4th UCAAT User Conference on Advanced Automated Testing

ETSI World Class Standards

SECURITY TESTING USING MODELS AND TEST PATTERNS

Presented by [Bruno Legeard, Elizabeta Fourneret]

ARMOUR

smartesting Optimize your Test Center

4th UCAAT User Conference on Advanced Automated Testing

ETSI World Class Standards

MODEL-BASED SECURITY TESTING

Positionning with respect to the state of the art

ARMOUR

smartesting Optimize your Test Center

Model-Based Testing

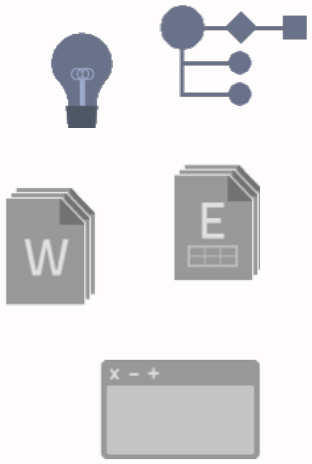
- Model-Based Testing (MBT) is based or involved on models, called MBT models

ISTQB

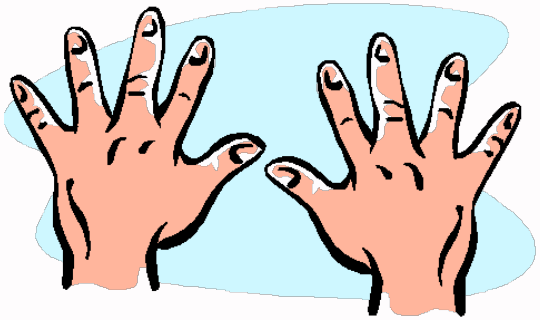
- It extends and supports classic test design techniques integrating closely with the existing lifecycle in an enterprise.

MBT Process

Functional needs
Business needs
Requirements



Test design and implementation



Test Repository
(Excel, HP/ALM…)

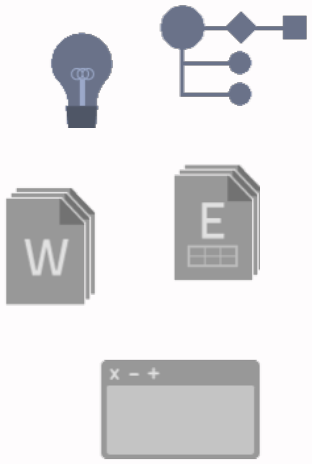
Functional tests

Manual execution

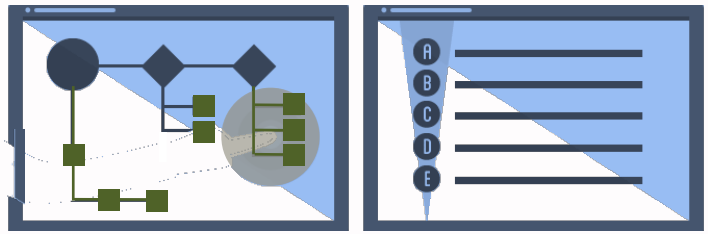
& scripts for automation

MBT Process

Functional needs
Business needs
Requirements



Modeling for test generation



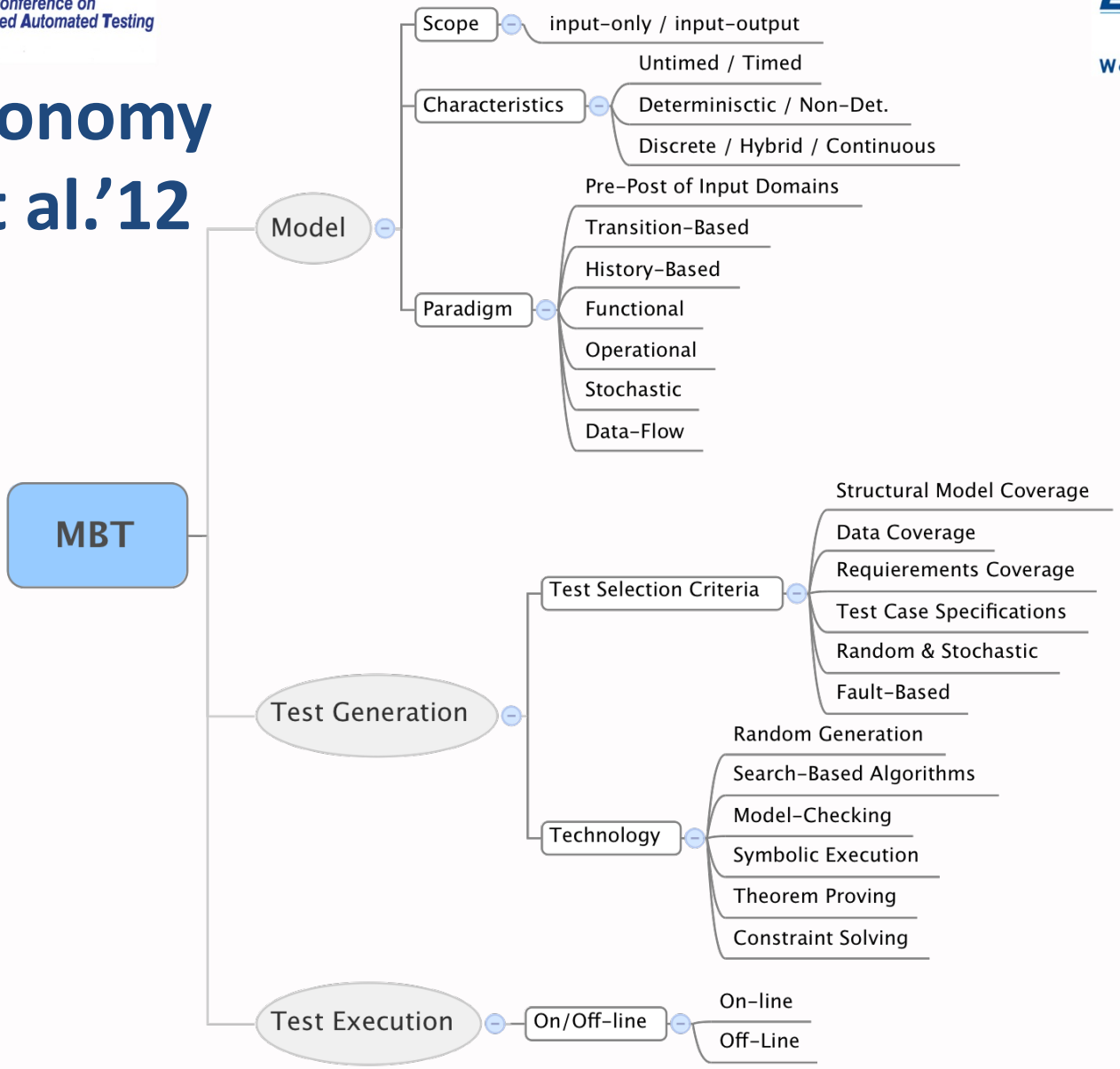
Automatic or manual test conception

Test Repository
(Excel, HP/ALM…)

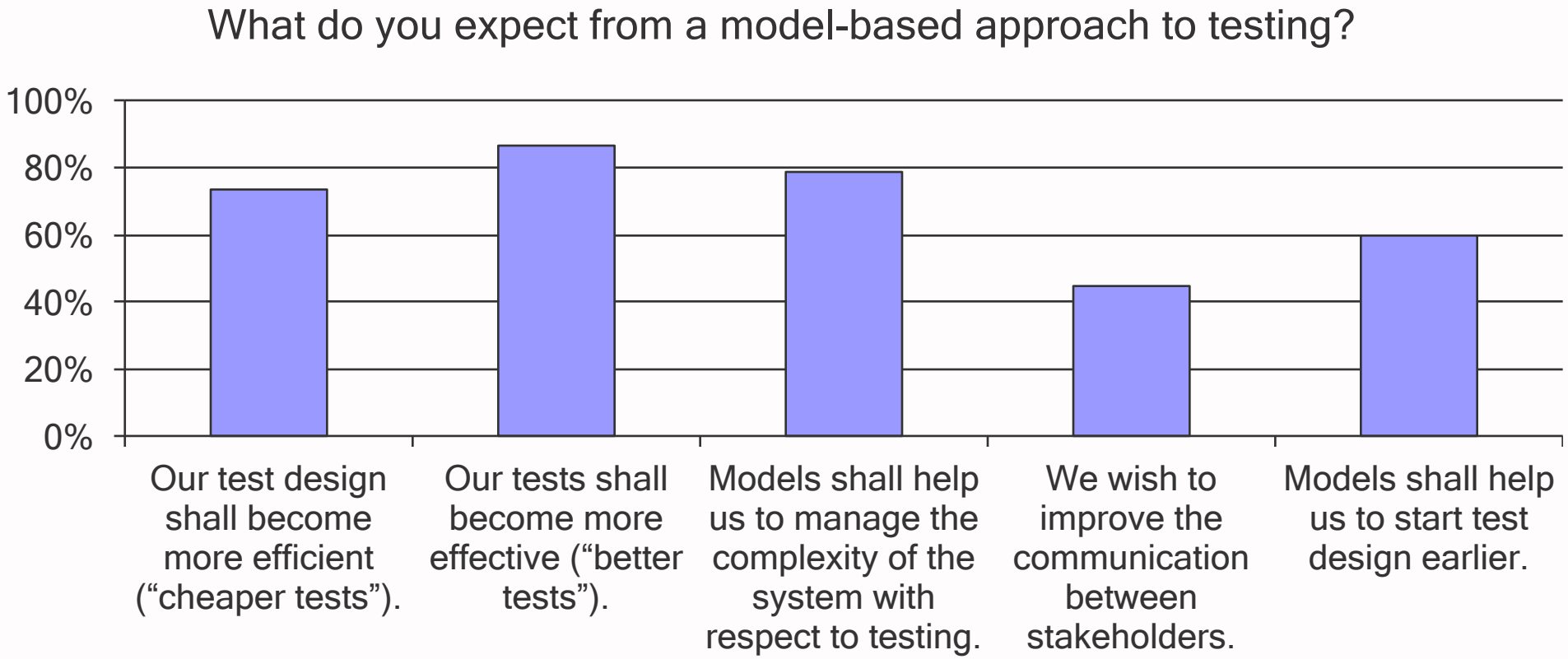
Manual execution

& scripts for automation

MBT Taxonomy Utting et al.'12



What are the benefices of MBT from industry point of view ?



