smartesting®
Optimize your Test Center

Model-Driven Testing for Information Systems

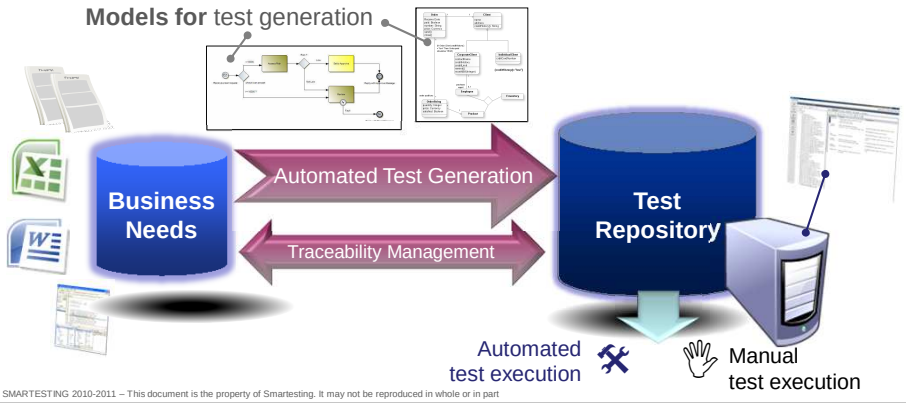
From Business Requirements to Test Repositories

Author: Jean-Pierre Schoch (pronounce "shore")
Contact: jean-pierre.schoch@smartesting.com

ICTSS – November 7, 2011

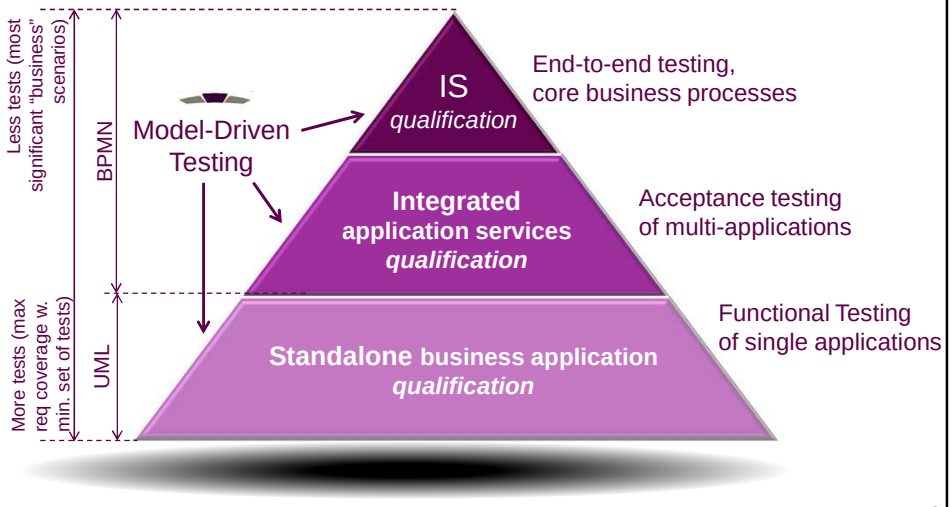
Model-Driven Testing at a Glance

- Model-Driven Testing refers to the process and techniques for:
  - The automatic derivation of abstract test cases from test generation models based on requirement models
  - The generation of concrete tests from abstract tests
  - And the manual or automated execution of the resulting concrete test cases

