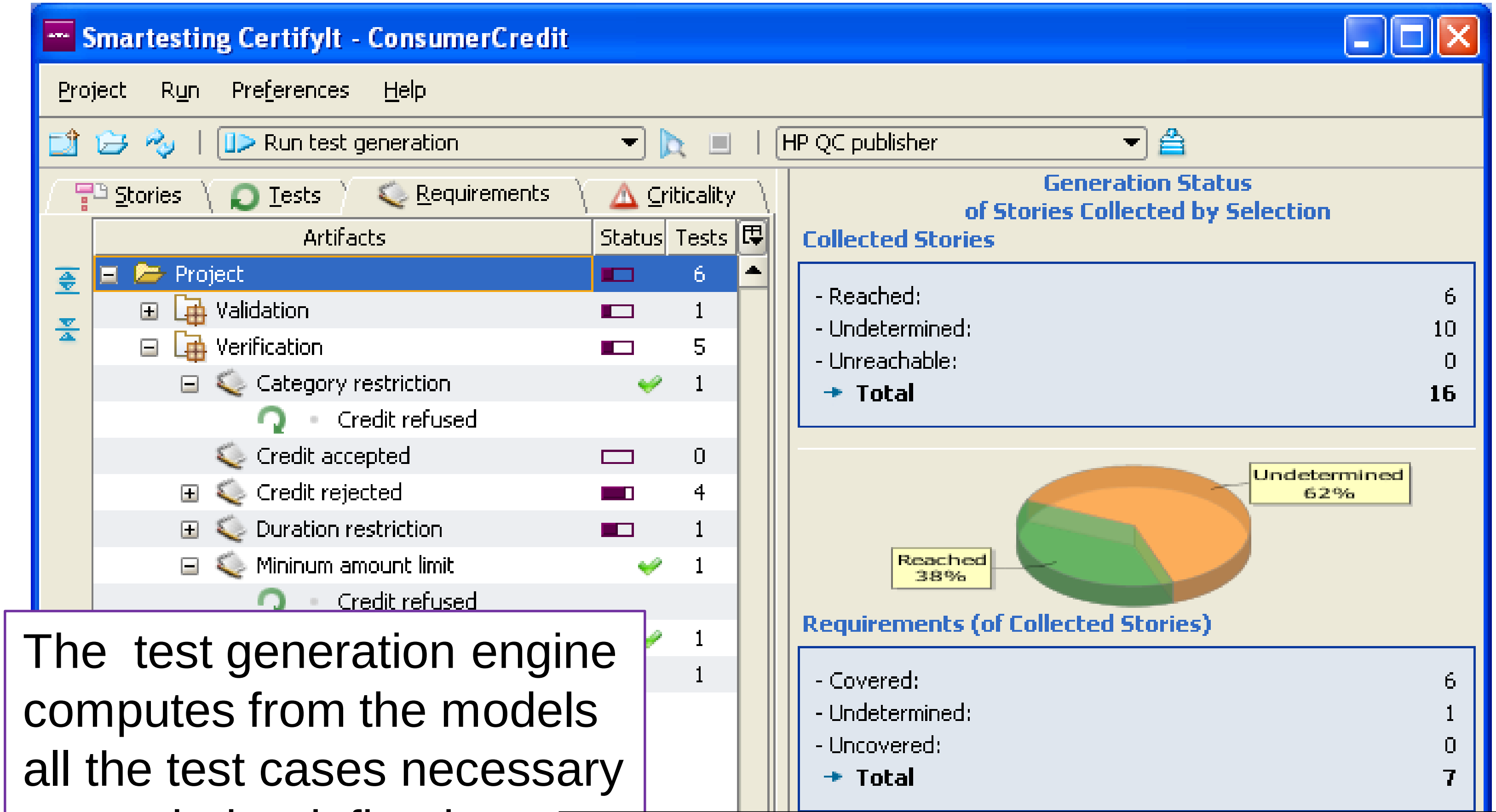
Smartesting maintains a matrix of requirements covered in the test model
Can be accessed from the Smartesting console when checking/exporting the model

Indicates the requirement is included in the input requirement list

Indicates that the requirement is covered in the model

The requirement is included in the set that will be exported

Automated test generation



Smartesting CertifyIt - ConsumerCredit

Project Run Preferences Help

Run test generation | HP QC publisher

Stories | Tests | Requirements | Criticality

Artifacts	Status	Tests
Project	▬	6
Validation	▬	1
Verification	▬	5
Category restriction	✓	1
• Credit refused		
Credit accepted	▭	0
Credit rejected	▬	4
Duration restriction	▬	1
Mininum amount limit	✓	1
• Credit refused		
	✓	1
		1

Generation Status
of Stories Collected by Selection

Collected Stories

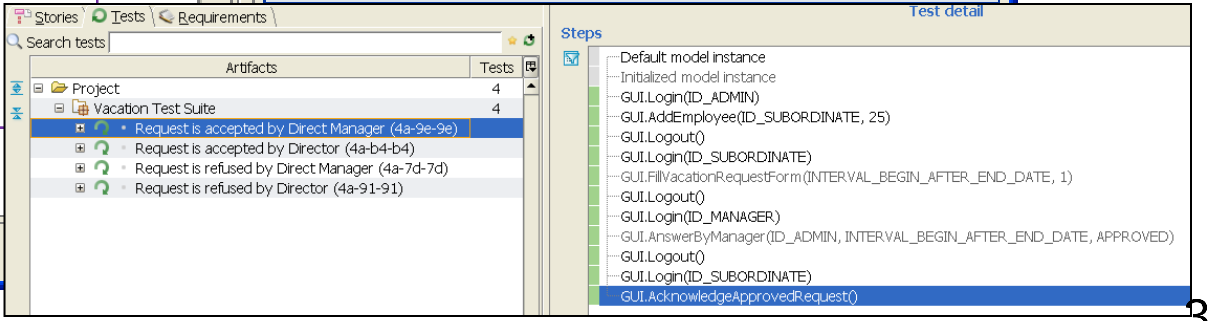
- Reached:	6
- Undetermined:	10
- Unreachable:	0
→ **Total**	**16**

Requirements (of Collected Stories)

- Covered:	6
- Undetermined:	1
- Uncovered:	0
→ **Total**	**7**

1 : Console

The test generation engine computes from the models all the test cases necessary to reach the defined coverage



