2nd UCAAT User Conference on Advanced Automated Testing

September 16-18 2014, Munich, Germany

ETSI World Class Standards

Model-Based Security Testing with Test Patterns

Julien BOTELLA (Smartesting)
Jürgen GROSSMANN (FOKUS)
Bruno LEGEARD (Smartesting)
Fabien PEUREUX (Smartesting)
Martin SCHNEIDER (FOKUS)
Fredrik SEEHUSEN (SINTEF)

http://www.rasenproject.eu/

RASEN Compositional Risk Assessment and Security Testing of Networked Systems

Fraunhofer FOKUS SINTEF smartesting Optimize your Test Center

SEVENTH FRAMEWORK PROGRAMME

Agenda

- Context, motivation and objectives
- Approach for Risk-Based Security Testing
- Illustration of the end-to-end process
- Conclusion and future work

Context

- *FP7 RASEN project (2012-2015)*

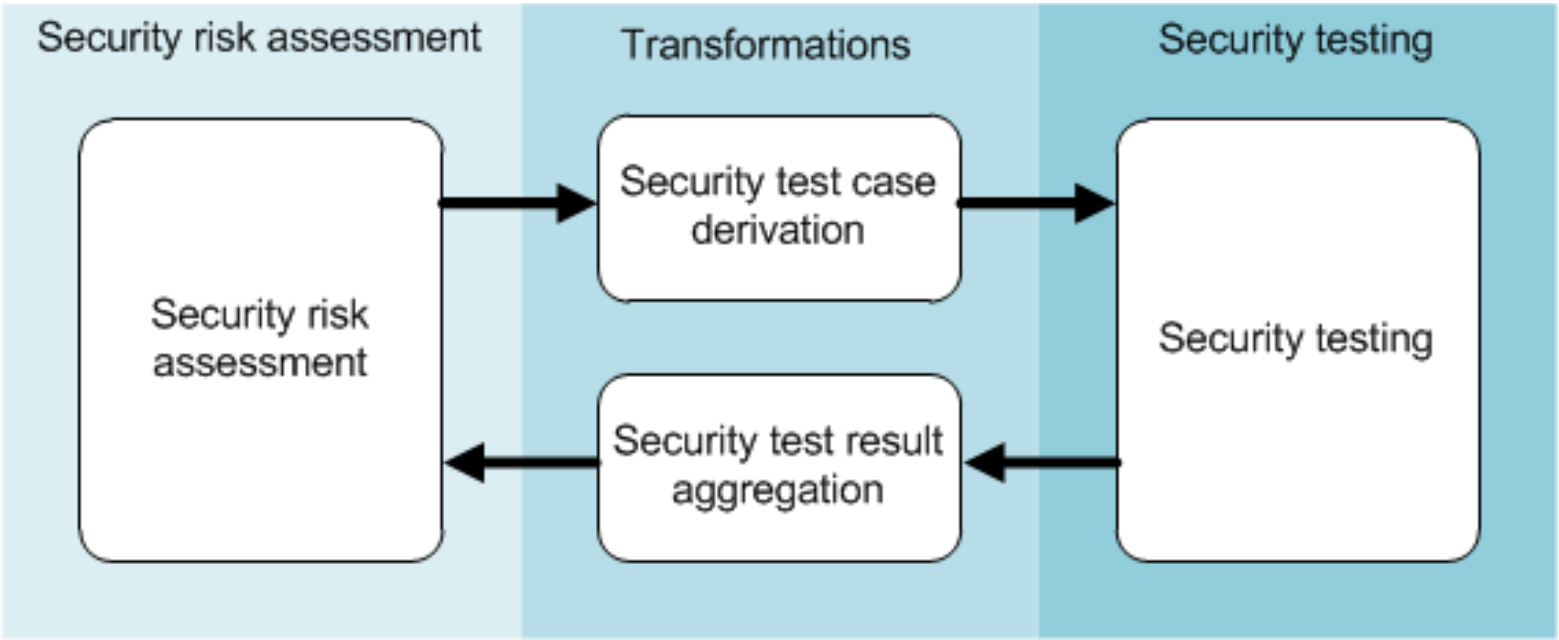
RASEN

SEVENTH FRAMEWORK PROGRAMME

Compositional Risk Assessment and Security Testing of Networked Systems

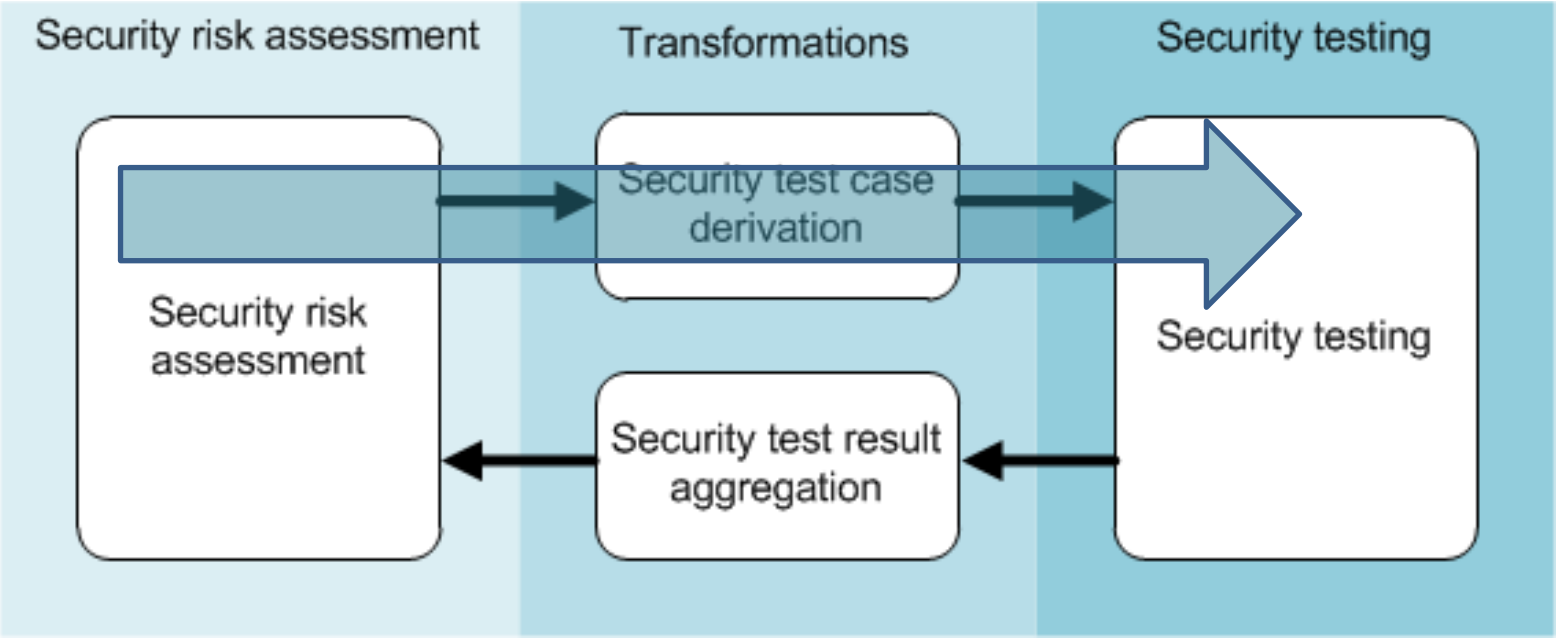
  - Strengthen European organisations' ability to conduct security assessment of large scale networked systems
    - taking into account the context in which the system is used, such as liability, legal, organisational and technical issues,
    - through the combination of compliance management security risk assessment and security testing.