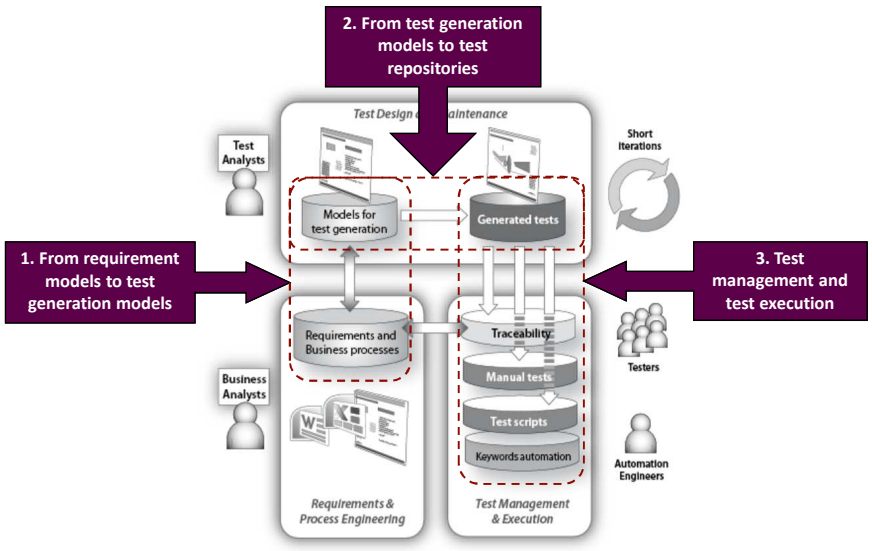


2

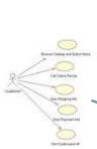
Typology of Tests for Model-Driven Testing



A Typical Model-Driven Testing Process

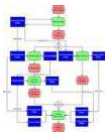


From Requirements to Test Generation Models :
1. Gathering Requirements


Use Cases

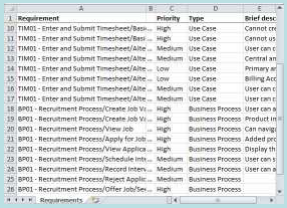

Textual Requirements


Application Mockups


Business Processes


And all Other Sources…

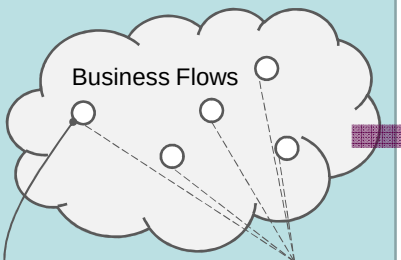
Test Objective Charter



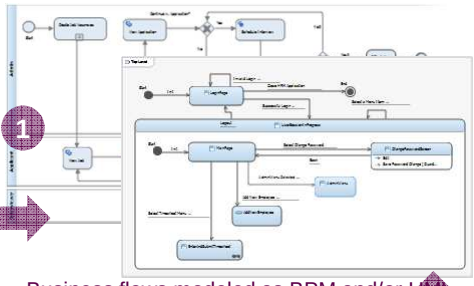
- Single reference of "test" requirements
- Can be exported from existing requirement repositories
- Includes attributes such as priority, criticality, target release, etc.
- The TODO list of the modeling team

From Requirements to Test Generation Models:
2. Modeling Behavior: Business Flows and Rules