Stories | Tests | Requirements

Search tests

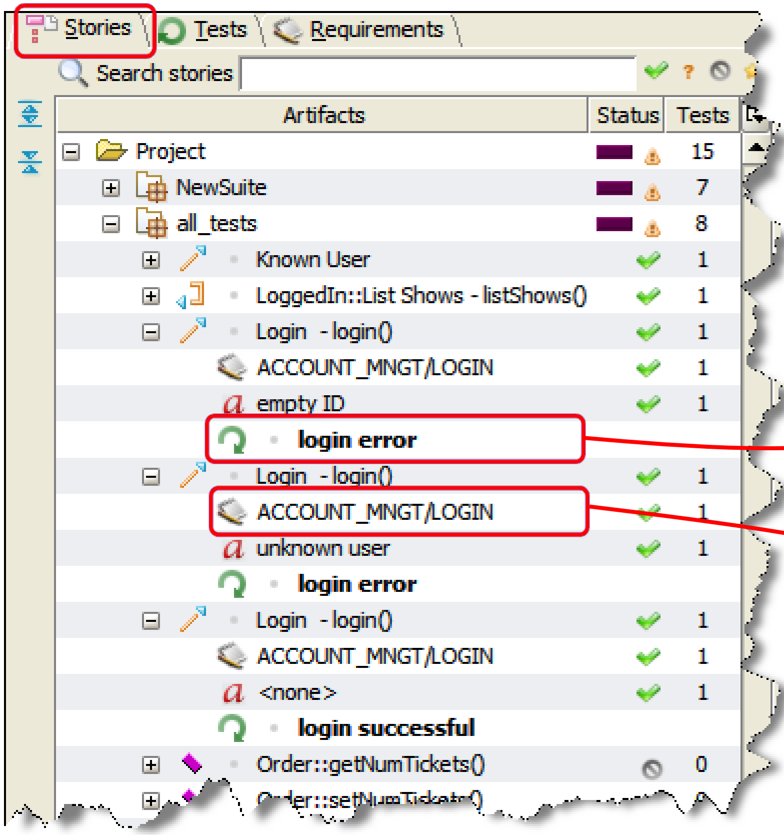
Artifacts	Tests
Project	4
Vacation Test Suite	4
• Request is accepted by Direct Manager (4a-9e-9e)	
• Request is accepted by Director (4a-b4-b4)	
• Request is refused by Direct Manager (4a-7d-7d)	
• Request is refused by Director (4a-91-91)	

Test detail

Steps

```
Default model instance
Initialized model instance
GUI.Login(ID_ADMIN)
GUI.AddEmployee(ID_SUBORDINATE, 25)
GUI.Logout()
GUI.Login(ID_SUBORDINATE)
GUI.FillVacationRequestForm(INTERVAL_BEGIN_AFTER_END_DATE, 1)
GUI.Logout()
GUI.Login(ID_MANAGER)
GUI.AnswerByManager(ID_ADMIN, INTERVAL_BEGIN_AFTER_END_DATE, APPROVED)
GUI.Logout()
GUI.Login(ID_SUBORDINATE)
GUI.AcknowledgeApprovedRequest()
```

Requirements in Smartesting CertifyIt

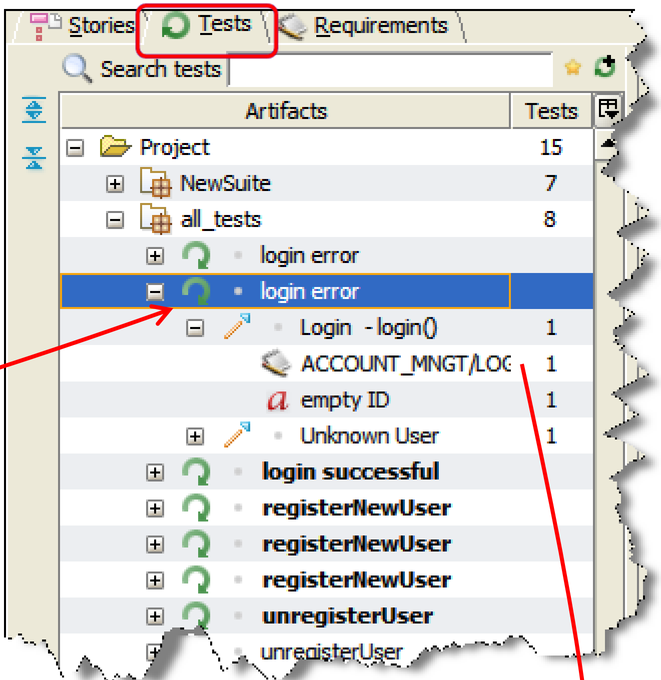


Stories | Tests | Requirements

Search stories

Artifacts	Status	Tests
⊟ Project	▬ ⚠	15
⊞ NewSuite	▬ ⚠	7
⊟ all_tests	▬ ⚠	8
⊞ Known User	✓	1
⊞ LoggedIn::List Shows - listShows()	✓	1
⊟ Login - login()	✓	1
ACCOUNT_MNGT/LOGIN	✓	1
empty ID	✓	1
login error		
⊟ Login - login()	✓	1
ACCOUNT_MNGT/LOGIN	✓	1
unknown user	✓	1
login error		
⊟ Login - login()	✓	1
ACCOUNT_MNGT/LOGIN	✓	1
<none>	✓	1
login successful		
⊞ Order::getNumTickets()	⊘	0
⊞ Order::setNumTickets()		0

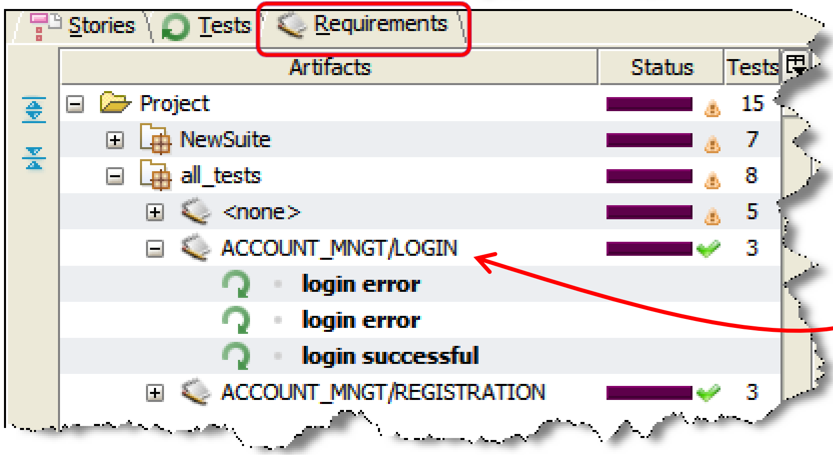
Stories, tests and requirements are automatically linked together in Smartesting



Stories | Tests | Requirements

Search tests

Artifacts	Tests
⊟ Project	15
⊞ NewSuite	7
⊟ all_tests	8
⊞ login error	
⊟ login error	
⊟ Login - login()	1
ACCOUNT_MNGT/LOG	1
empty ID	1
⊞ Unknown User	1
⊞ **login successful**	
⊞ **registerNewUser**	
⊞ **registerNewUser**	
⊞ **registerNewUser**	
⊞ **unregisterUser**	
⊞ unregisterUser	