Motivation of the RASEN project

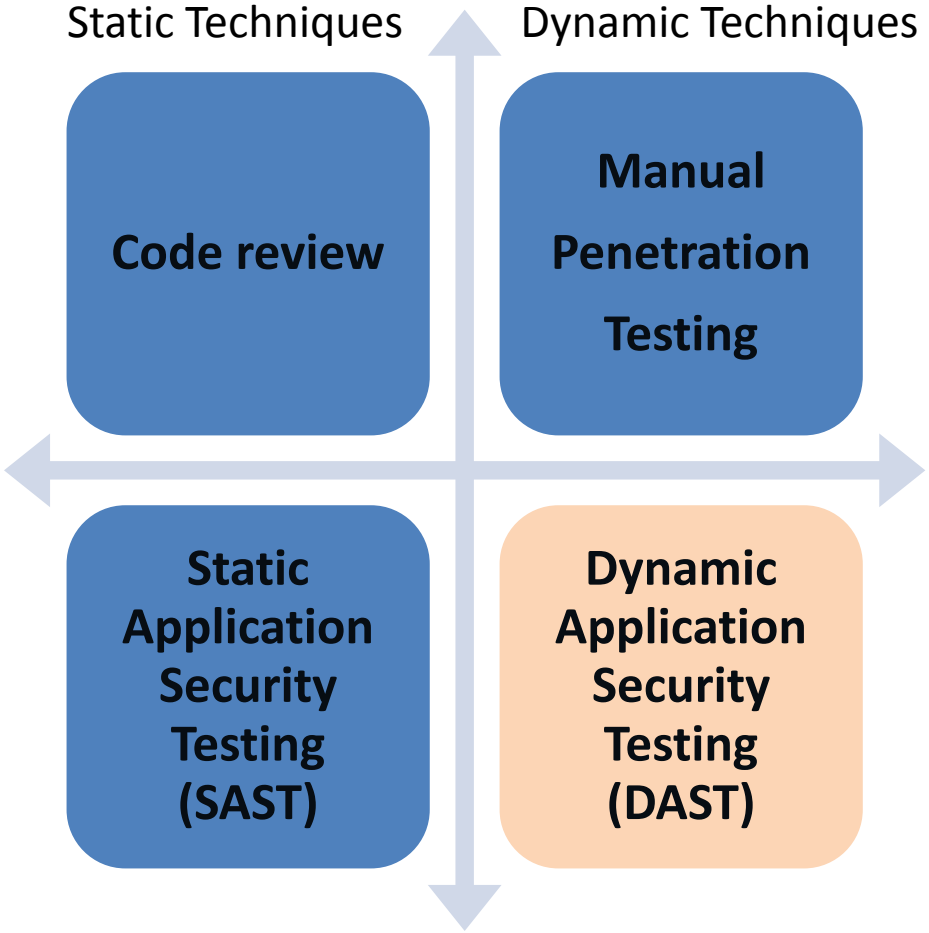


Contents of the presentation

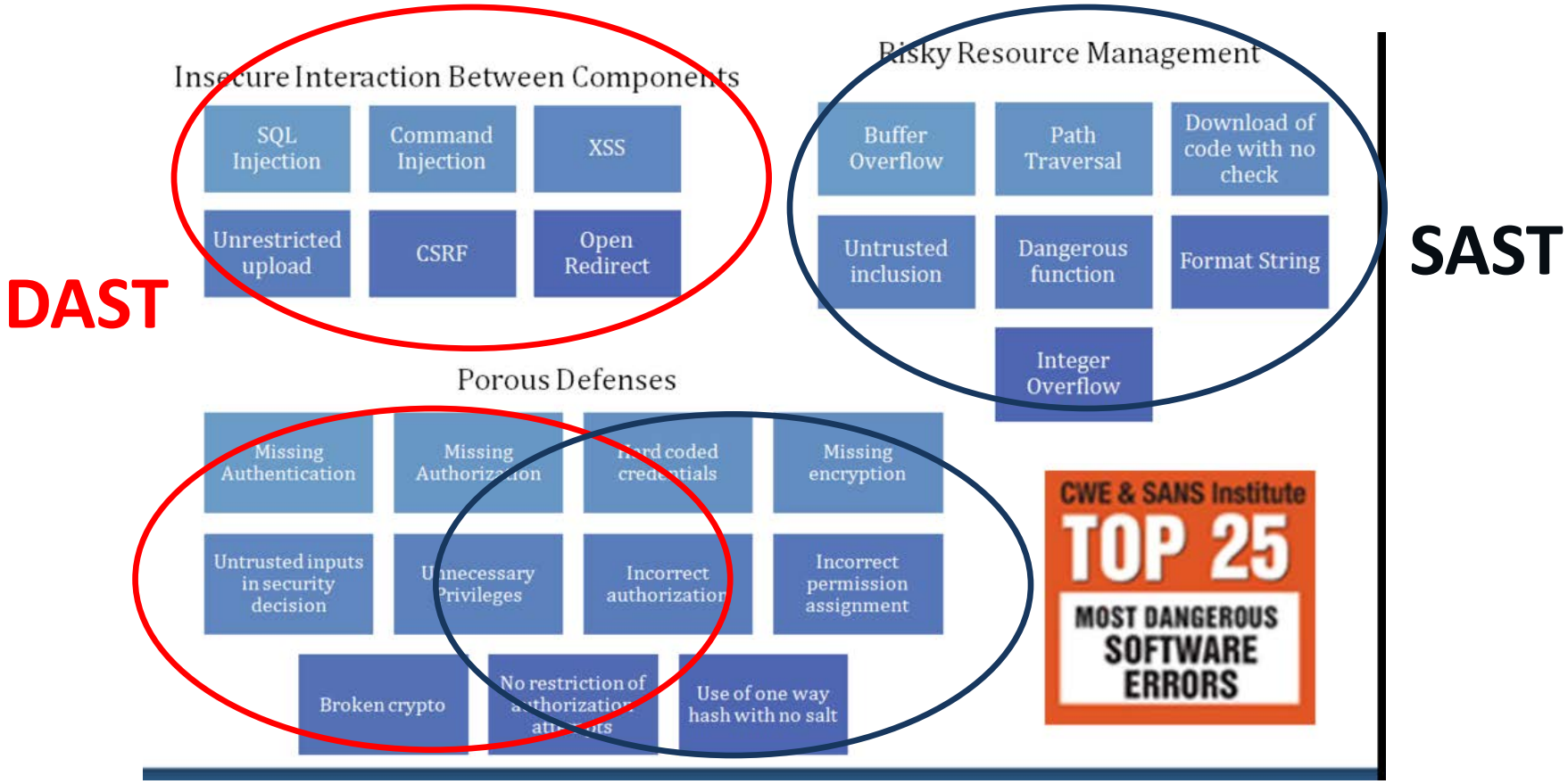
- Security and risk-based testing approach to guide the security testing using a systematic derivation of test cases from risk assessment results.