Expectations of MBT practitioners (from 2014 MBT User Survey)

What are the MBT pitfalls and drawbacks?

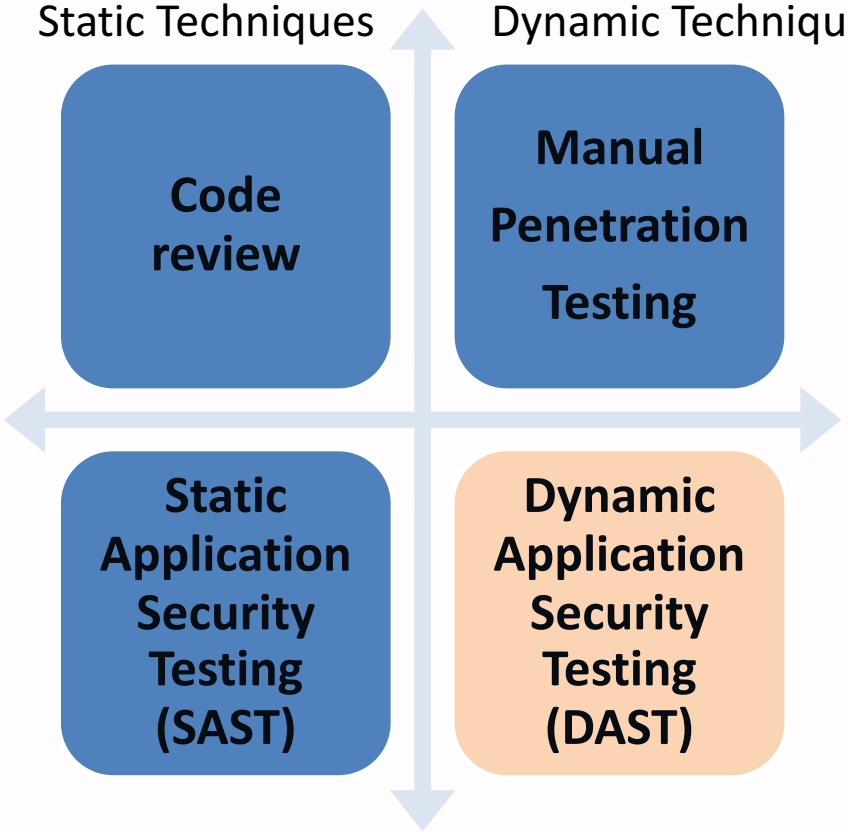
- MBT does not solves all problems
- MBT is not just a matter of tooling
- MBT models are not always correct
- MBT generates a myriad of test cases how to deal with this high number ?

Model-Based Security Testing

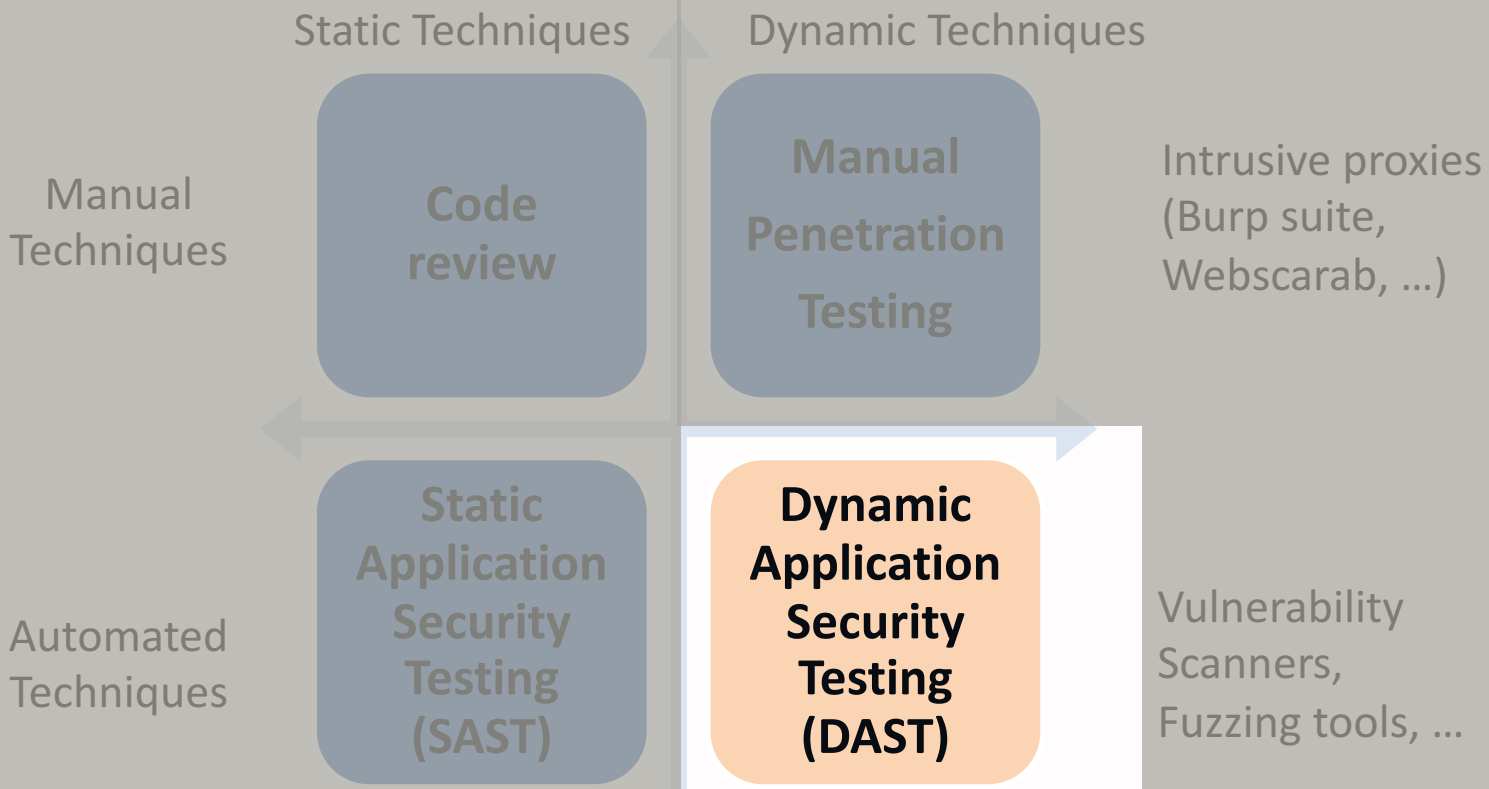
- What are the challenges in Security Testing ?

- Where MBT stands for Security Testing ?

Security Testing Approaches



Security Testing Approaches



DAST Approaches

Model-based Security Testing (MBST) Approaches

- Rely on a variety of techniques to compute black-box test cases
  - Patterns,
  - Fuzzing,
  - Model-checking, etc.

- Promising results for pattern-based techniques
  - Better detection rates than scanners
  - Less time consuming than manual penetration testing
  - Test execution integrated within large-scale testbeds

4th UCAAT User Conference on Advanced Automated Testing

ETSI World Class Standards

IN PRACTICE

Pattern-driven and Model-Based Security Testing

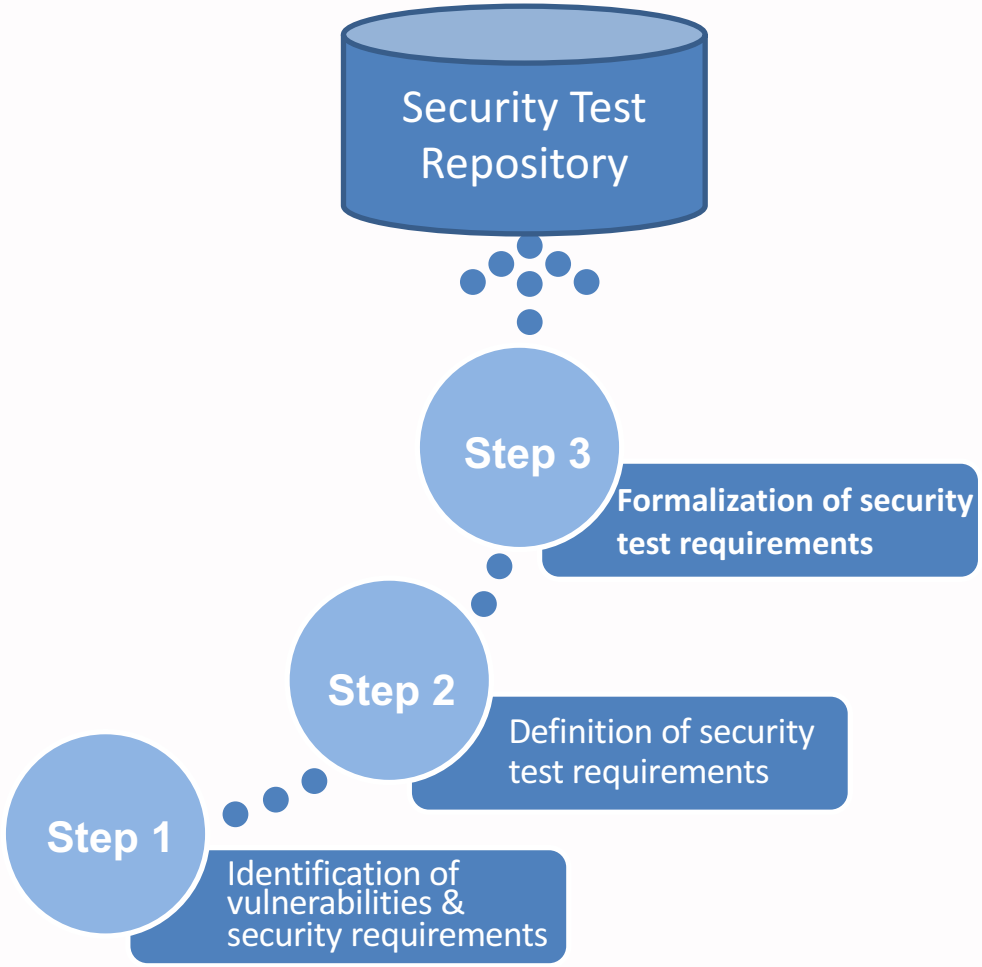
ARMOUR

smartesting Optimize your Test Center

Objectives in theory and in practice

- Objective 1 : Improve the coverage of security requirements, keeping overall traceability
- Objective 2 : Increase the fault detection capability of the test suite
- Objective 3: Cost-effectiveness