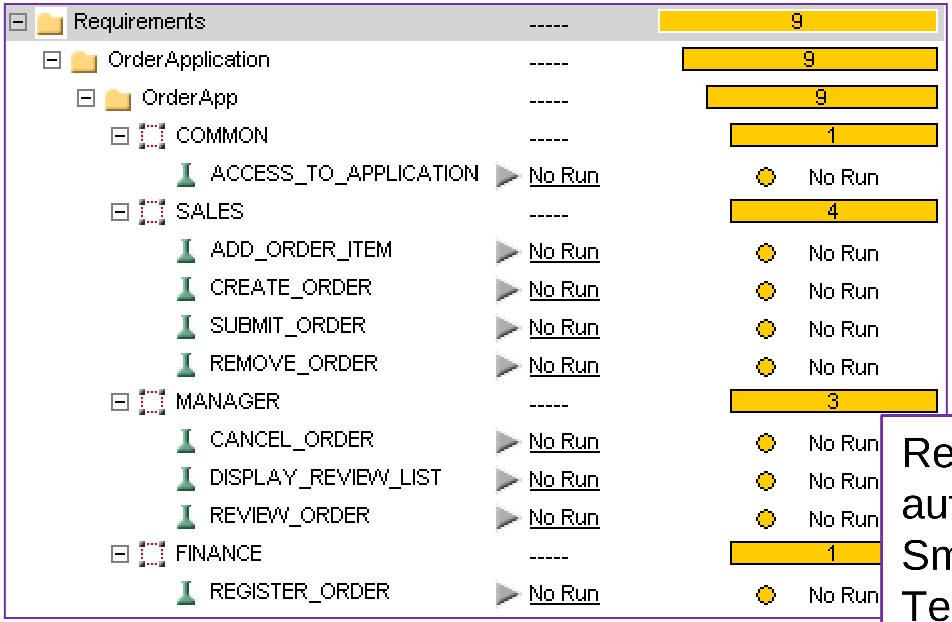
Stories | Tests | Requirements

Artifacts	Status	Tests
⊟ Project	▬ ⚠	15
⊞ NewSuite	▬ ⚠	7
⊟ all_tests	▬ ⚠	8
⊞ <none>	▬ ⚠	5
⊟ ACCOUNT_MNGT/LOGIN	▬ ✓	3
login error		
login error		
login successful		
⊞ ACCOUNT_MNGT/REGISTRATION	▬ ✓	3

The Stories view displays all the stories (i.e. the test objectives) computed by Smartesting CertifyIt

Execute Tests:
10 - Test Publication

- Target testing environment with requirement management traceability information if available
- Synchronization: Obsolete tests are deleted or replaced
- Publication can be customized (Java code provided, open API of the Smartesting publisher provided)



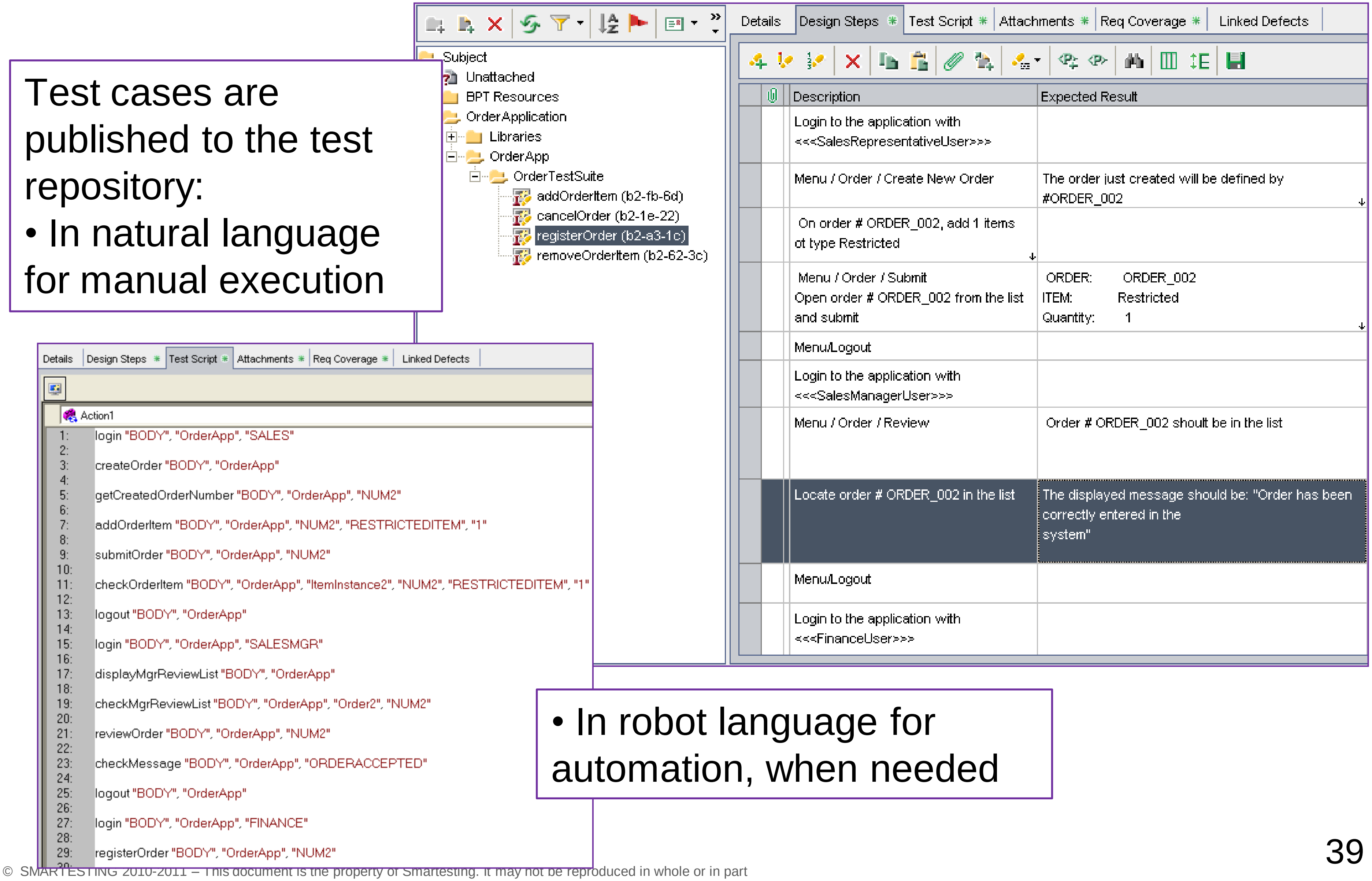
⊟ Requirements	-----	9
⊟ OrderApplication	-----	9
⊟ OrderApp	-----	9
⊟ COMMON	-----	1
ACCESS_TO_APPLICATION	▶ No Run	● No Run
⊟ SALES	-----	4
ADD_ORDER_ITEM	▶ No Run	● No Run
CREATE_ORDER	▶ No Run	● No Run
SUBMIT_ORDER	▶ No Run	● No Run
REMOVE_ORDER	▶ No Run	● No Run
⊟ MANAGER	-----	3
CANCEL_ORDER	▶ No Run	● No Run
DISPLAY_REVIEW_LIST	▶ No Run	● No Run
REVIEW_ORDER	▶ No Run	● No Run
⊟ FINANCE	-----	1
REGISTER_ORDER	▶ No Run	● No Run