Security testing: state of the practice



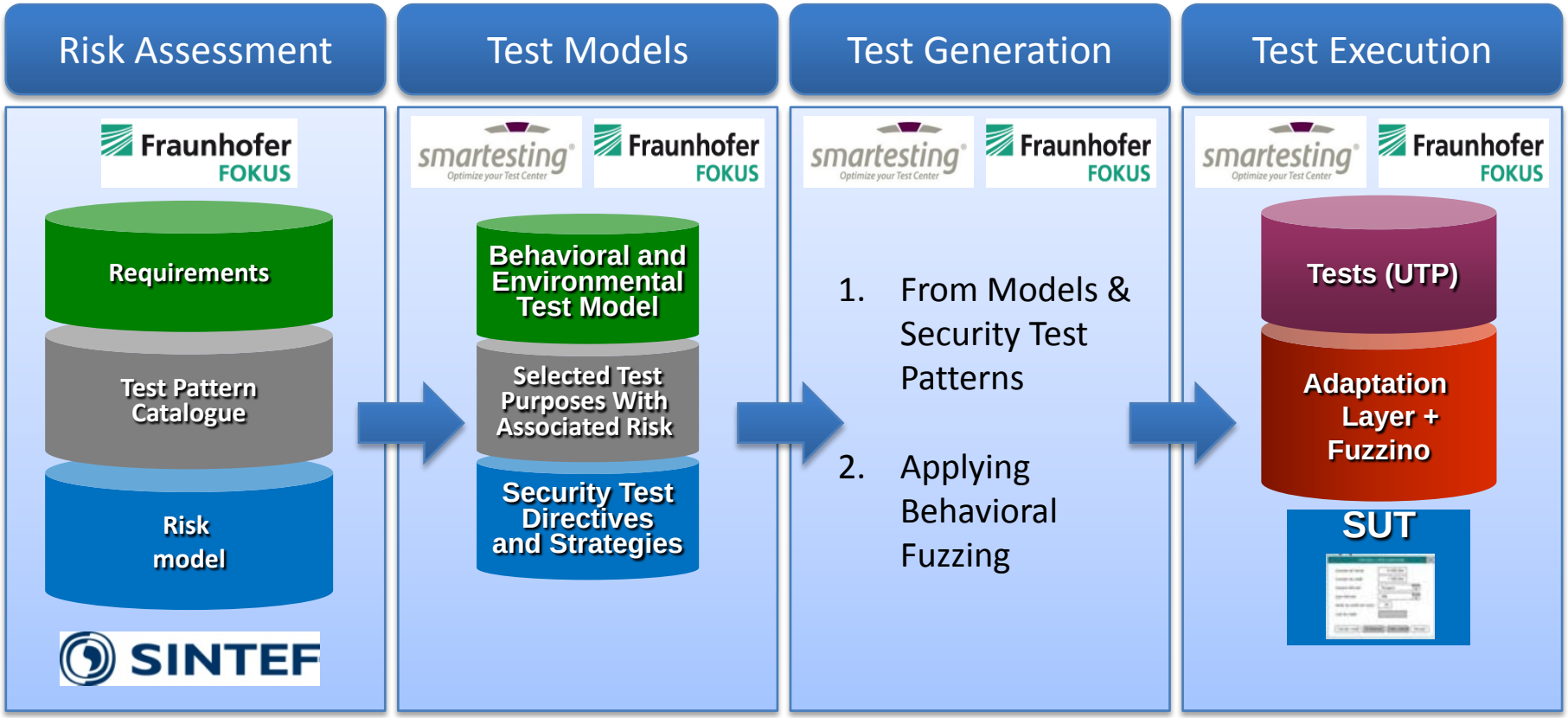
SAST vs DAST – Top 25 / 2011



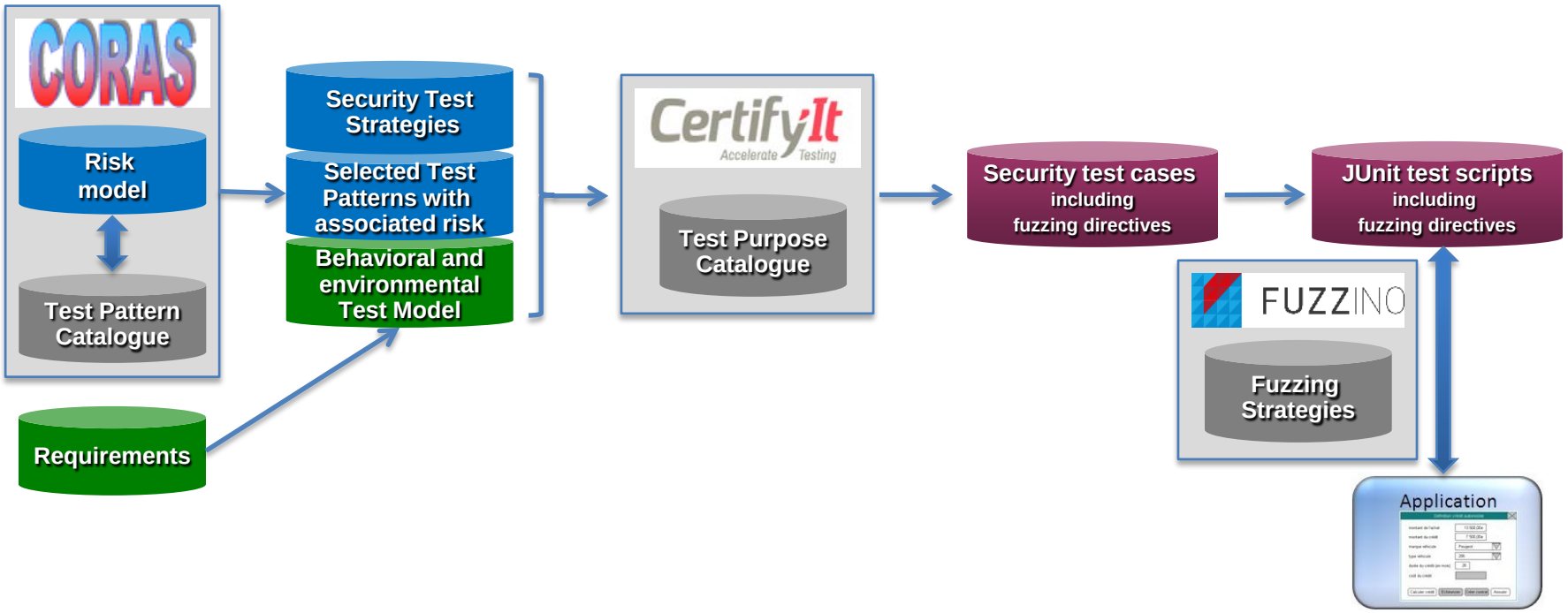
Objectives of the testing approach

- To provide a systematic guidance for DAST security testing techniques from risk assessment
- To automate test case derivation and execution using model-based security testing techniques
- To support compositional analysis to manage large scale networked system in complex environments