A Schematic View of the Application to Test



Business Rules
X …
Y …
Z …
…

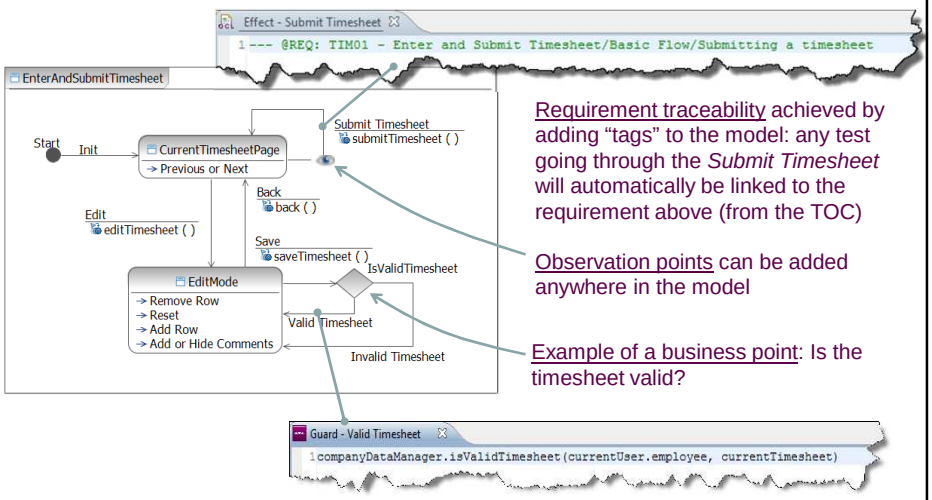


Business flows modeled as BPM and/or UML State Machine diagrams

Business rules modeled as operations of UML classes (business entities) and OCL pre-/post conditions

From Requirements to Test Generation Models: 2. Modeling Behavior: Refining the Model

A user can edit a "timesheet", make changes and save the timesheet



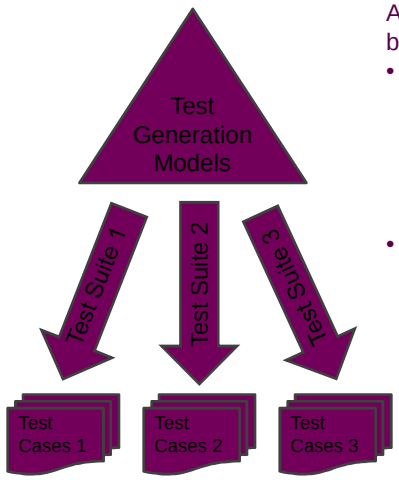
Requirement traceability achieved by adding "tags" to the model: any test going through the *Submit Timesheet* will automatically be linked to the requirement above (from the TOC)

Observation points can be added anywhere in the model

Example of a business point: Is the timesheet valid?

7

From Test Generation Models to Test Repositories: 1. Test Suites

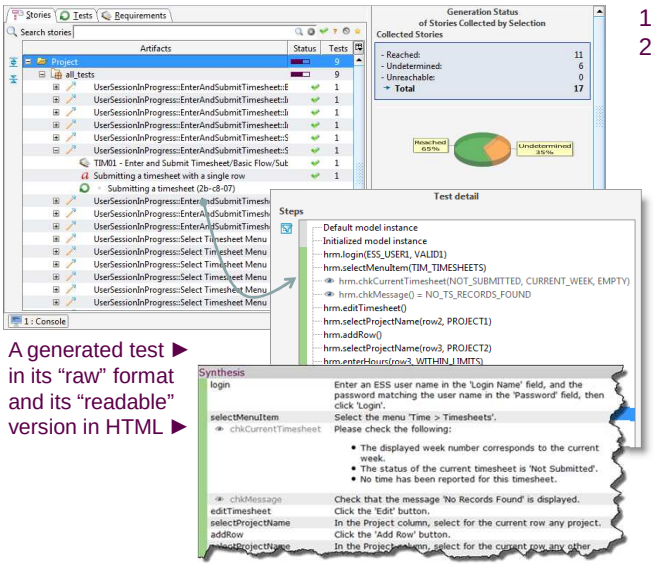


A "Smartsuite" produces test cases from a behavioral model based on:

- A given set of logical test data
  - Logical test data based on *equivalence partitioning* and equivalence classes
  - Strategies: no predefined data; predefined in the model; imposed test data; hybrid solutions
- Test selection criteria
  - Mostly *coverage-based criteria* thru the systematic use of tags (e.g. to select only one function, or only nominal cases)
  - *Pair-wise testing* through the use of decision tables
  - *Scenario-based testing*: custom scenarios consistent with the modeled behavior
  - *Business scenarios* (BPMN only): BAs specify mandatory steps (like in a GPS!)