Requirements traceability is automatically managed by Smartesting and maintained in the Test Repository for coverage analysis and impact analysis

Publication to the Test Repository for Test Execution

Test cases are published to the test repository:

- In natural language for manual execution

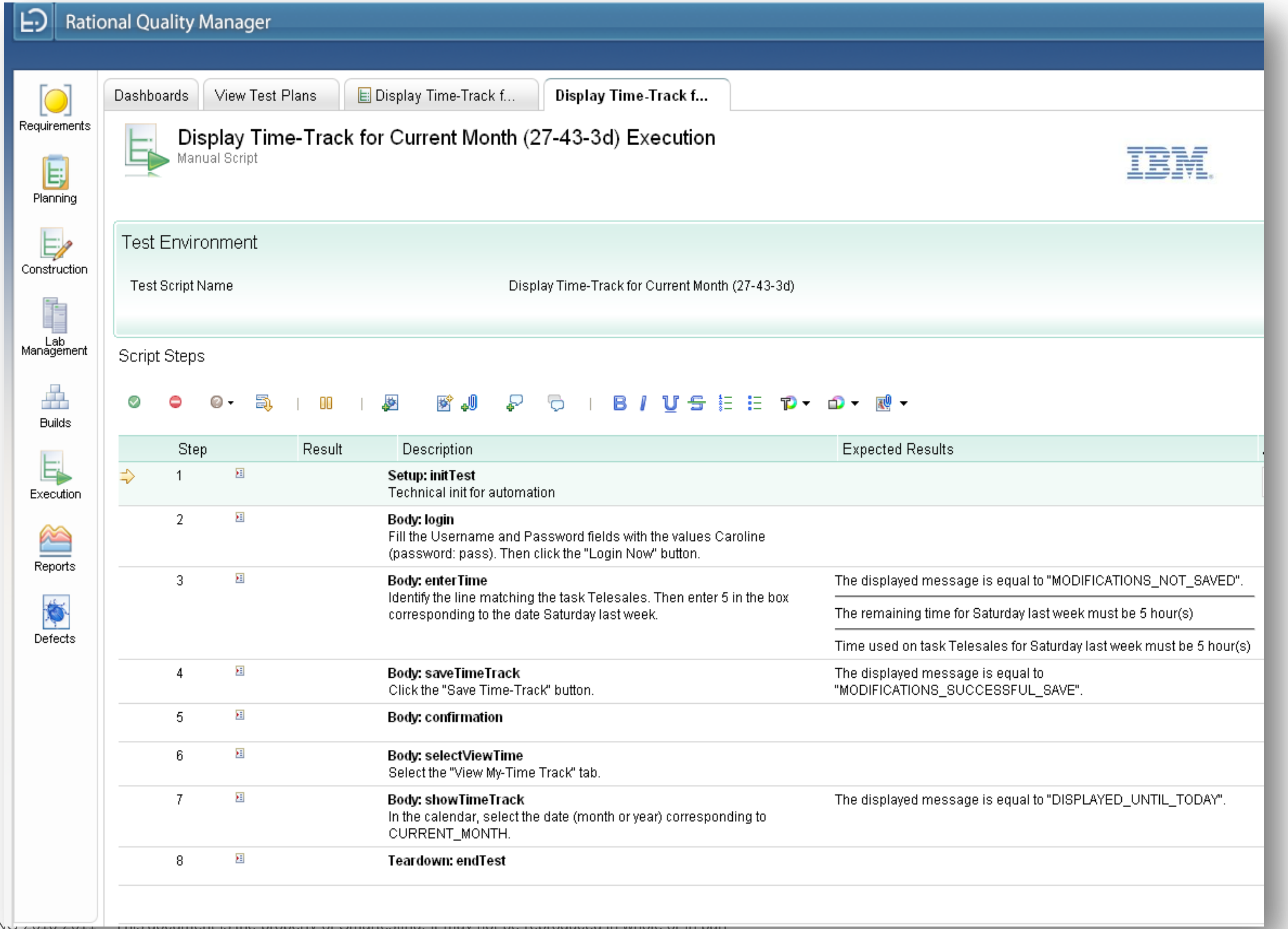


Description	Expected Result
Login to the application with <<<SalesRepresentativeUser>>>	
Menu / Order / Create New Order	The order just created will be defined by #ORDER_002
On order # ORDER_002, add 1 items ot type Restricted	
Menu / Order / Submit Open order # ORDER_002 from the list and submit	ORDER: ORDER_002 ITEM: Restricted Quantity: 1
Menu/Logout	
Login to the application with <<<SalesManagerUser>>>	
Menu / Order / Review	Order # ORDER_002 shoult be in the list
Locate order # ORDER_002 in the list	The displayed message should be: "Order has been correctly entered in the system"
Menu/Logout	
Login to the application with <<<FinanceUser>>>	

```
login "BODY", "OrderApp", "SALES"

createOrder "BODY", "OrderApp"

getCreatedOrderNumber "BODY", "OrderApp", "NUM2"

addOrderItem "BODY", "OrderApp", "NUM2", "RESTRICTEDITEM", "1"

submitOrder "BODY", "OrderApp", "NUM2"

checkOrderItem "BODY", "OrderApp", "ItemInstance2", "NUM2", "RESTRICTEDITEM", "1"

logout "BODY", "OrderApp"

login "BODY", "OrderApp", "SALESMGR"

displayMgrReviewList "BODY", "OrderApp"

checkMgrReviewList "BODY", "OrderApp", "Order2", "NUM2"

reviewOrder "BODY", "OrderApp", "NUM2"

checkMessage "BODY", "OrderApp", "ORDERACCEPTED"

logout "BODY", "OrderApp"

login "BODY", "OrderApp", "FINANCE"

registerOrder "BODY", "OrderApp", "NUM2"
```

- In robot language for automation, when needed

Generated Repository is complete, executable and fully documented



Display Time-Track for Current Month (27-43-3d) Execution

Manual Script

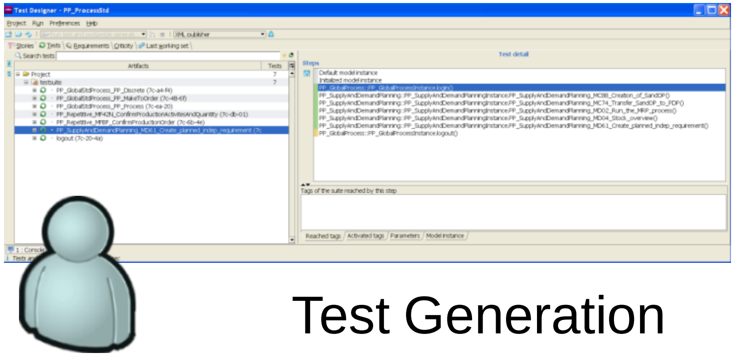
Test Environment

Test Script Name: Display Time-Track for Current Month (27-43-3d)