Risk-Based Security Testing process



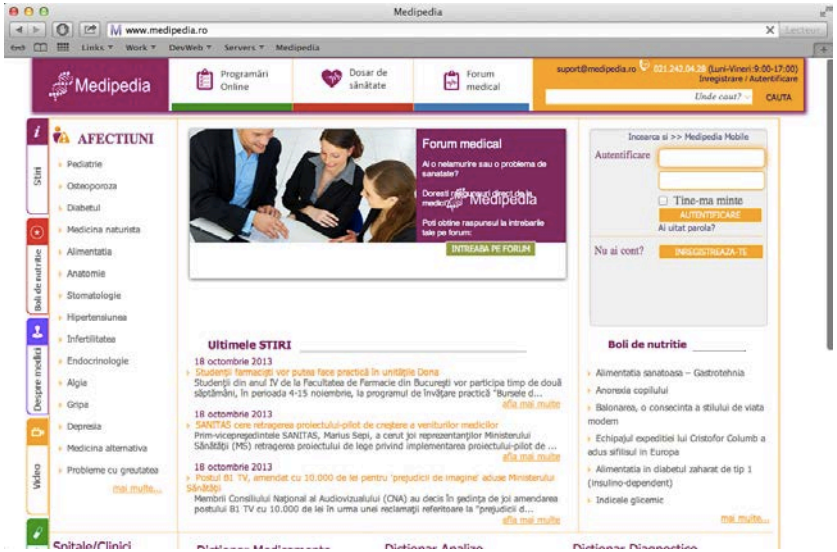
RASEN toolchain overview



Use case: InfoWorld MediPedia

Medipedia is a web service that:

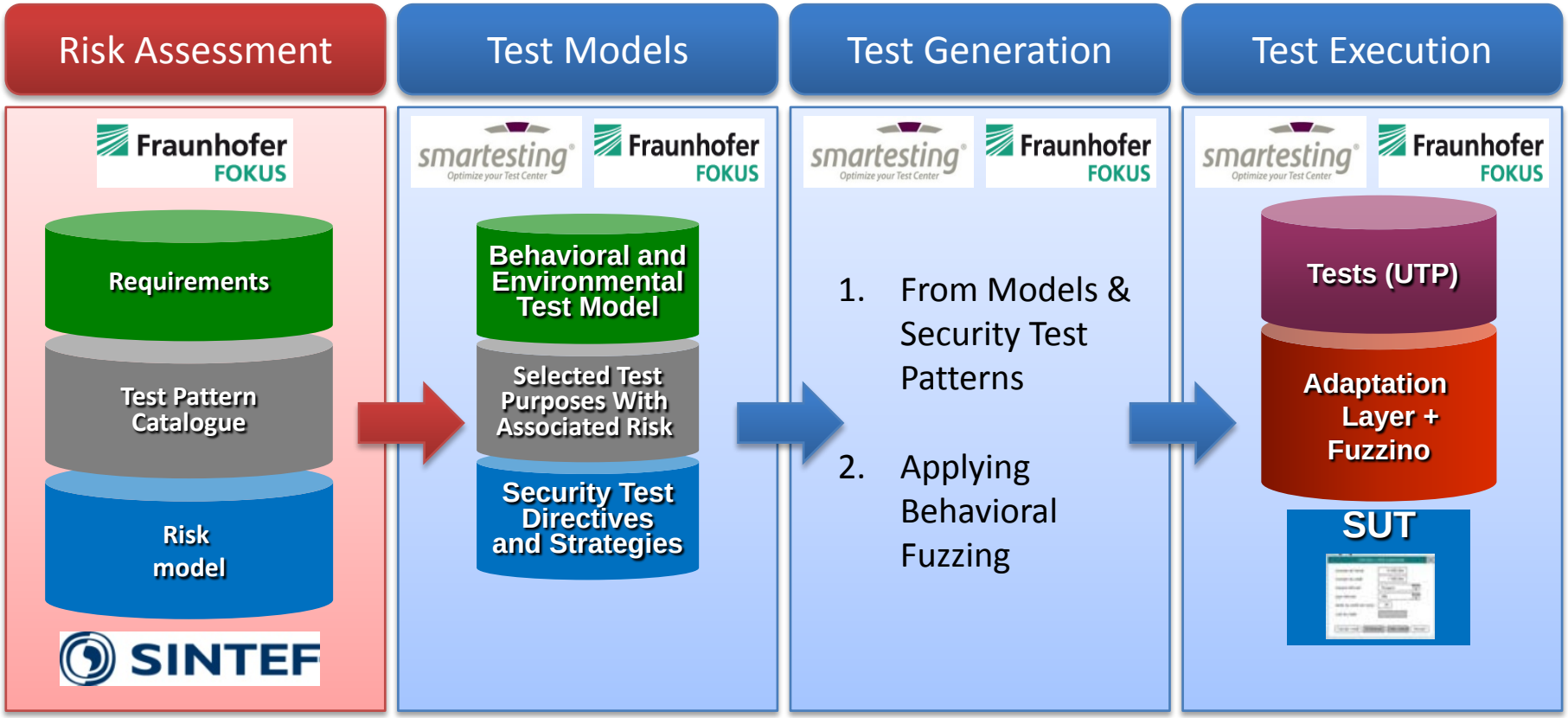
- allows patients to collect and organize all medical information, from multiple healthcare providers in a single health record,
- provides both public and secured username and password based access (public and secured information managing patient medical records).