Pattern-based process to reach the objectives



MBT Process for security testing

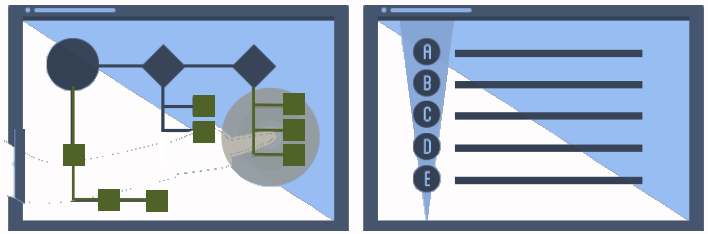
Security needs & requirements

Security Test Patterns

Risk Analysis

Security Test Purposes

Modeling for test generation



Automatic or manual test conception

Test Repository (Excel, TTCN-3, ALM…)

Functional tests

Manual execution

& scripts for automation

MBT Process for security testing

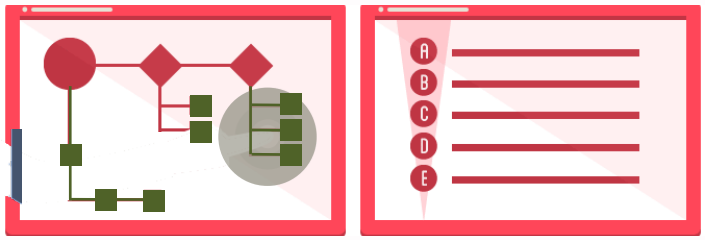
Security needs & requirements

Security Test Patterns

Risk Analysis

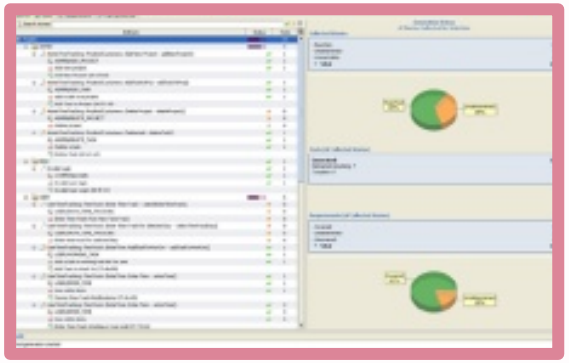
Security Test Purposes

Modeling for test generation



MBT Tool

Automatic test generation



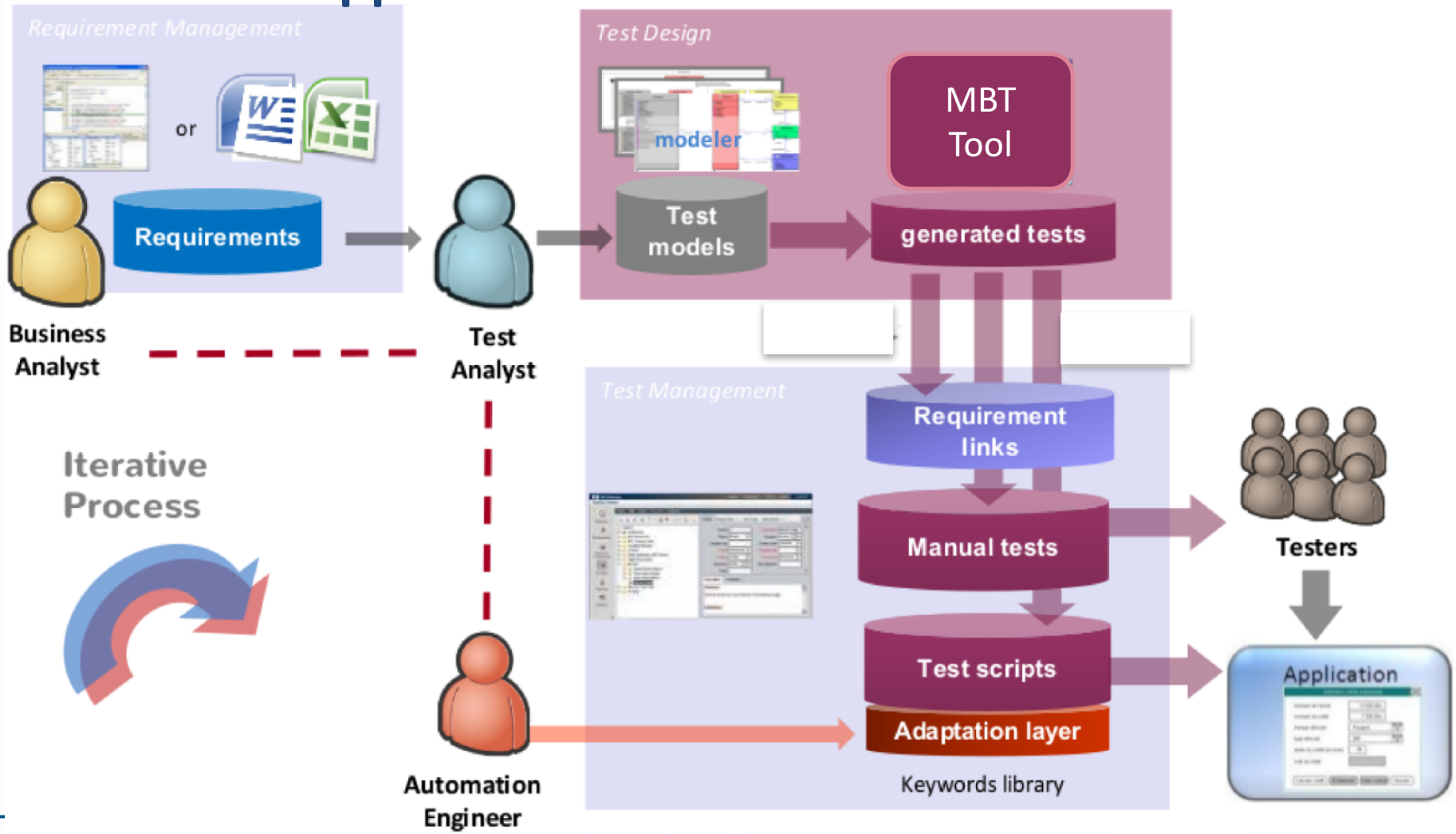
Test Repository (Excel, TTCN-3, ALM…)

Functional tests

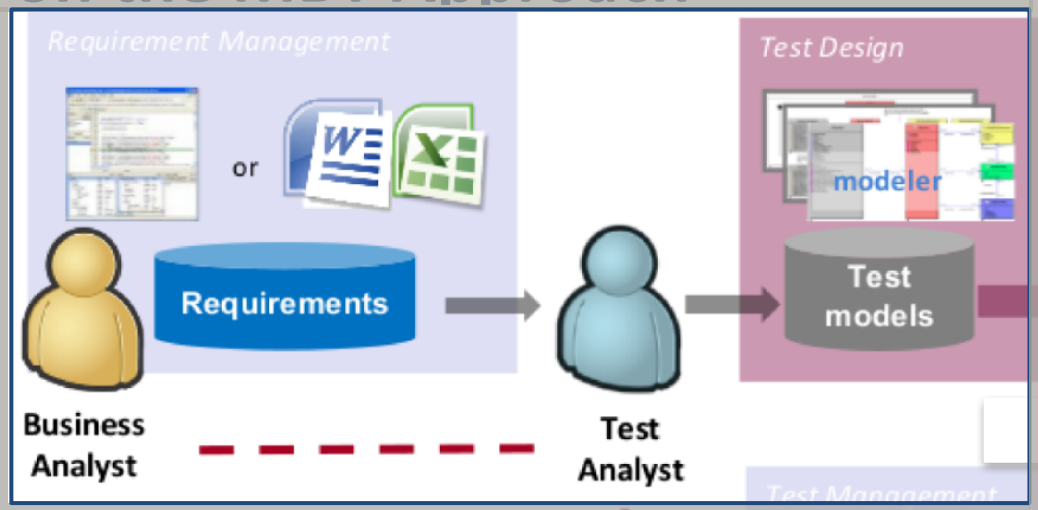
Manual execution

& scripts for automation

Background on the MBT Approach



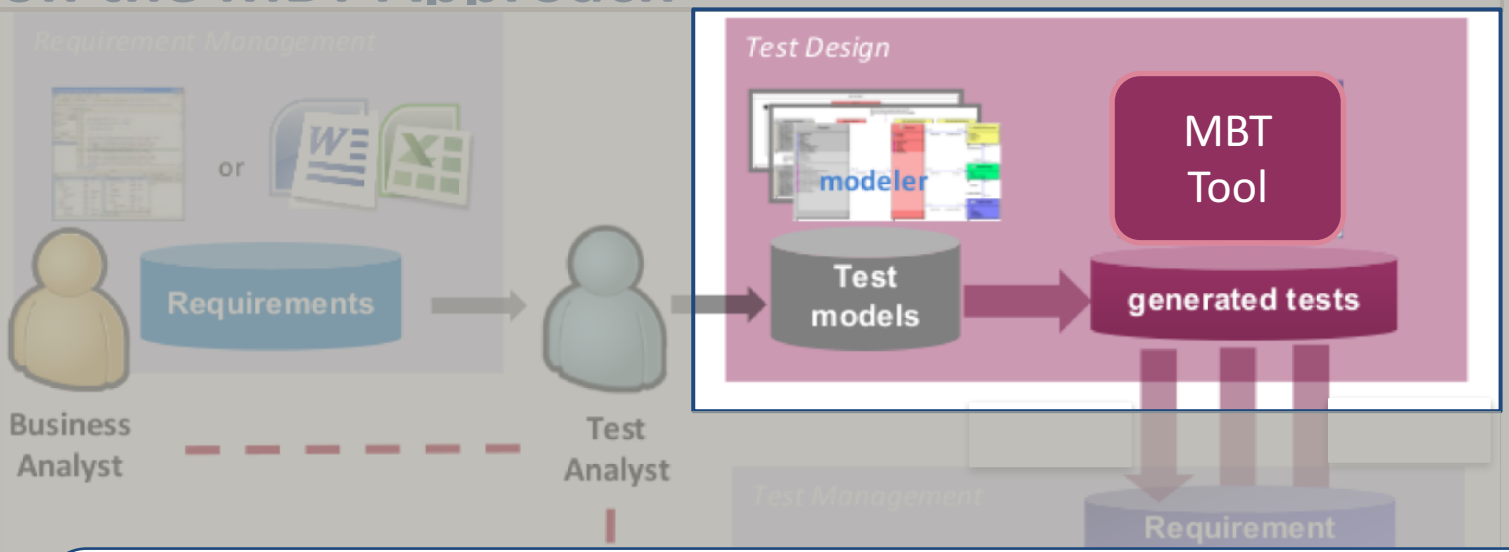
Background on the MBT Approach



Behavioral modeling notation (UML4MBT) based on UML metamodel:

- **Class diagrams** specify the static structure
- **Object diagrams** specify concrete entities and initial state
- **OCL** (Object Constraint Language) to describe its behavioral characteristics

Background on the MBT Approach



Test selection depending on two test characteristics:

- **Functional behavioral testing** → activate all behaviors
- **Security testing** → formalization of test scenarios using temporal properties (TOCL) and test patterns (TP)

Test generation relies on symbolic state exploration of the model:

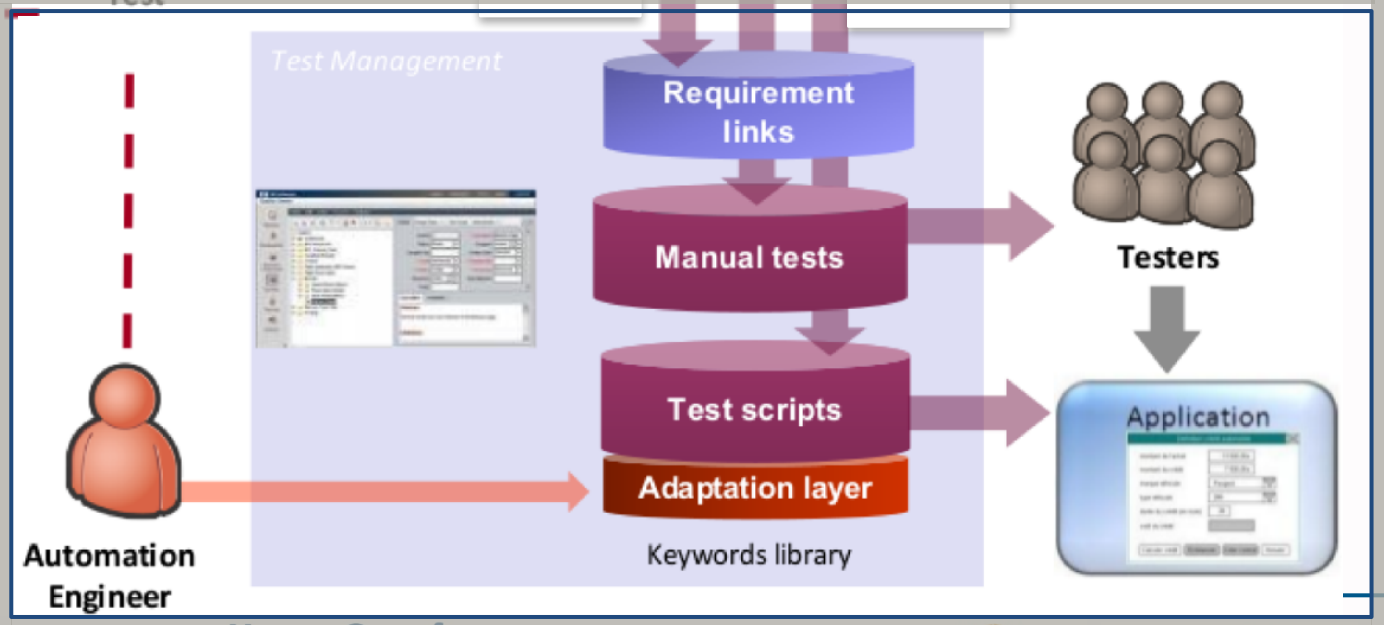
- **Functional behavioral testing** → A test target per behavior to activate
- **Security testing** → Test targets derived by unfolding each TOCL and TP

Test cases must be **concretized** to be made executable:

- Conformity table between abstract data and concrete data
- Implementation of class operations

Test cases may be published for test management and execution tools:

- HP Quality Center, TestLink, etc.



TOCL and TP test selection criteria

- **TOCL** and **TP** make possible to generate tests that exercise **corner cases,** relevant when testing security properties and vulnerabilities
- **TOCL** allows to express **temporal properties**, for instance of succession or precedence, contributing to the MBT process with:
  - Evaluation of the existing tests coverage
  - Verification of the model's conformance to these properties
    - Simplifying the model debugging
- **TP** allow to express in terms of **procedures of tests** based on a verbose representation and using the experts experience and knowledge on the system vulnerabilities

Design of Temporal Properties using TOCL

- TOCL = Temporal OCL
  - overlay of OCL to express temporal properties
  - based on Dwyer *et al.* property patterns [DAC99]
  - does not require the use of a complex formalism (e.g. LTL, CTL)

- TOCL Property = Pattern + Scope
  - Pattern: describes occurrences or orderings of events

    (always, never, eventually k times, precedes, follows)
  - Scope: describes the observation window on which the pattern is supposed to hold

    (globally, between, after, before)

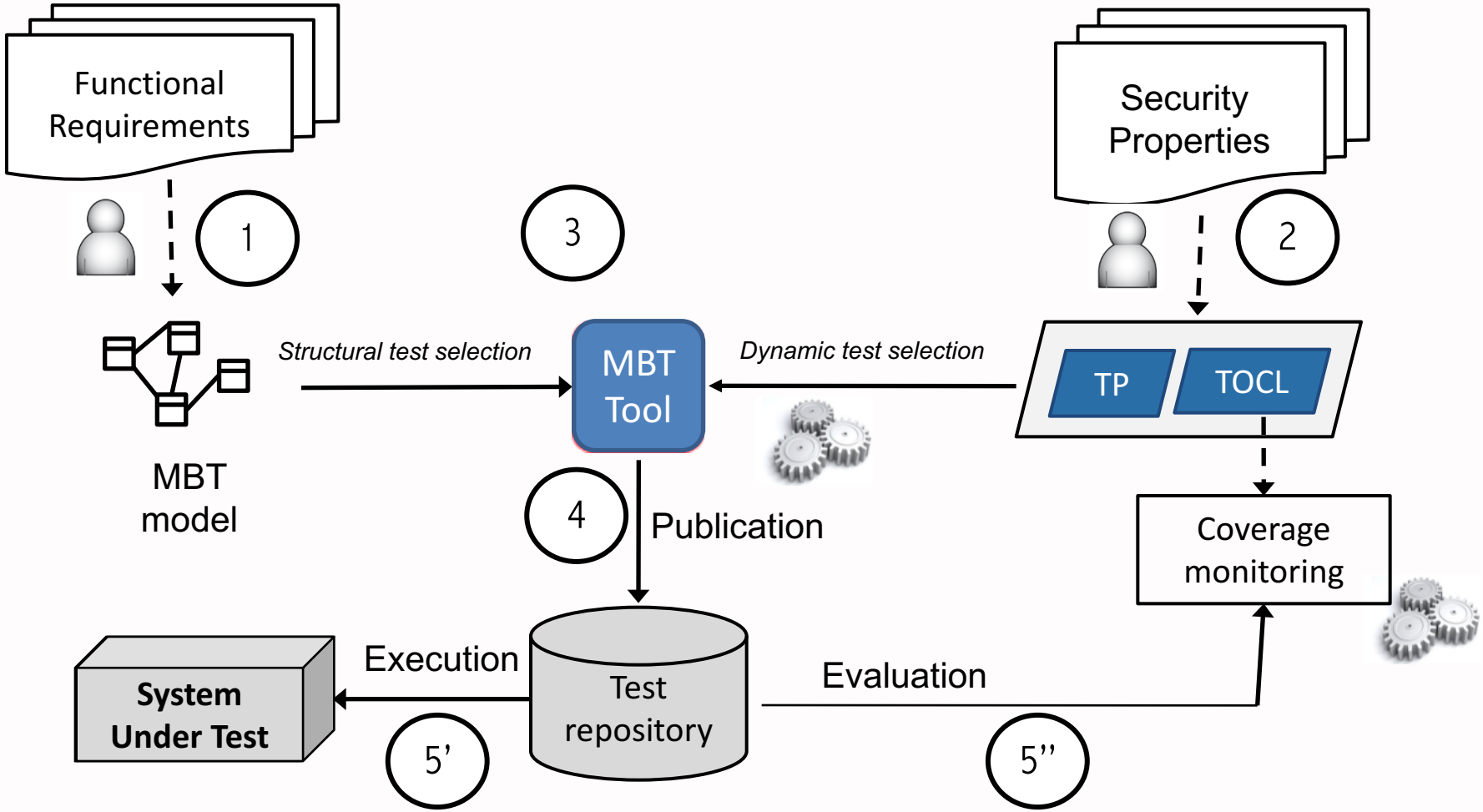
[DAC99] M. Dwyer, G. Avrunin, and J. Corbett. *Patterns in property specifications for finite-state verification*. ICSE'99.