Script Steps

Step	Result	Description	Expected Results
1		**Setup: initTest** Technical init for automation	
2		**Body: login** Fill the Username and Password fields with the values Caroline (password: pass). Then click the "Login Now" button.	
3		**Body: enterTime** Identify the line matching the task Telesales. Then enter 5 in the box corresponding to the date Saturday last week.	The displayed message is equal to "MODIFICATIONS_NOT_SAVED". The remaining time for Saturday last week must be 5 hour(s) Time used on task Telesales for Saturday last week must be 5 hour(s)
4		**Body: saveTimeTrack** Click the "Save Time-Track" button.	The displayed message is equal to "MODIFICATIONS_SUCCESSFUL_SAVE".
5		**Body: confirmation**	
6		**Body: selectViewTime** Select the "View My-Time Track" tab.	
7		**Body: showTimeTrack** In the calendar, select the date (month or year) corresponding to CURRENT_MONTH.	The displayed message is equal to "DISPLAYED_UNTIL_TODAY".
8		**Teardown: endTest**	

MBT for Automated Testing – Reuse of existing test components (‘HP BPT’ like approach)


Test Designer - PP_ProcessStd
Project Run Preferences Help
XML publisher
Stories Tests Requirements Criticity Last working set
Search tests
Artifacts
Tests
Test detail
Steps
Tags of the suite reached by this step
Reached tags
Activated tags
Parameters
Model instance
1 : Console

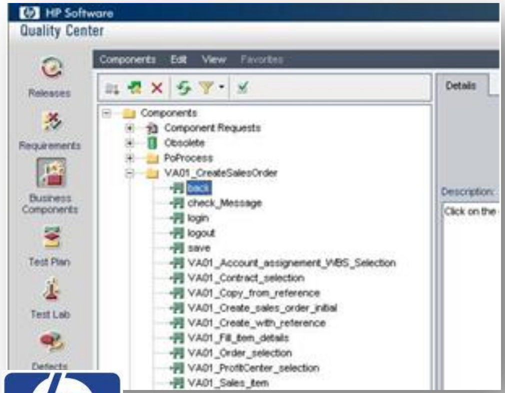
- Project 7
  - testsuite 7
    - PP_GlobalStdProcess_PP_Discrete (7c-a4-f4)
    - PP_GlobalStdProcess_PP_MakeToOrder (7c-48-6f)
    - PP_GlobalStdProcess_PP_Process (7c-ea-20)
    - PP_Repetitive_MF42N_ConfirmProductionActivitiesAndQuantity (7c-db-01)
    - PP_Repetitive_MFBF_ConfirmProductionOrder (7c-6b-4e)
    - PP_SupplyAndDemandPlanning_MD61_Create_planned_indep_requirement (7c
    - logout (7c-20-4a)

```
Default model instance
Initialized model instance
PP_GlobalProcess::PP_GlobalProcessInstance.login()
PP_SupplyAndDemandPlanning::PP_SupplyAndDemandPlanningInstance.PP_SupplyAndDemandPlanning_MC88_Creation_of_SandOP()
PP_SupplyAndDemandPlanning::PP_SupplyAndDemandPlanningInstance.PP_SupplyAndDemandPlanning_MC74_Transfer_SandOP_to_PDP()
PP_SupplyAndDemandPlanning::PP_SupplyAndDemandPlanningInstance.PP_SupplyAndDemandPlanning_MD02_Run_the_MRP_process()
PP_SupplyAndDemandPlanning::PP_SupplyAndDemandPlanningInstance.PP_SupplyAndDemandPlanning_MD04_Stock_overview()
PP_SupplyAndDemandPlanning::PP_SupplyAndDemandPlanningInstance.PP_SupplyAndDemandPlanning_MD61_Create_planned_indep_requirement()
PP_GlobalProcess::PP_GlobalProcessInstance.logout()
```

Test Analyst

Test Generation

HP Business Components Requests



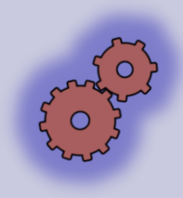
- Components
  - Component Requests
  - Obsolete
  - PoProcess
  - VA01_CreateSalesOrder
    - back
    - check_Message
    - login
    - logout
    - save
    - VA01_Account_assignement_WBS_Selection
    - VA01_Contract_selection
    - VA01_Copy_from_reference
    - VA01_Create_sales_order_initial
    - VA01_Create_with_reference
    - VA01_Fill_item_details
    - VA01_Order_selection
    - VA01_ProfitCenter_selection
    - VA01_Sales_item