infoworld
CONNECTING HEALTHCARE

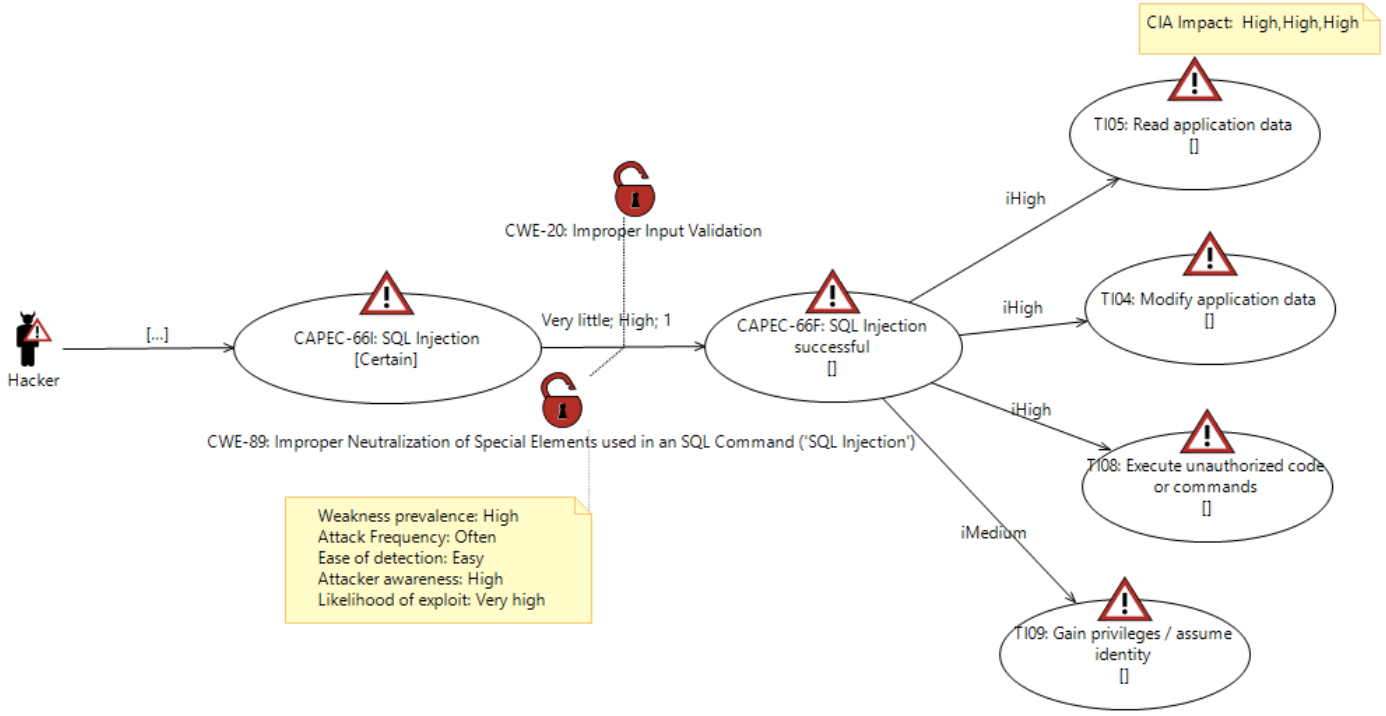
MediPedia
The Medical Encyclopedia

1. Risk assessment inputs

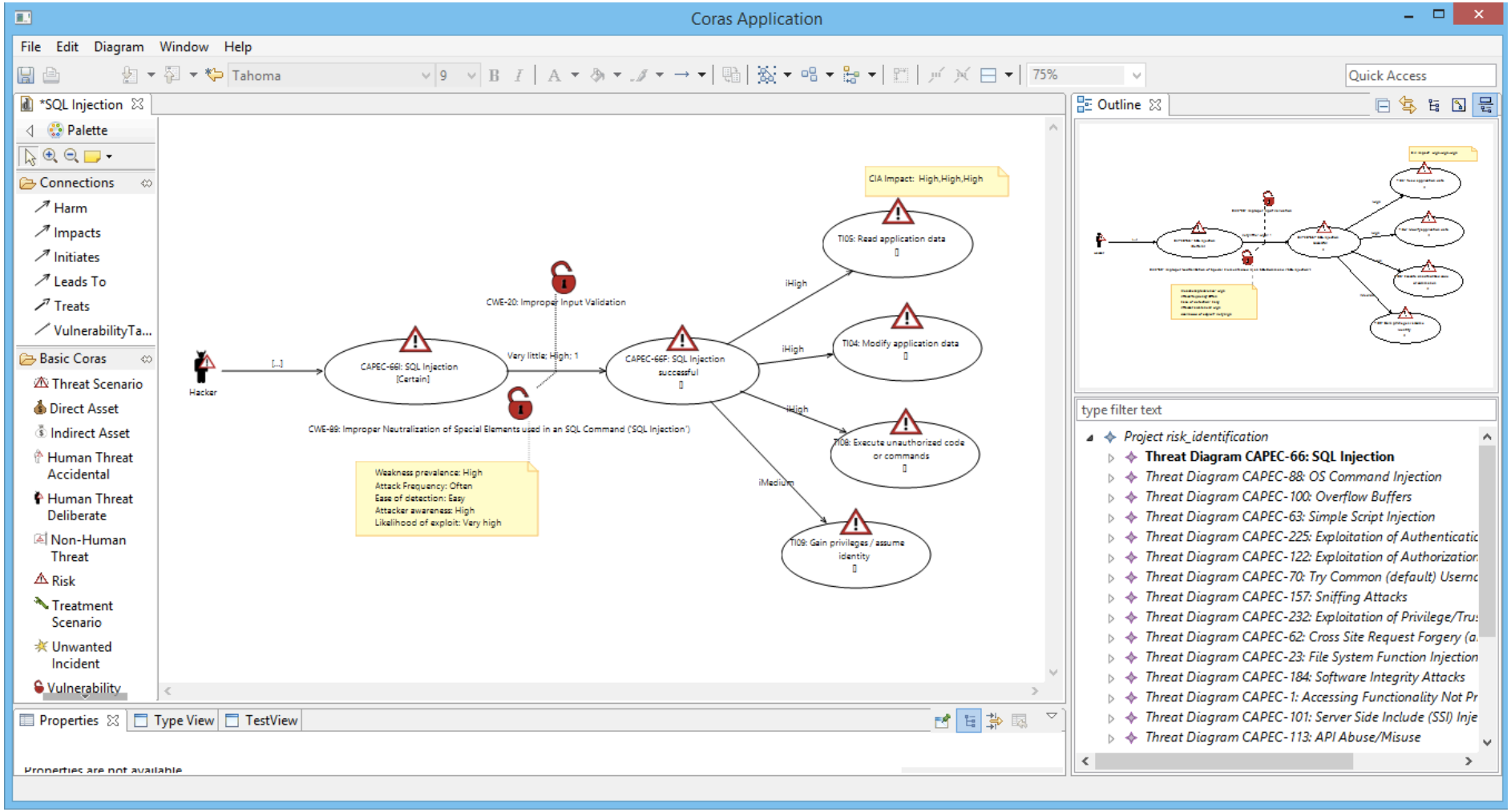


Risk identification and prioritization

- Using the CORAS approach to provide test case identification and prioritization based on the risk analysis:
  - Definition of selected test procedures from identified risk
  - Prioritization of the test procedures regarding risk assessment results



Risk model in CORAS tool