Test Patterns

- Test Purpose Language
  - relies on the use of keywords to represent a test scenario expressing a combinations of test steps and test input parameters
  - powerful and easy to read by test engineers
  - does not require the use of a complex formalism (e.g. LTL, CTL)

- Test Purpose (TP) = Quantifiers + Blocks
  - Quantifiers: describes the context in which an action defined by the block will be activated
    - (**for_each** behavior $X **from** {list})
  - Blocks: describes the actions to be taken in order to activate a state in the model
    - (**use** any_operation any_number_of_times **to_active** $X)

MBT Process for Security Testing



4th UCAAT User Conference on Advanced Automated Testing

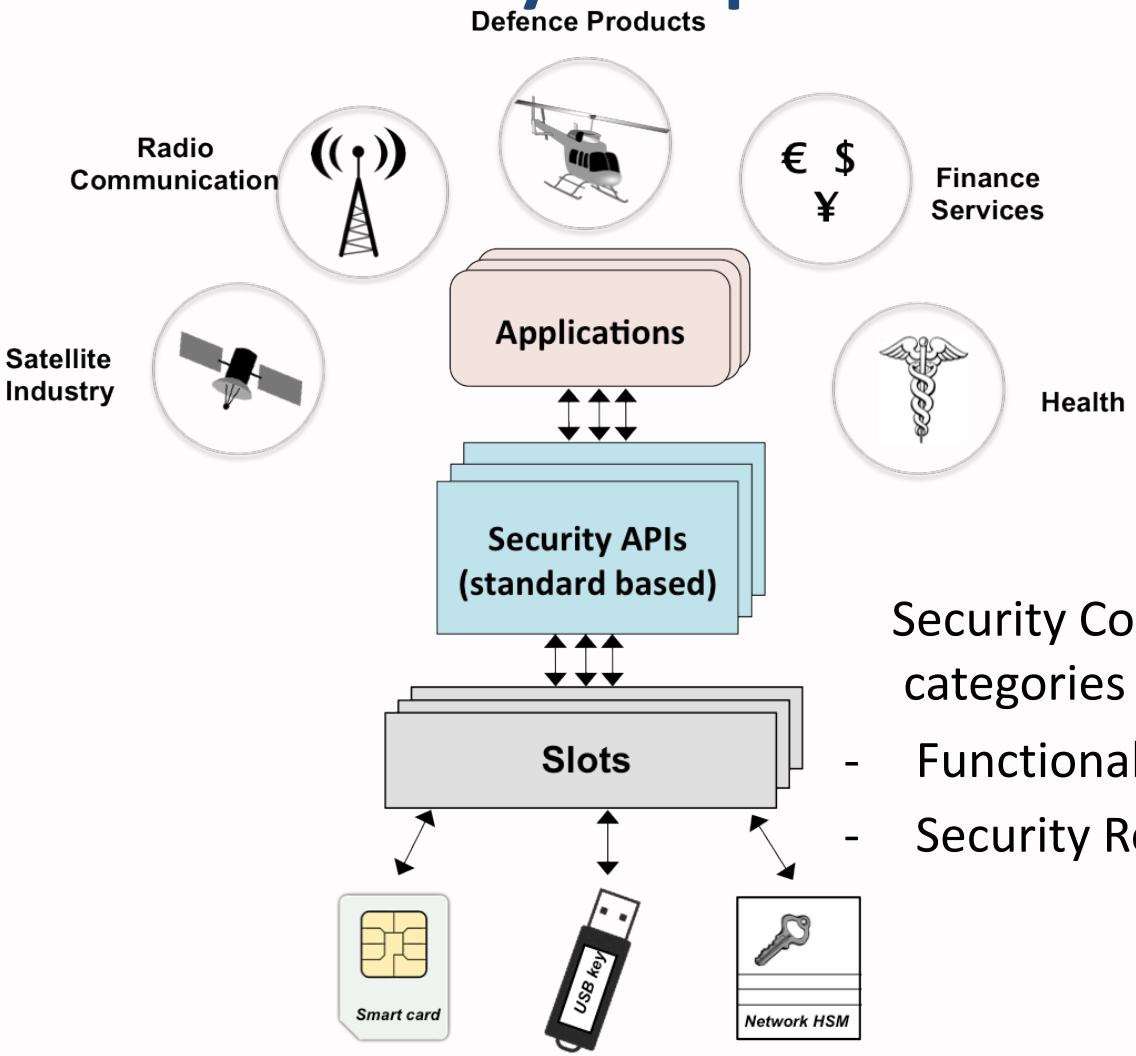
ETSI World Class Standards

EXPERIENCE IN SECURITY COMPONENTS TESTING

ARMOUR

smartesting Optimize your Test Center

Experience in security components testing



Security Components have two categories of test requirements:

- Functional Requirements
- Security Requirements

Experience in security components testing

- PKCS#11 is an RSA standard that defines an interface called Cryptoki to promote interoperability and security of cryptographic tokens.

- Scope: 24 functions most commonly present in the tokens, such as session, token, key and user management functions, as well as cryptographic functions for signing messages and verifying signatures.

- To ensure the repeatability of the MBT process we chose SoftHSM - virtual cryptographic store largely used for exploring PKCS#11 without the necessity to posses an HSM (created by the group OPENDNSSEC).

PKCS#11: Functional description

Specification documents :

PKCS #11 v2.20: Cryptographic Token Interface Standard

RSA Laboratories

28 June 2004

Table of Contents

♦ **C_OpenSession**

```
CK_DEFINE_FUNCTION(CK_RV, C_OpenSession)(
  CK_SLOT_ID slotID,
  CK_FLAGS flags,
  CK_VOID_PTR pApplication,
  CK_NOTIFY Notify,
  CK_SESSION_HANDLE_PTR phSession
);
```

C_OpenSession opens a session between an application and a token in a particular slot. *slotID* is the slot's ID; *flags* indicates the type of session; *pApplication* is an application-

June 2004

defined pointer to be passed to the notification callback; *Notify* is the address of the notification callback function (see Section 11.17); *phSession* points to the location that receives the handle for the new session.

When opening a session with **C_OpenSession**, the *flags* parameter consists of the logical OR of zero or more bit flags defined in the **CK_SESSION_INFO** data type. For legacy reasons, the **CKF_SERIAL_SESSION** bit must always be set; if a call to **C_OpenSession** does not have this bit set, the call should return unsuccessfully with the error code CKR_PARALLEL_NOT_SUPPORTED.

PKCS#11: Functional requirements

	A	B	C	F	J	K	L
1	@REQ	Requirement description	Spec v2.20	Main function			
	C_DigestFinal	C_DigestFinal finishes a multiple-part message-digesting operation, returning the message digest.	Section 11.10 p.151	C_DigestFinal	Effet : OPERATION_NOT_INITIALIZED	Effet : BUFFER_TOO_SMALL	Effet : OK
17	C_SignInit	C_SignInit initializes a signature operation	Section 11.11 p.152	C_SignInit	Effet : USER_NOT_LOGGED_IN	Effet : OPERATION_ACTIVE	Effet : MECHANISM_INVALID - the mechanism is not valid
18	C_Sign	C_Sign signs data in a single part	Section 11.11 p.153	C_Sign	Effet : OPERATION_NOT_INITIALIZED	Effet : BUFFER_TOO_SMALL	Effet : USER_NOT_LOGGED_IN
19	C_SignUpdate	C_SignUpdate continues a multiple-part signature operation	Section 11.11 p.154	C_SignUpdate	Effet : OPERATION_NOT_INITIALIZED	Effet : USER_NOT_LOGGED_IN	Effet : ARGUMENTS_BAD - the data to sign is null
	C_SignFinal	C_SignFinal finishes a multiple-part signature operation	Section 11.11 p.154-155	C_SignFinal	Effet : OPERATION_NOT_INITIALIZED	Effet : BUFFER_TOO_SMALL	Effet : USER_NOT_LOGGED_IN

PKCS#11: Test model

The **test model** is a System Under Test **abstraction**, representing its **expected behavior**.

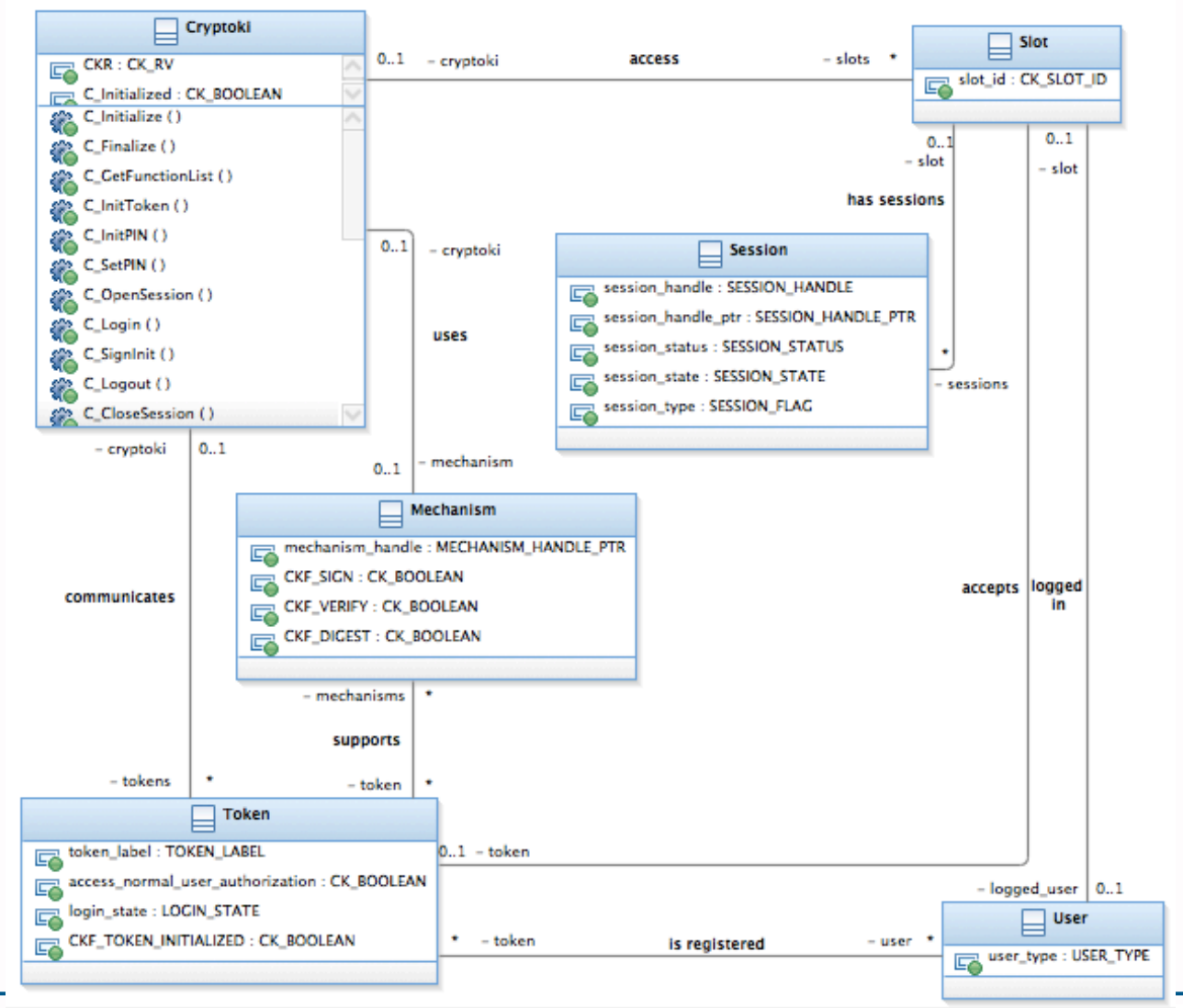
Class Diagram: → *represents the business objects that can be used by the System Under Test* → *classes own operations that can be called on the system under test (control and observation points)*
OCL: → *represents the expected behavior of an operation on the System Under Test, regarding the system state and the operation parameters*
Instance Diagram: → *represents the initial state of the System Under Test*

PKCS#11: Test model

The **test model** is a System Under Test **abstraction**, representing its **expected behavior**.