HP Quality Center

Test Repository

manual tests

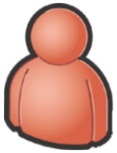
automated tests



Test automation robot

HP Quality Center

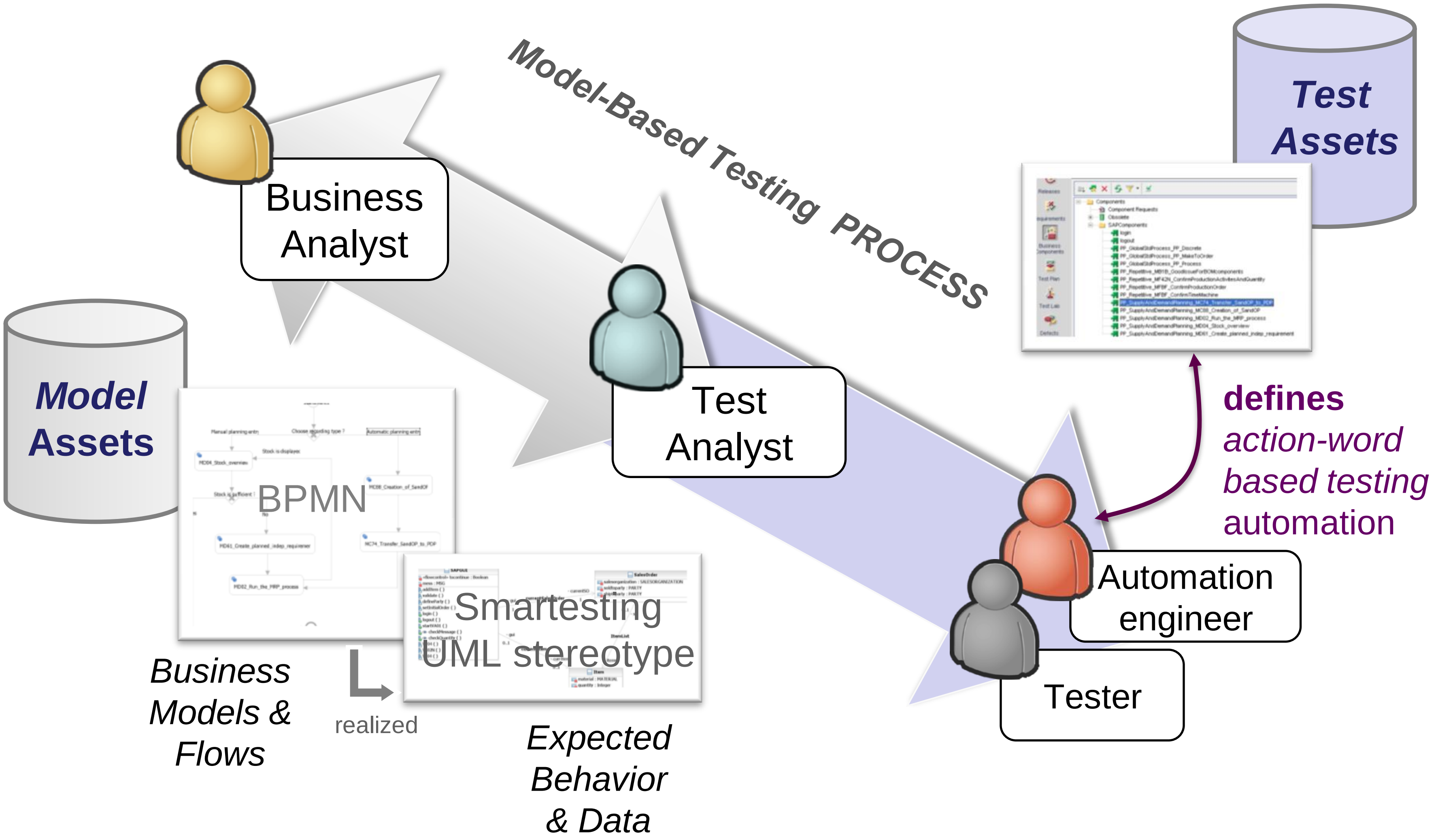
Automation engineer



Agenda

- ➲ Part 1: Principles of Model-based Testing for IT Systems…...... 3
  - Testing challenges in the context of Large-scale Enterprise Information Systems
  - Types of testing addressed by model-based testing
  - Composing Business Process models, Behavioral models and Test data models for automated test generation
- ➲ Part 2: Step-by-Step Example with Smartesting CertifyIt………14
  - Use BPMN to describe Business Processes for Test Generation
  - Link Business Processes, Business Rules and Logical Test data
  - Define Requirement Traceability information
  - Generate tests with Smartesting CertifyIt
  - Export your tests in the test repository for manual test execution
  - Generate scripts for automated test execution
- ➲ Part 3: How to start with MDT, adoption principles...……………. 44
  - Test team organization, roles and responsibilities
  - Agile and model-based testing, a perfect fit
  - Model-based testing and Risk-based testing

Roles: Separation of Concerns



Model-Based Testing PROCESS

Business Analyst

Test Analyst

Automation engineer

Tester

Model Assets

Test Assets

BPMN

Smartesting UML stereotype

realized

Business Models & Flows

Expected Behavior & Data

defines *action-word based testing* automation

Business Analyst Role and profile

Role	**Actions**	**Skills (existing / new)**
Business Analyst (BA)	▪Write and draw business needs ▪Pilot testing by the risks	✓Functional documentation capabilities ➢process notation (BPMN) ➢Test strategy
Profile	▪Knows the business processes and functional rules ▪Knows how to find missing information in the organization ▪Requirement oriented ▪Risk oriented ▪Test oriented ▪Ability to abstract (e.g. BPMN) ▪Customer oriented	

Test Analyst Role and profile

Role	**Actions**	**Skills (existing / new)**
Test Analyst (TA)	▪Apply the test strategy ▪Model application behaviors and validate specifications ▪Generate test plans ▪Pilot test execution	✓Test professional ➢Smartesting modeling and test generation
Profile	▪Good knowledge of functional testing methodologies within AGS ▪Knows Object Oriented methodology basics (or dev. experience) ▪Modeling experience can be a plus ▪Knows corp. testing tools ▪Knows project organization, lifecycle and test needs ▪Basic knowledge of test automation concerns	

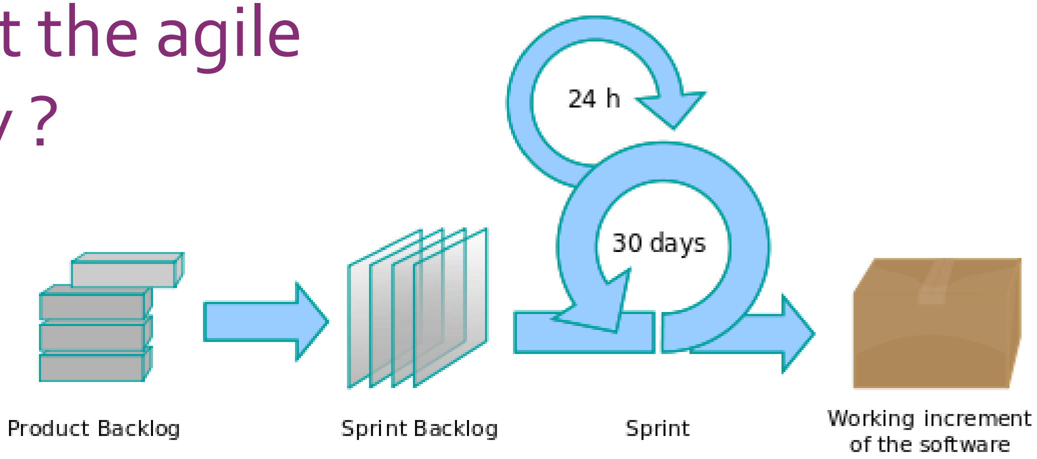
Other roles in the Smartesting project

Role	Actions	Skills (existing / new)
Project Manager (PM)	▪Manage the test team ▪Pilot testing by the risks	✓Project management capabilities ✓Functional documentation capabilities ➢process notation (BPMN) ➢Test strategy
Testers	Execute the test cases manually	✓Light knowledge on the application is a +
Automation expert	Develop the specified keyword library	✓Skills on the robot required ➢Smartesting management of technical assets
Implementation solution team	▪Develop/customize the application for the customer	✓Development ✓Customizing ✓Writing technical specification ✓Unit Test

Agile Projects Challenges

Black box functional testing is poorly used..