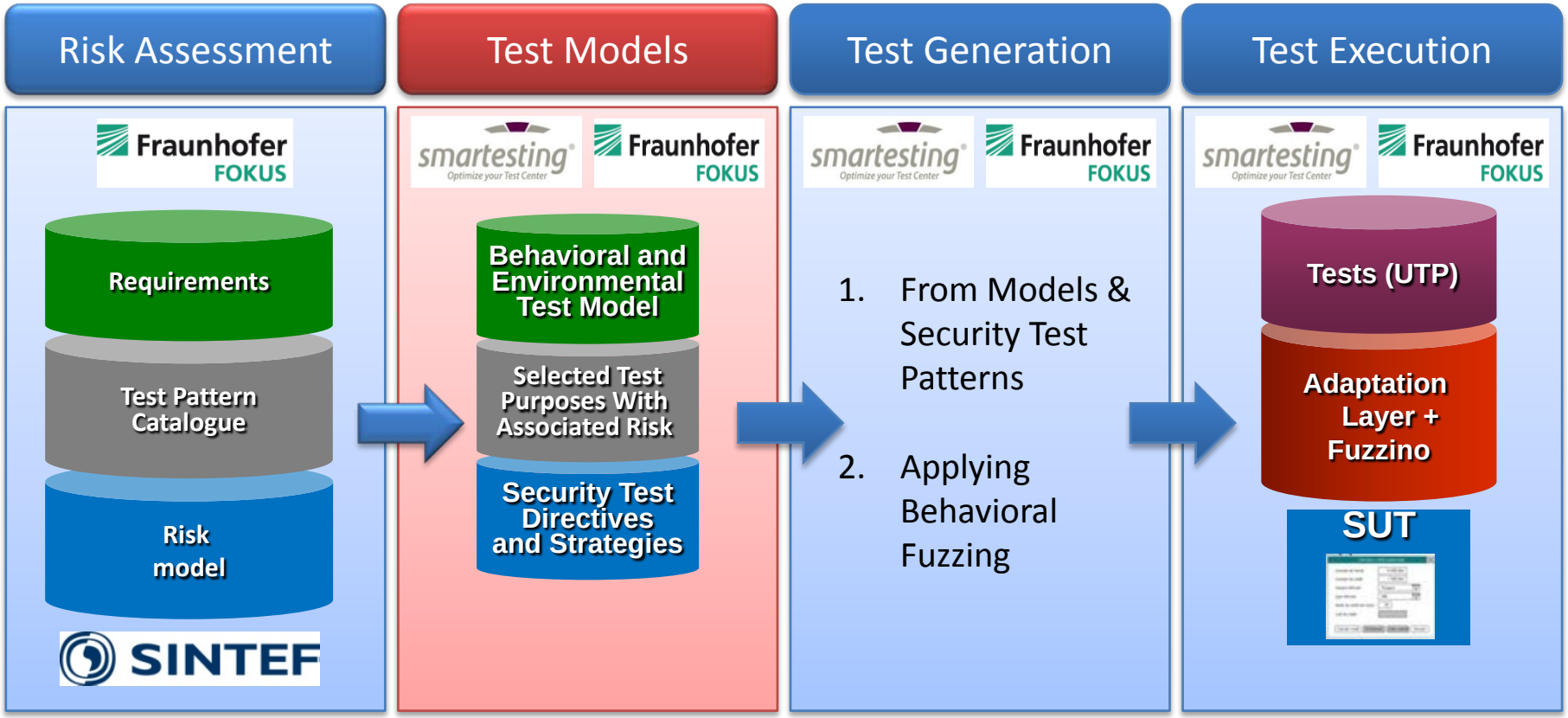


Link to Security Test Patterns

- Security test patterns are typically related to vulnerability catalogues
  - MITRE CWE & CAPEC
  - OWASP Top 10
- Solution
  - one or more **test design technique** and corresponding **strategies,** test **effort** and **effectiveness**
- Test Data
  - instructions for crafting test data
  - references to test data libraries or generators
- Tools
  - references tools that can be used to generate and execute such test cases