8

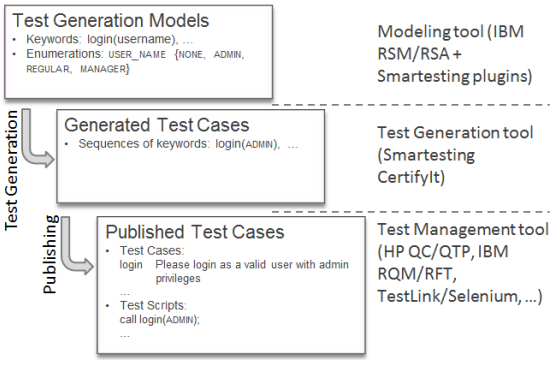
From Test Generation Models to Test Repositories: 2. Generating Test Cases



A generated test ► in its "raw" format and its "readable" version in HTML ►

1. Generating (auto)
2. Managing Test Cases:
   - Naming test cases
   - Checking the requirement coverage
   - Creating a readable form (for validation by BAs and other functional experts)
   - Simulating

Test Management and Test Execution



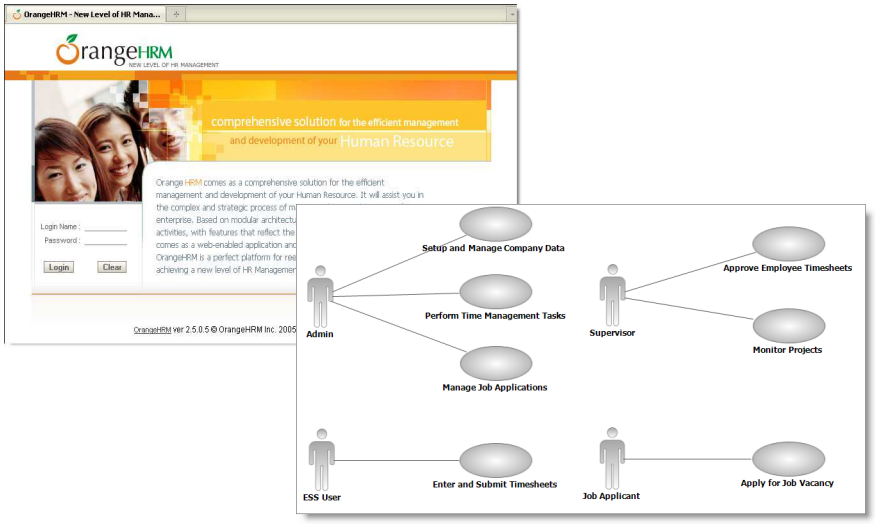
What is published ?

- The full test cases with the test steps and expected results
- The traceability links (if supported by the test management tool/ environment)
- The test scripts (if you intend to automate your tests): the test scripts are the computer-version of the test cases in the language supported by your test management tool/environment

Managing Tests in the Test Management Tool

- Most of the standard/existing procedures still apply
- Most significant difference: tests no longer created directly in the test management tool and changes to the test cases must be systematically reported in the original test generation models