Class Diagram: → *represents the business objects that can be used by the System Under Test* → *classes own operations that can be called on the system under test (control and observation points)*
OCL: → *represents the expected behavior of an operation on the System Under Test, regarding the system state and the operation parameters*
Instance Diagram: → *represents the initial state of the System Under Test*

PKCS#11: Class Diagram



PKCS#11: Test model

The **test model** is a System Under Test **abstraction**, representing its **expected behavior**.

Class Diagram:

→ *represents the business objects that can be used by the System Under Test*

→ *classes own operations that can be called on the system under test (control and observation points)*

OCL:

→ *represents the expected behavior of an operation on the System Under Test, regarding the system state and the operation parameters*

Instance Diagram:

→ *represents the initial state of the System Under Test*

	A	B	C	F	J
1	@REQ	Requirement description	Spec v2.20	Main function	
16	C_DigestFinal	C_DigestFinal finishes a multiple-part message-digesting operation, returning the message digest.	Section 11.10 p.151	C_DigestFinal	Effet : OPERATION_NOT_INITIALIZED
17	C_SignInit	C_SignInit initializes a signature operation	Section 11.11 p.152	C_SignInit	Effet : USER_NOT_LOGGED_IN
	C_Sign	C_Sign signs data in a single part	Section 11.11 p.153	C_Sign	Effet : OPERATION_NOT_INITIALIZED

```
if((self.C_SignInitialized=CK_BOOLEAN::CK_FALSE) and (self.mechanism.oclIsUndefined()))=false then
    self.CKR = CK_RV::CKR_OPERATION_ACTIVE
    /**@AIM:OPERATION_ACTIVE*/
else
    if((session.slot.logged_user.oclIsUndefined() then
        self.CKR = CK_RV::CKR_USER_NOT_LOGGED_IN
        /**@AIM:USER_NOT_LOGGED_IN*/
    else
        let mech : Mechanism = mechs->any(true) in
        self.C_SignInitialized = CK_BOOLEAN::CK_TRUE and
        mech.key = key and
        self.mechanism = mech and -- creating the link "is currently used"
        self.CKR = CK_RV::CKR_OK
        /**@CKR:OK*/
        /**@AIM:OK*/
    endif
endif
```

PKCS#11: Test model

The **test model** is a System Under Test **abstraction**, representing its **expected behavior**.

Class Diagram:

→ *represents the business objects that can be used by the System Under Test*

→ *classes own operations that can be called on the system under test (control and observation points)*

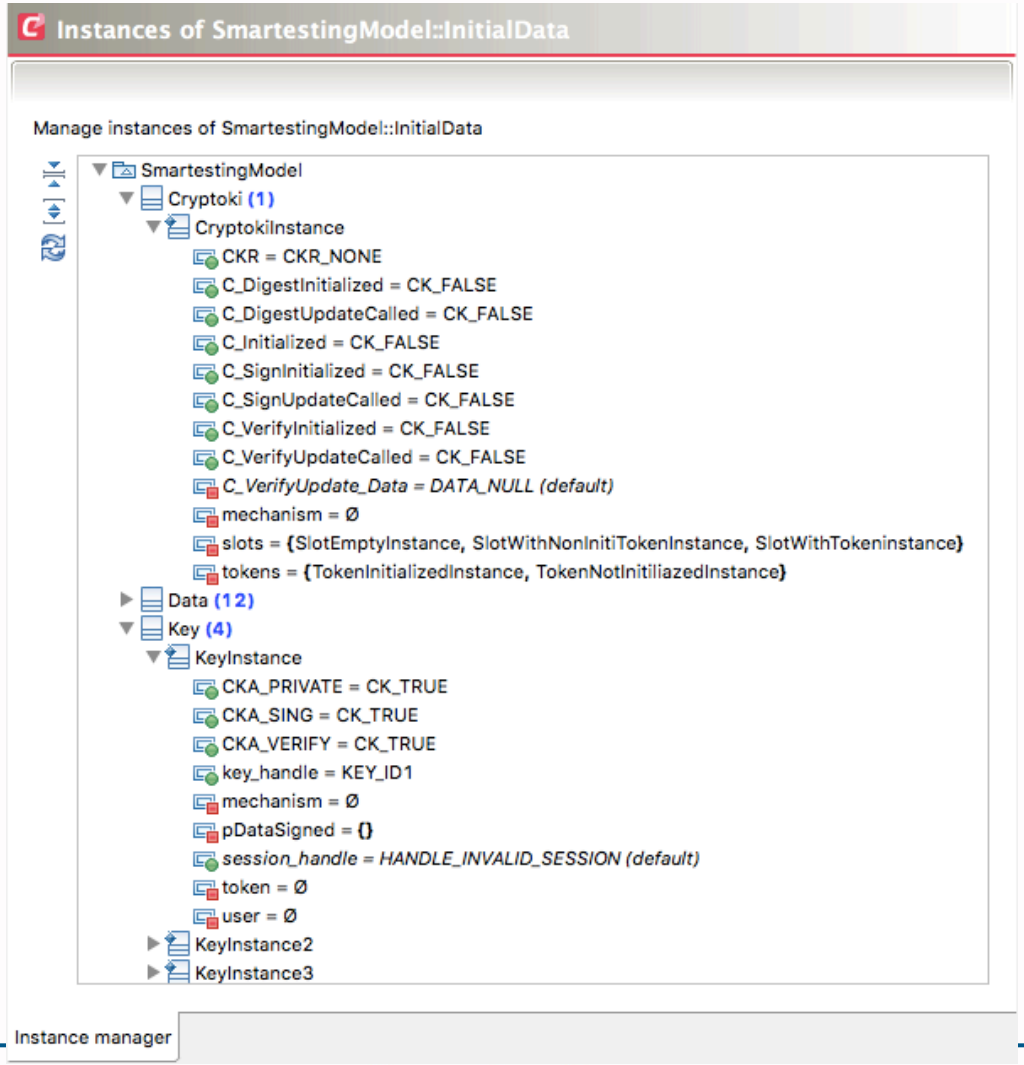
OCL:

→ *represents the expected behavior of an operation on the System Under Test, regarding the system state and the operation parameters*

Instance Diagram:

→ *represents the initial state of the System Under Test*

PKCS#11: Initial State



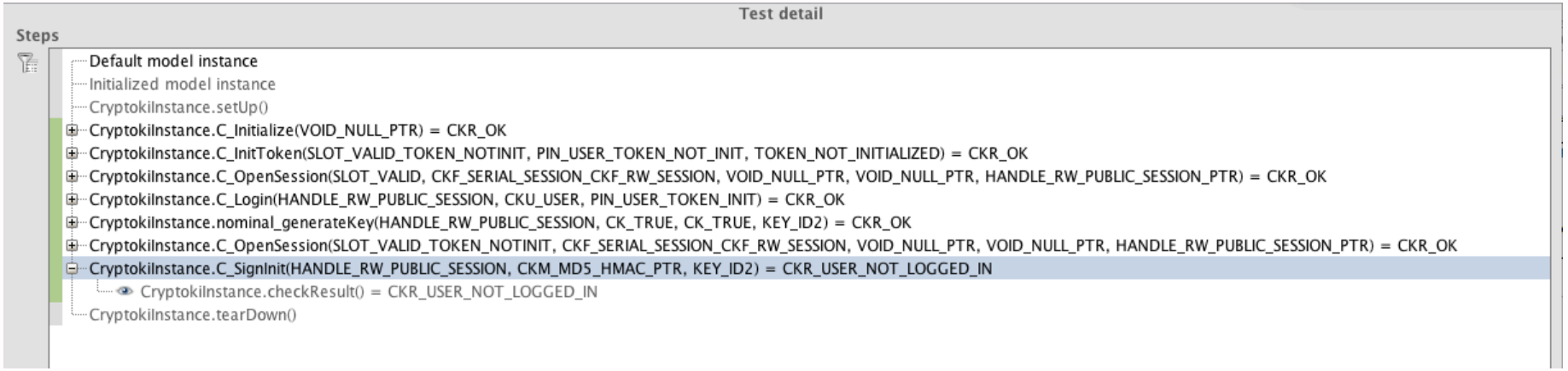
Exercise Functional vs Security Functional Requirements

- Functional Requirement
  - Cryptoki signs data if the user logged, otherwise it responds with an error code USER_NOT_LOGGED_IN
- Security Functional Requirement
  - Wh tion's
    har s later
    log . In
    addition, all private session objects from sessions belonging to the application are destroyed.

How will you test these requirements ?
How you will express them, using TP or TOCL ?