Pattern Name	*A meaningful name for the pattern, e.g. the name of the weakness.*	
CWE-ID(s)	*The IDs of a weakness from the Common Weakness Enumeration.*	
Weakness Description	*A high-level description of the weakness.*	
Solution	*How the weakness could be revealed manually.*	
	Test Design Technique	*Test design technique that is able to find the weakness.*
	Test Strategies	*Test strategies specific for a certain test design technique that shall be applied in order to generate test cases for the weakness in question.*
	Effort	*The effort to generate and execute such test cases on a scale with the values 'low', 'medium', and 'high'-*
	Effectiveness	*How effective is the test design technique in finding such a weakness (how many test cases are necessary to find one weakness, how many weaknesses might be missed).*
Description of Test Coverage Items	*Informal description of items to be covered by test cases created on basis of a pattern.*	
Metrics	*Appropriate test and coverage metrics. These will be developed in Task T4.3. This field is omitted within this deliverable.*	
Discussion	*A short discussion on the pitfalls of applying the pattern and the potential impact it has on test design in general and on other patterns applicable to that same context in particular.*	
Test Data	*Actual or references to test data and test data generators.*	
Tools	*References to tools appropriate for test case generation and execution.*	
Generalization of	*References to other security test patterns that are specializing this pattern.*	
References	*References to OWASP Top 10 weaknesses CWE descriptions, related CAPEC attack patterns*	

2. Test model design



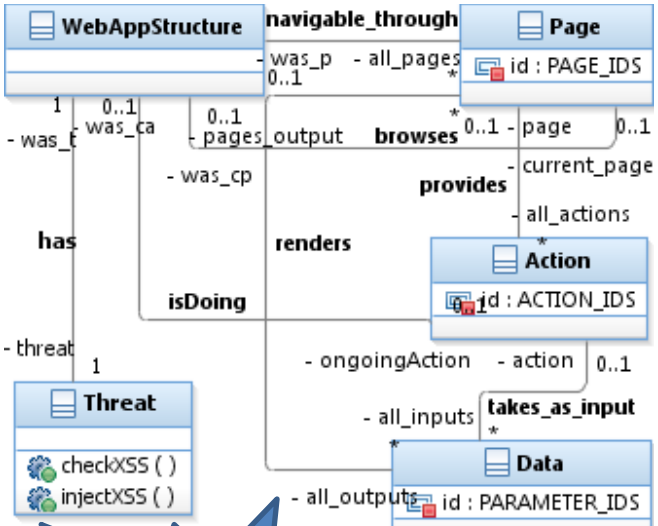
Test model and testing directives

- Testing artefacts are composed of:
  - A functional and behavioral model of the application under test
  - A set of test purposes, selected from risk assessment model (identification phase), to drive the test generation
  - The prioritization of the risk assessment model to apply an appropriate test coverage

- Smartesting Test Purpose Language is used to represent Security Test Patterns into a machine-readable language:
  - Designed for security means
  - Textual language based on regular expressions
  - Reasons in term of states to be reached and operations to be called

Behavioral model design using DSML

- Behavioral modeling notation is based on UML metamodel:
  - Class diagrams specify the static structure (points of control and observation)
  - Object diagrams specify concrete business entities
  - State diagrams graphically describe its behavioural characteristics