Sample Application: OrangeHRM

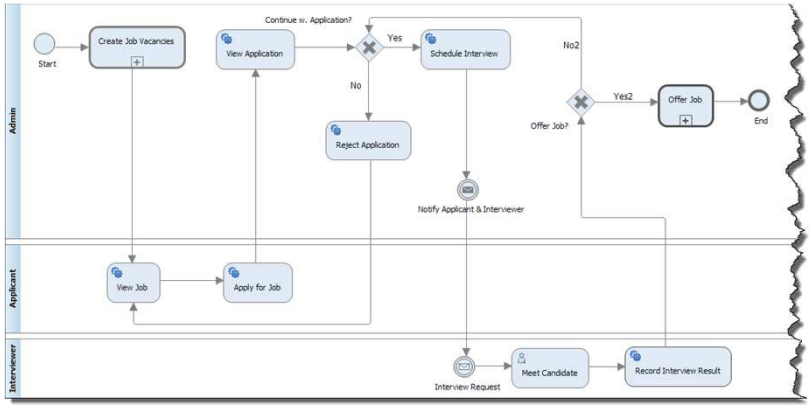


Tutorial Part 1: UML

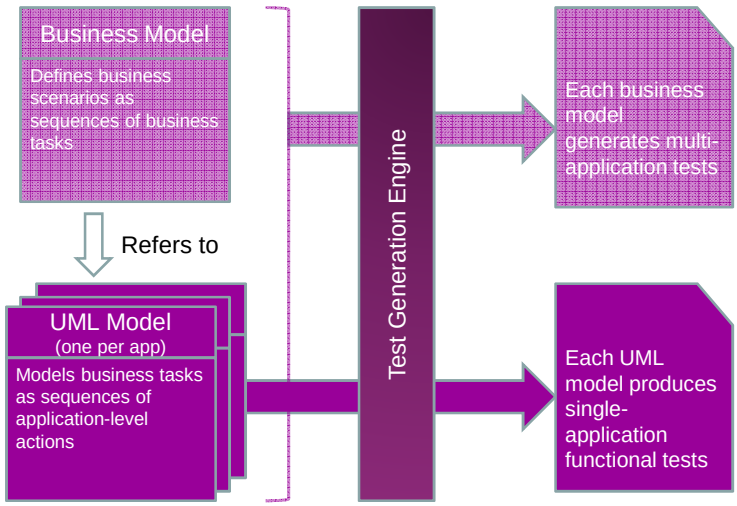
- Presentation of the Sample Application:
  - UC Scenario: Regular User logs in, then enters and submits a timesheet
- Presentation of the UML model and generated tests:
  - Presenting the UML Model
  - Inspecting the Test Objective Charter
  - Generating and inspecting the Test Cases
  - Publishing in HTML Format
- Making a Change to the Model
  - Making a Change to the Model
  - Regenerating the Test Cases
  - Adding an Observation

Business Process Modeling

- BPMN = Business Process Modeling Notation
  - A standard notation that is readily understandable by all business and technical users
  - Dedicated to describing Business Processes



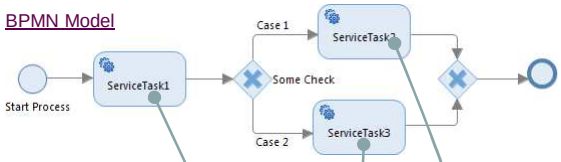
Combining BPMN and UML



14

Combining BPMN and UML – An Example

BPMN Model

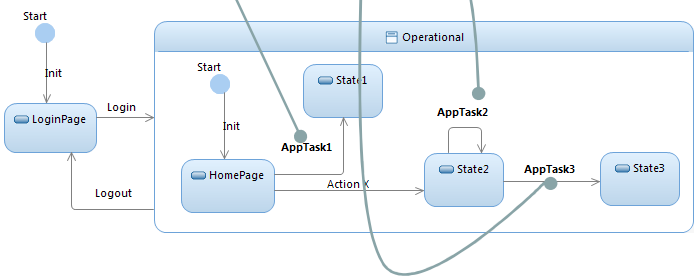


ServiceTask*i* is mapped to AppTask*i*

Here is one possible business scenario (business analyst view):