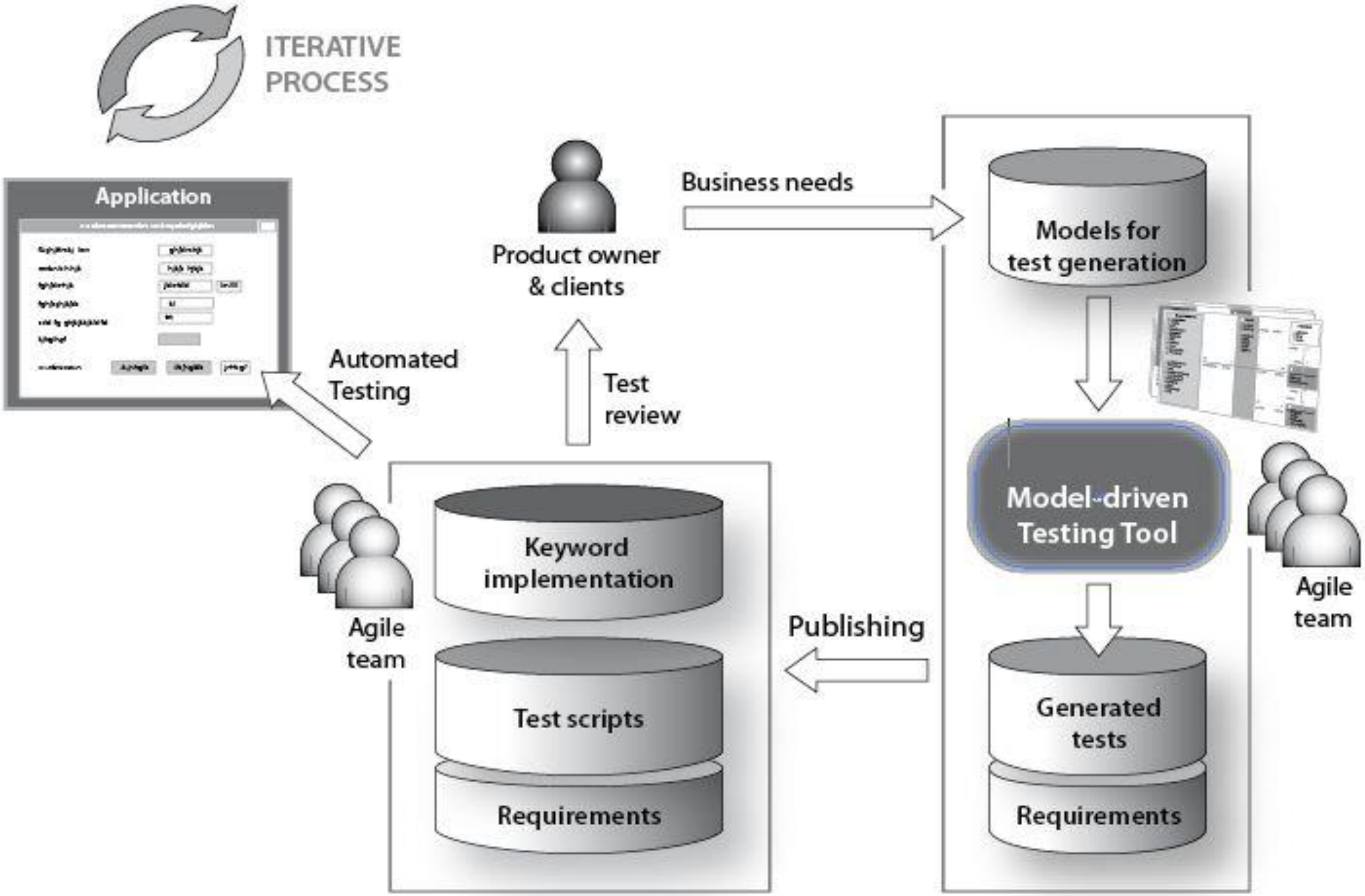
- ➲ Is it against the agile philosophy ?



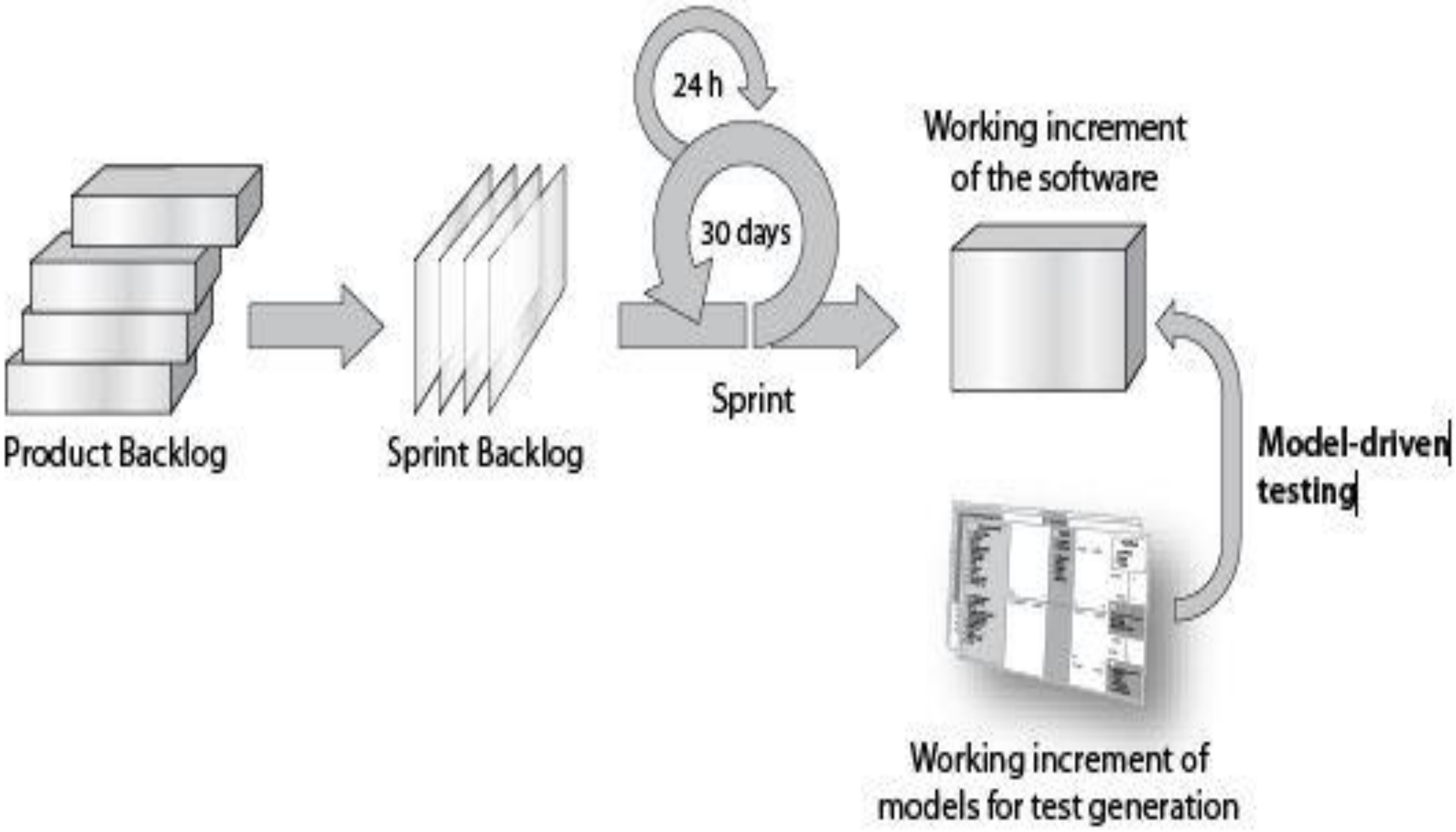
  – NO! It's just costly in **highly iterative processes**!