Exercise Solution (1/3)

- Functional Requirement
  - The tool creates a test case by choosing the shortest path



Exercise Solution (2/3)

- Security Functional Requirement

```
for_each literal $KEY from KEY_ID1 or KEY_ID2 or KEY_ID4 or KEY_ID5,
use any_operation any_number_of_times then
use cryptoki.C_OpenSession(_)
        to_activate behavior_with_tags  {CKR:OK} then
use cryptoki.C_Login(_)
        to_activate behavior_with_tags {AIM:C_Login/CKU_USER_RW} then
use cryptoki.nominal_generateKey(_,_,CK_TRUE,_)
        to_activate behavior_with_tags {AIM:GENERATE_KEY/OK} then
use cryptoki.C_Finalize(_) then
use any_operation any_number_of_times then
use cryptoki.C_Login(_)
        to_activate behavior_with_tags {AIM:C_Login/CKU_USER_RW} then
use cryptoki.C_SignInit(_,_,$KEY)
```

Exercise Solution (3/3)

- Security Functional Requirement

Test detail

Steps

- Default model instance
- Initialized model instance
- CryptokiInstance.setUp()
- CryptokiInstance.C_Initialize(VOID_NULL_PTR) = CKR_OK
- CryptokiInstance.C_OpenSession(SLOT_VALID, CKF_SERIAL_SESSION_CKF_RW_SESSION, VOID_NULL_PTR, VOID_NULL_PTR, HANDLE_RW_PUBLIC_SESSION_PTR) = CKR_OK
- CryptokiInstance.C_Login(HANDLE_RW_PUBLIC_SESSION, CKU_USER, PIN_USER_TOKEN_INIT) = CKR_OK
- CryptokiInstance.nominal_generateKey(HANDLE_RW_PUBLIC_SESSION, CK_TRUE, CK_TRUE, KEY_ID1) = CKR_OK
- CryptokiInstance.C_Finalize(VOID_NONNULL_PTR) = CKR_ARGUMENTS_BAD
- CryptokiInstance.C_CloseAllSessions(SLOT_VALID) = CKR_OK
- CryptokiInstance.C_OpenSession(SLOT_VALID, CKF_SERIAL_SESSION_CKF_RW_SESSION, VOID_NULL_PTR, VOID_NULL_PTR, HANDLE_RW_PUBLIC_SESSION_PTR) = CKR_OK
- CryptokiInstance.C_Login(HANDLE_RW_PUBLIC_SESSION, CKU_USER, PIN_USER_TOKEN_INIT) = CKR_OK
- CryptokiInstance.C_SignInit(HANDLE_RW_PUBLIC_SESSION, CKM_SHA512_PTR, KEY_ID1) = CKR_OBJECT_HANDLE_INVALID
- CryptokiInstance.tearDown()

PKCS#11 in numbers

PKCS#11 set up metrics

Test Requirement category	#FR	#SFR
general purpose	7	4
slot and token management	22	5
session management	32	9
object management	6	1
digesting	28	9
signing	32	10
verifying signatures	31	10
total	158	48

PKCS#11 model element	
#classes	9
#enumerations	20
#enum. literals	123
#associations	17
#class attributes	34
#operations	24
#observations	1
#behaviors	206
#tocl properties	50
#test purposes	5
#LOC	1308

LOC: Lines of OCL constraints

PKCS#11 in numbers

PKCS#11 test generation coverage and execution metrics

Test Selection Criterion	#Test targets	#Test cases	Cov. in % FR	Cov. in % SFR
Structural	206	184	100	40
TOCL	311	90	31	58
Test Purpose	24	24	9	2
Manual	24	24	45	/

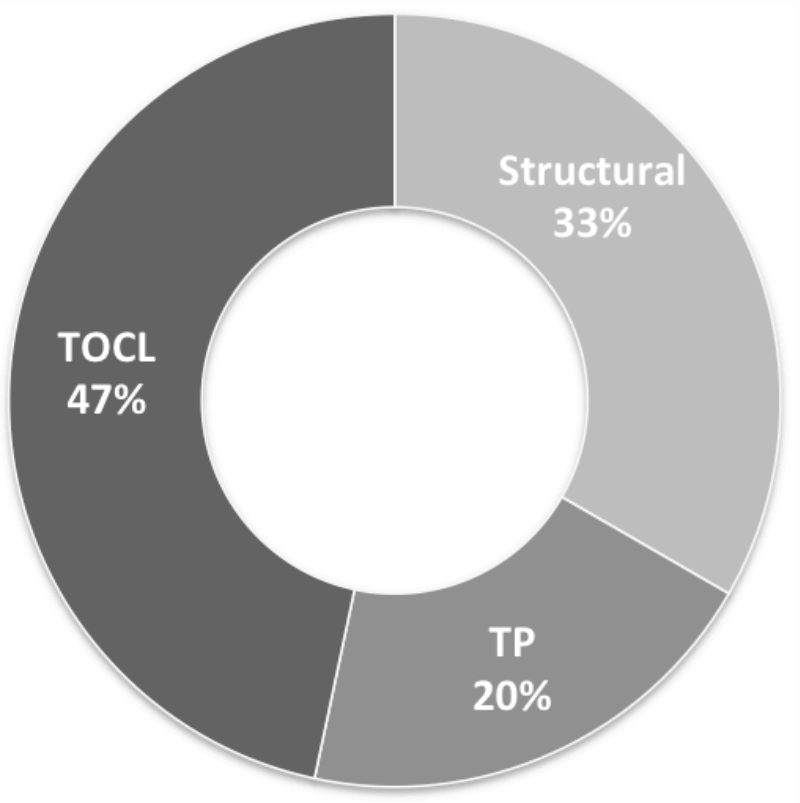
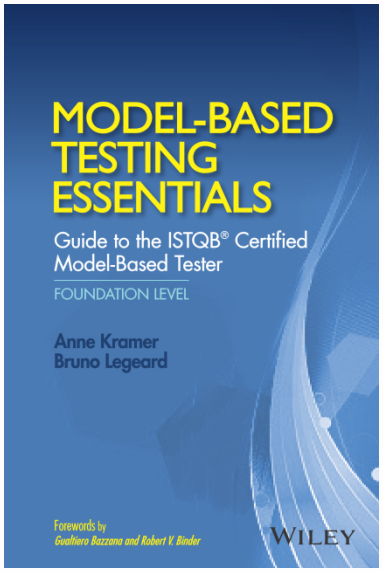


Fig. Distinct fault detection capabilities per coverage requirement

To learn more about this case study



Chapter 11 – PKCS #11 case study

Published in 2016 – Related to the ISTQB® Model-Based Tester Certification

4th UCAAT User Conference on Advanced Automated Testing

ETSI World Class Standards

EXPERIENCE IN SECURITY TESTING FOR IOT SYSTEMS

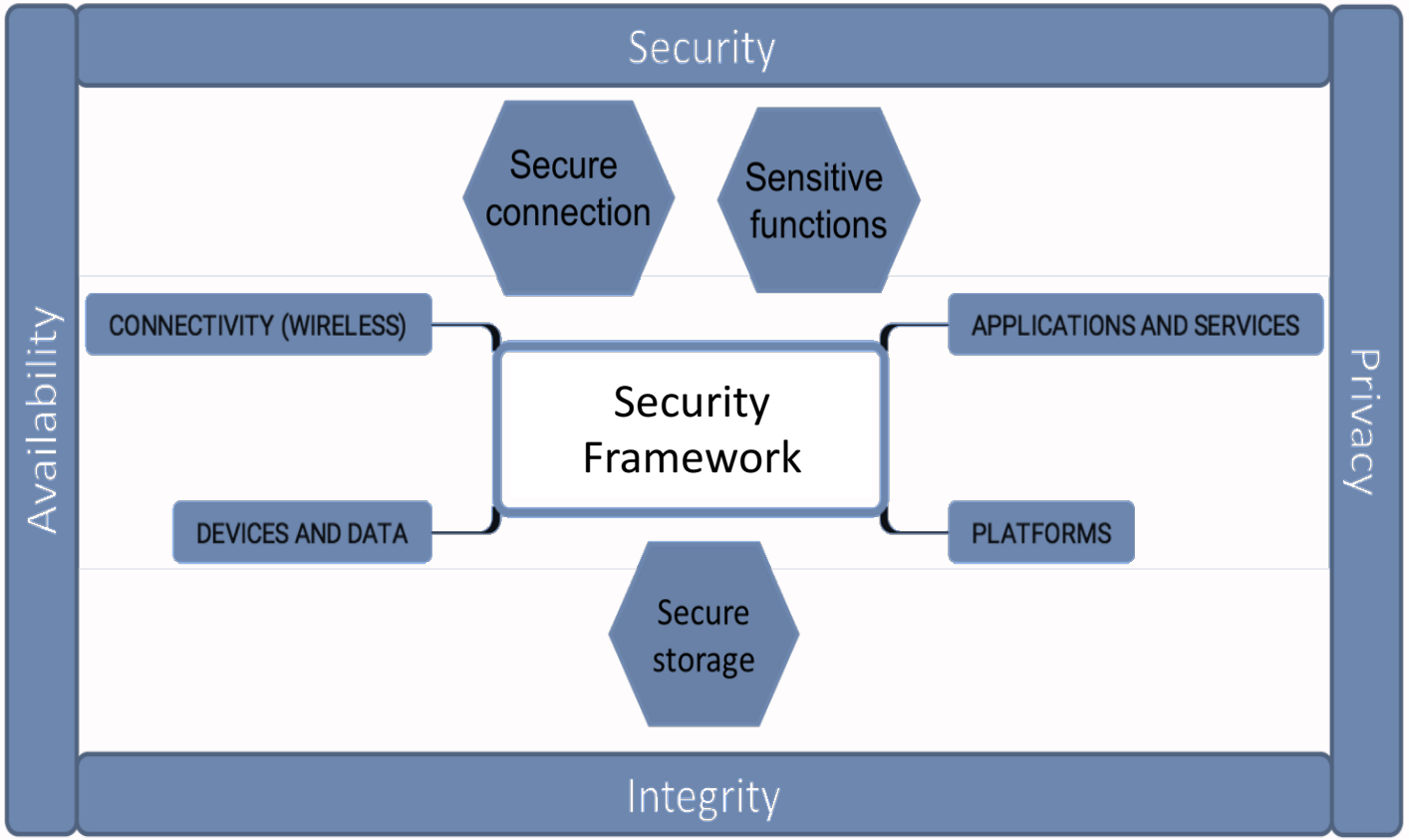
ARMOUR

smartesting Optimize your Test Center

IoT Existing Security Frameworks

- **OWASP IoT** defines a framework that gathers information on security issues associated to the IoT development, deployment or technology assessment. However, it remains **too high level and lacks a specific methodology** that could be used in a systemic way - for instance in security audits.
- **GSMA** provides a set of security guideline documents that target all IoT involved entities (service providers, device manufacturers, developers, network operators etc.). However, the GSMA Security Framework only points to **currently available solutions, standards and best practices**.
- **oneM2M** identifies 4 security domains (*Application*, *Intra Common Services*, *Inter Common Services*, *Underlying Network*), and 3 layers (*Security Functions, Security Environment Abstraction, Secure Environment*). Selected as **a starting point for ARMOUR risk analysis and mitigation methodology**.