- Start
- ServiceTask1
- ServiceTask3
- End

UML Model



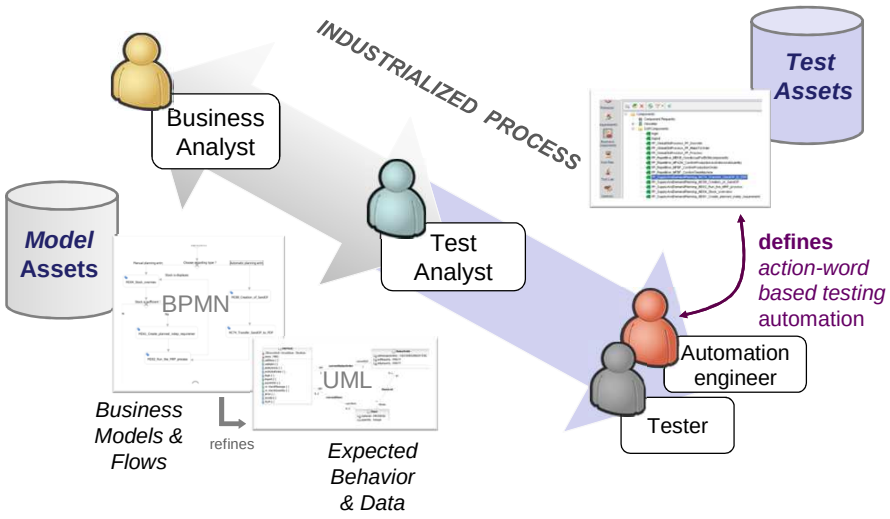
Here is the "tester" version (test analyst view):

- Login
- AppTask1
- Logout
- Login
- Action X
- AppTask3
- End

Tutorial Part 2: BPMN

- Presentation of the *Recruitment* Business Process
  - The Business Process Model
  - The Business Scenarios
- Creating a New Business Process

Roles in the Model-Driven Testing Process

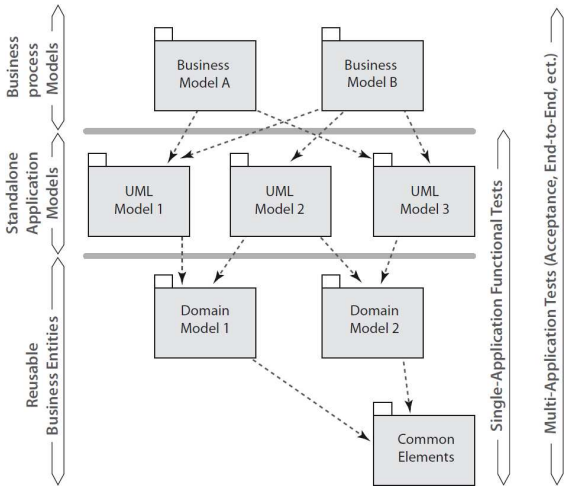


Impact of Model-Driven Testing

Impact on…	
Test Case Organization	1. Map the MDT test suites to the test plan organization 2. Customize the publisher as needed 3. Consider adapting your test plan to the MDT approach
Requirement Management	1. Incentive to properly manage requirements! 2. Create a Test Objective Charter 3. Traceability links maintained automatically
Test Data Management	1. Must map logical data from the models to physical data in the test environment
Test Automation	1. Test Script design is already done! 2. Recommended systematic use of “data tables” (or equivalent) to map logical data to physical data
Test Execution	1. No direct impact on test execution 2. Changes to the test cases must be systematically reported in the original test generation models 3. Use of a defect tracking system recommended
Test Maintenance	1. Time to implement a functional change or evolution is independent of the number of impacted test cases!

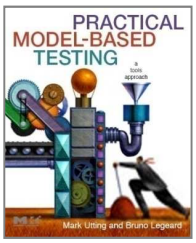
Scaling Up … Large Systems

- Use a layered architecture
- Promote reuse of common elements
- Use a configuration management and versioning tool to enable true team work
- New role: test architect (or test architecting team) to:
  - Define the overall model-driven testing process
  - Define the global system architecture and guidelines
  - Define the expected level of reuse
  - Track project progress

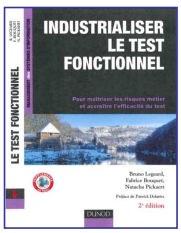


smartesting®
Optimize your Test Center

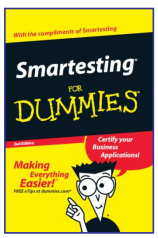
Thank you for your attention



Morgan&Kauffman
Dec. 2006



DUNOD
2nd Edition
Nov. 2011



Smartesting
Case Study
Aug. 2011

twitter
twitter.com/smartesting

LinkedIn
MBT Group