- ➲ Agility renews the testing challenge

Model-Based Testing in Agile team



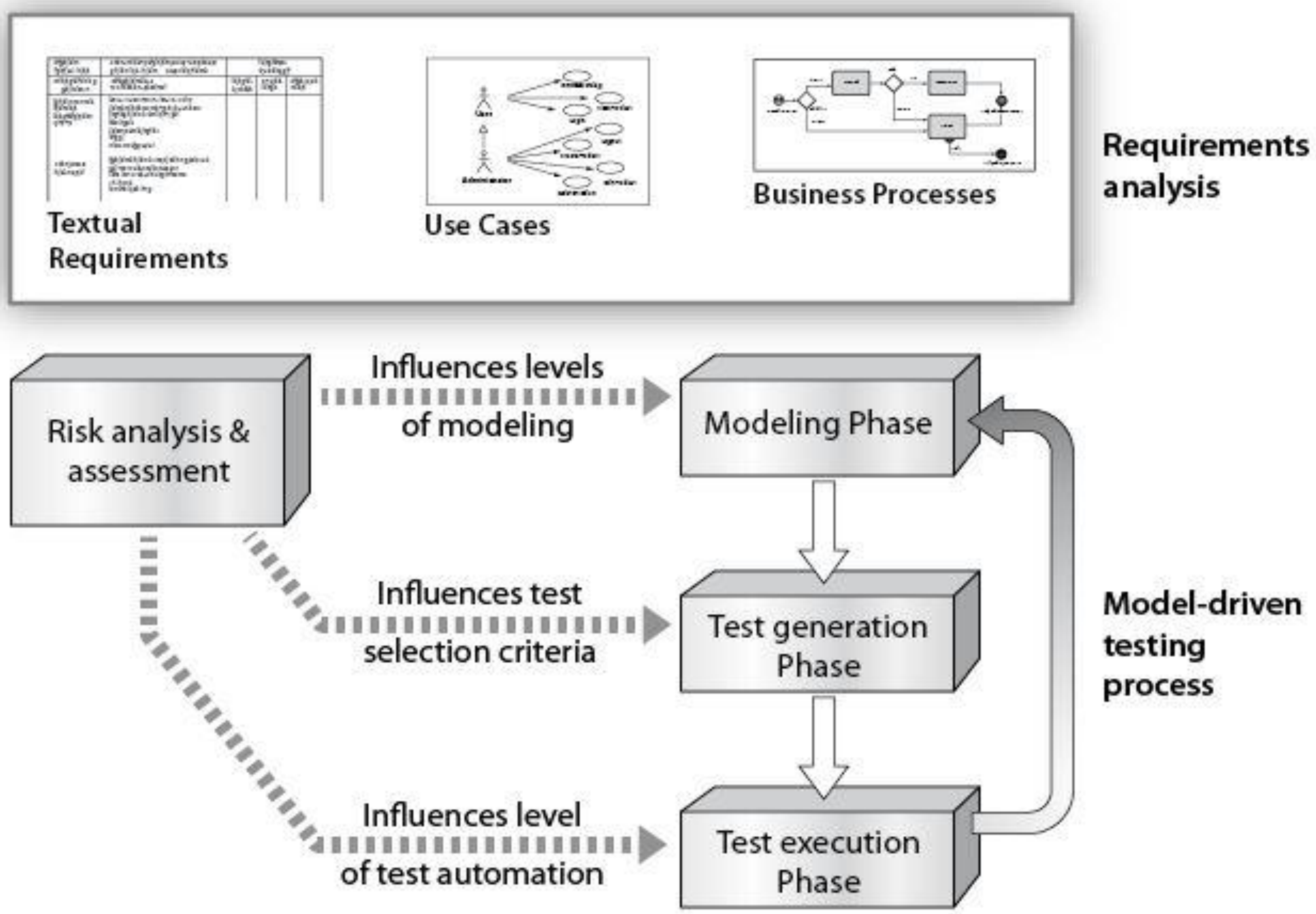
MBT in a Scrum Process



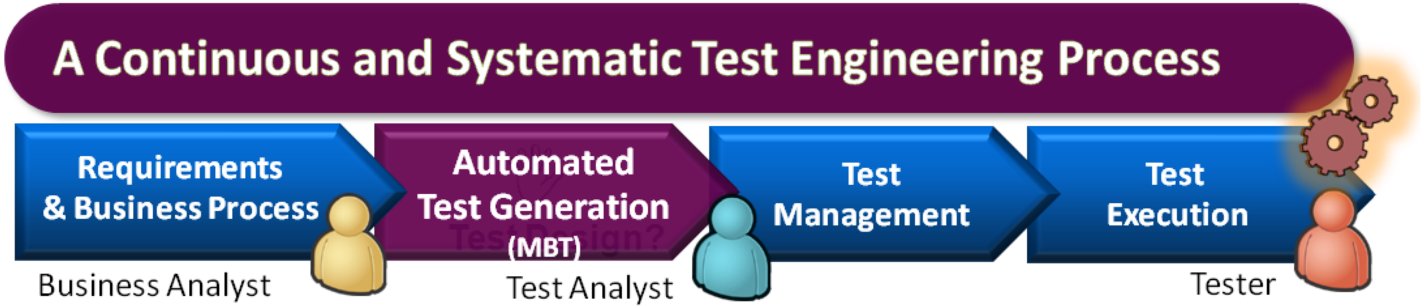
Model Based Testing gives you agility ‘*by design*’

- ➲ Responding to change
  - – Test models are easier to update than large test suites
- ➲ Individuals & interactions
  - – Testers & developers are working together
- ➲ Working software
  - – Increasing test coverage and quality
- ➲ Customer/User collaboration
  - – Test models are unambiguous communication tools

MBT and Risk-based testing


Textual Requirements
Use Cases
Business Processes
Requirements analysis
Risk analysis & assessment
Influences levels of modeling
Modeling Phase
Influences test selection criteria
Test generation Phase
Influences level of test automation
Test execution Phase
Model-driven testing process

Conclusion


A Continuous and Systematic Test Engineering Process
Requirements & Business Process
Automated Test Generation (MBT)
Test Management
Test Execution
Business Analyst
Test Analyst
Tester

- **Test case production and maintenance time accelerated**
- **Complete coverage** and traceability of selected business rules
- **Support agility** based on easier test maintenance and rapid feedback
- **Easier interaction** between Business experts and test analysts

FORRESTER®

‘Align BAs And Quality Assurance Professionals To Drive Higher Quality — And Happier Customers’

“by Mary Gerush and Margo Visitacion, Aug 2010”