IoT Security Framework [H2020 ARMOUR Project]



IoT Security Framework [H2020 ARMOUR Project]

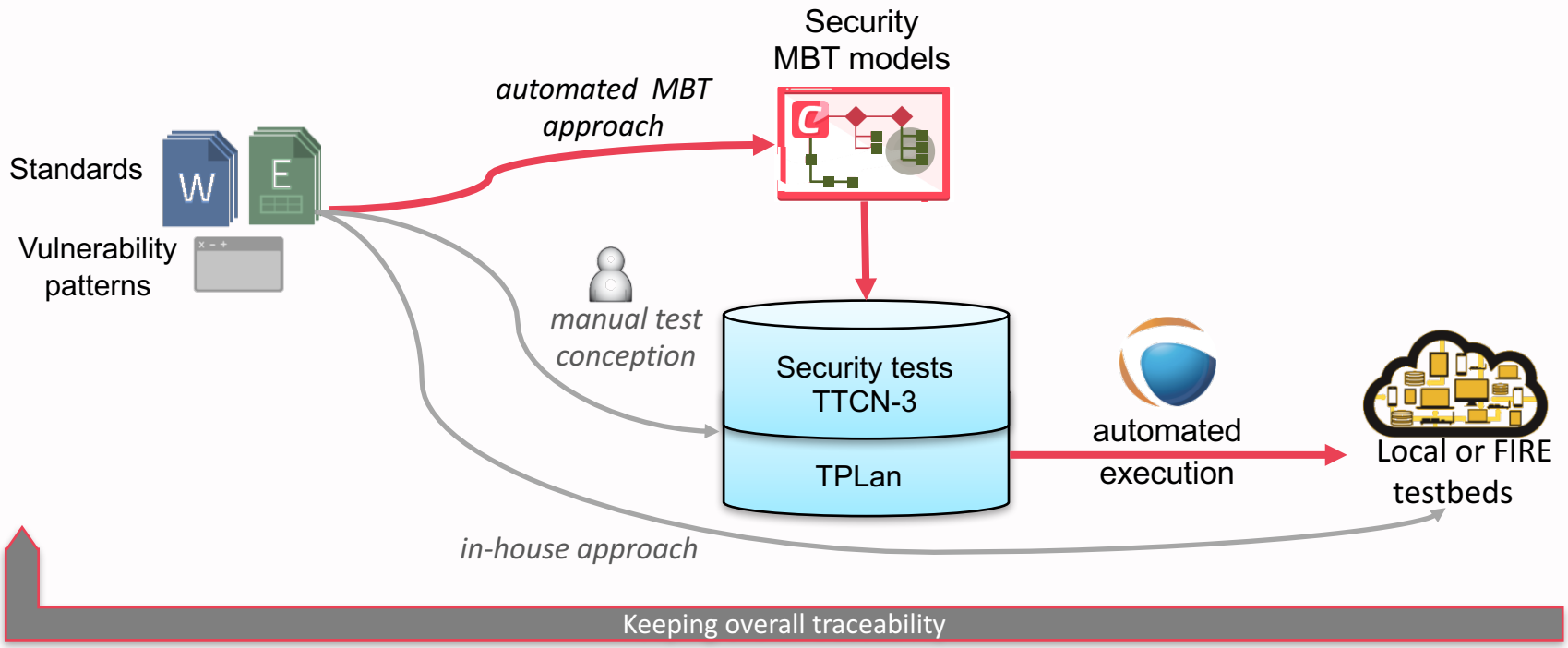
The ARMOUR security framework takes as entry the oneM2M vulnerabilities, threats and risk assessment methodology, and for each experiment:

- performed an analysis of the **risks/vulnerabilities** to be considered during all phases of the experimentation.
- defined a **set of countermeasures** in order to address the vulnerabilities and reduce associated risk.
- described the **scenarios and methodology** that will be the basis for the experiments execution.

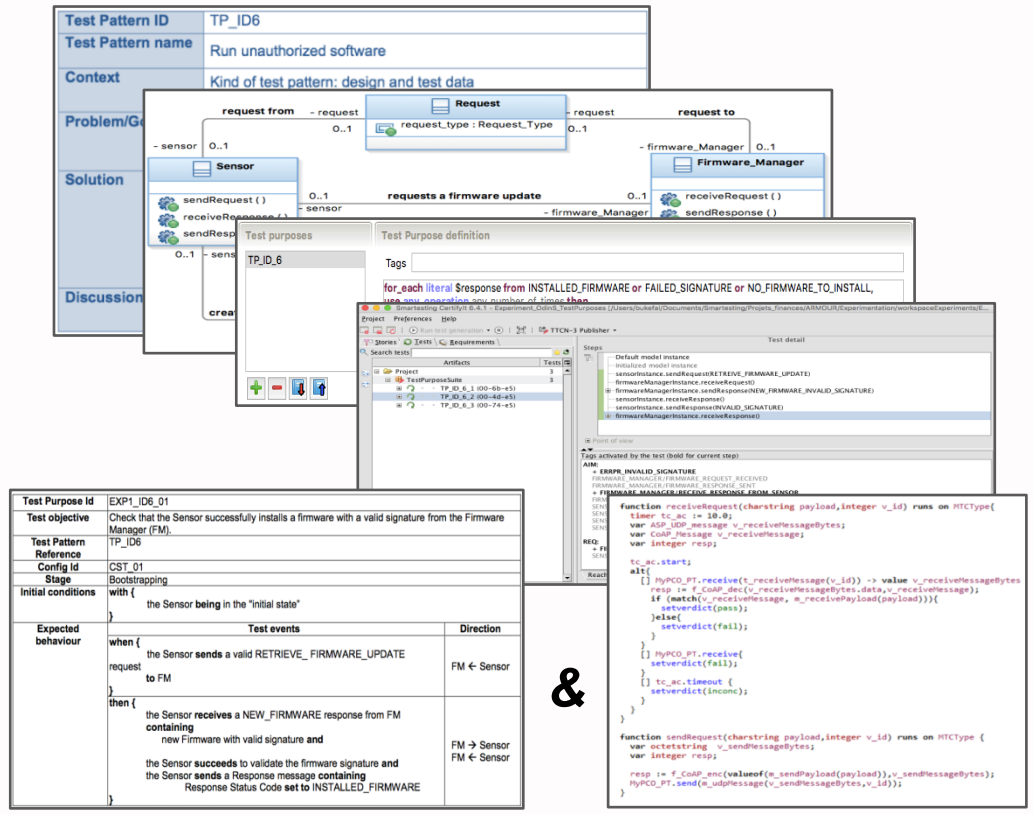
Vulnerability classification and patterns in IoT systems

- Built on existing frameworks and adapt them if necessary,
- A vulnerability pattern intends to describe vulnerabilities, their conditions of occurrence and impacts.
- CVE (Common Vulnerabilities and Exposure) framework: dictionary of publicly known information security vulnerabilities and exposures

MBT Methodology and Framework for Large-Scale IoT Testing



MBT Methodology in 5 steps applied to **oneM2M**



Test Purpose Id	EXP1_ID6_01
Test objective	Check that the Sensor successfully installs a firmware with a valid signature from the Firmware Manager (FM).
Test Pattern Reference	TP_ID6
Config Id	CST_01
Stage	Bootstrapping
Initial conditions	with { the Sensor **being** in the "initial state" }
Expected behaviour	**Test events** / **Direction**
	when { the Sensor **sends** a valid RETRIEVE_ FIRMWARE_UPDATE request **to** FM } — FM ← Sensor
	then { the Sensor **receives** a NEW_FIRMWARE response from FM **containing** new Firmware with valid signature **and** the Sensor **succeeds** to validate the firmware signature **and** the Sensor **sends** a Response message **containing** Response Status Code **set to** INSTALLED_FIRMWARE } — FM → Sensor, FM ← Sensor

&

```
function receiveRequest(charstring payload,integer v_id) runs on MTCType{
  timer tc_ac := 10.0;
  var ASP_UDP_message v_receiveMessageBytes;
  var CoAP_Message v_receiveMessage;
  var integer resp;

  tc_ac.start;
  alt{
    [] MyPCO_PT.receive(t_receiveMessage(v_id)) -> value v_receiveMessageBytes
       resp := f_CoAP_dec(v_receiveMessageBytes.data,v_receiveMessage);
       if (match(v_receiveMessage, m_receivePayload(payload))){
         setverdict(pass);
       }else{
         setverdict(fail);
       }
    [] MyPCO_PT.receive{
       setverdict(fail);
       }
    [] tc_ac.timeout {
       setverdict(inconc);
       }
  }
}

function sendRequest(charstring payload,integer v_id) runs on MTCType {
  var octetstring  v_sendMessageBytes;
  var integer resp;

  resp := f_CoAP_enc(valueof(m_sendPayload(payload)),v_sendMessageBytes);
  MyPCO_PT.send(m_udpMessage(v_sendMessageBytes,v_id));
}
```

1. ***Vulnerability patterns identification***
2. ***MBT model*** *based on ARMOUR guidelines*
3. ***Security test pattern formalization*** *using the Smartesting Test Purpose Language*
4. ***MBT test generation*** *based on ARMOUR test strategies using CertifyIt*
5. ***Publication*** *in* ***TPLan*** *– test description and* ***TTCN-3*** *test scripts*

testbed execution and results

Results from the oneM2M experience

1. Security test pattern formalization for test generation
2. Definition of generic MBT models for TTCN-3 and TPLan production
3. TTCN-3 publisher
4. TPLan publisher
5. Detection of inconsistencies in the IoT platform under test with the specification during the oneM2M interoperability event in South Korea.

CONCLUSION

Lessons learnt from experience

LESSONS LEARNT

Based on MBT pitfalls and drawbacks

- MBT is not just a matter of tooling
  - careful evaluation of the organisation is necessary for a successful adoption
  - efficient & effective test strategy on long term scale
- MBT models are not always correct
  - adequate tools are necessary to measure the quality of the models, as it leads to quality of test cases
- MBT generates a myriad of test cases
  - to deal with it adequate formalism are necessary to benefit from domain experts experience

LESSONS LEARNT

Benefits from the MBT approach based on our experience

- it produced **cheaper** tests (average time spent of modelling - 2 days)
- it produces **better** tests (increased fault detection, better model quality)
- MBT models are **clear**
- **early** test design (detection of inconsistencies in specification)

CONCLUSION AND FUTURE WORK

- Model-Based Security Testing position at the research and industry state of the art
- MBT tooling extended to security testing
- Initial MBT Framework for Security Testing of IoT systems at different levels
- OneM2M experience → *A. Ahmad presentation on Friday*
- **Security is number one challenge in the IoT domain → Looking forward for new proof of concepts**

ISTQB® Certified Tester Foundation Level Model Based Testing

THANK YOU

QUESTIONS ?