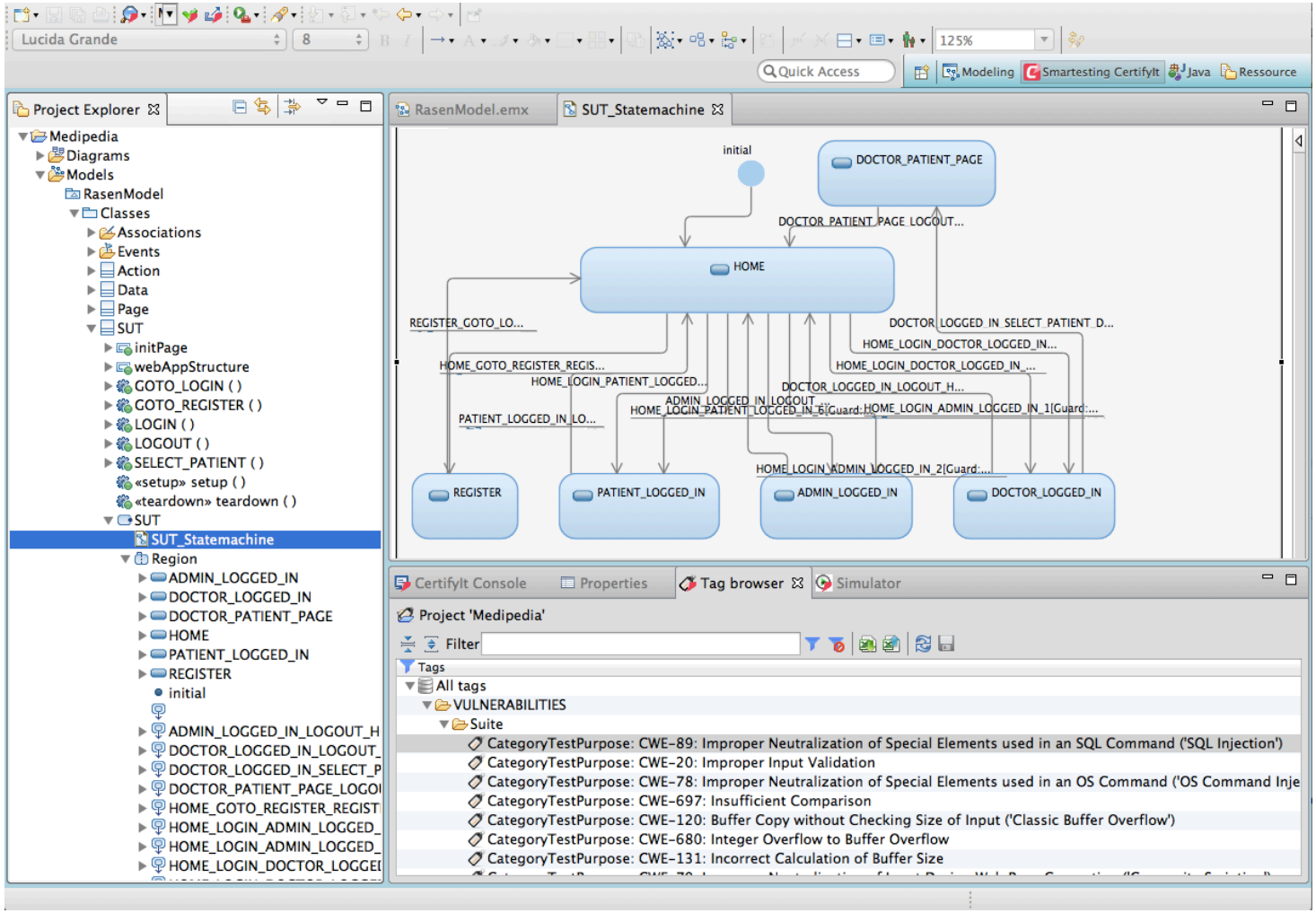


```
PAGES {
        "HOME":INIT {
                ACTIONS {
                        "LOGIN" ("USERNAME" = "admin" => "ADMIN_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "ADMIN_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "admin2" => "ADMIN_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "ADMIN_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "homed" => "DOCTOR_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "DOCTOR_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "test_med2" => "DOCTOR_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "DOCTOR_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "hopac" => "PATIENT_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "PATIENT_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "iliecatalin" => "PATIENT_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "PATIENT_LOGGED_IN"
                }
                NAVIGATIONS {
                        "GOTO_REGISTER"
                                -> "REGISTER"
                }
        }
        "ADMIN_LOGGED_IN" {
                NAVIGATIONS {
                        "LOGOUT"
                                -> "HOME"
                }
        }
        "DOCTOR_LOGGED_IN" {
                ACTIONS {
                        "SELECT_PATIENT" ("NAME" = "ILIE" => "DOCTOR_PATIENT_PAGE", "FIRST_NAME" = "Catalin" => "DOC
"DOCTOR_PATIENT_PAGE")
                                -> "DOCTOR_PATIENT_PAGE"
                }
                NAVIGATIONS {
                        "LOGOUT"
```

Generic

RASEN

State diagram from DSML

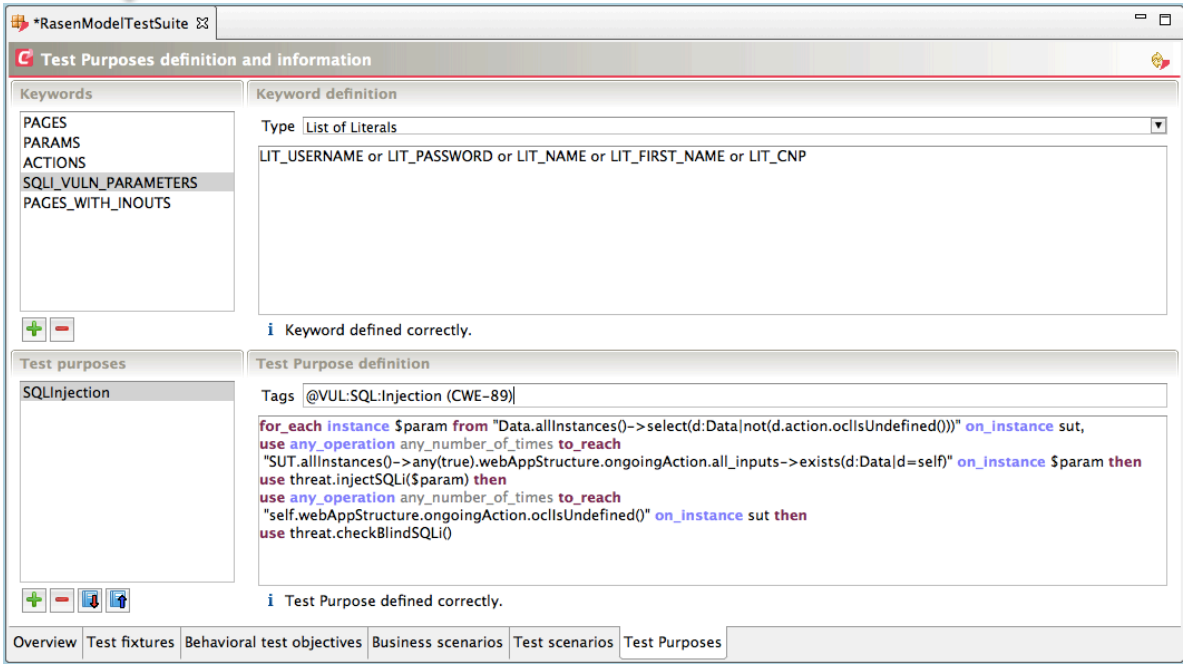


Test Purpose derivation

Pattern Name	SQL Injection
CWE-ID(s)	CWE-89
Weakness Description	The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component. **Error! Reference source not found.**
Solution	Based on attack pattern CAPEC-66 **Error! Reference source not found.** 1. Use the application, client or web browser to inject SQL constructs input through text fields or through HTTP GET parameters. 2. Use a possibly modified client application or web application debugging tool such to submit SQL constructs for submitted values or to modify HTTP POST parameters, hidden fields, non-freeform fields, etc. 3. Check for error messages, delays, disclosed values in the client application and new/modified/deleted values in the database.
Test Design Technique	Data fuzzing Pattern-based testing
Test Strategies	SQL Injection
Effort	Low to medium: can be highly automated using fuzzing techniques or SQL injection dictionaries.
Effectiveness	Medium **Error! Reference source not found.** to high, depending on detection capabilities by access to the affected database and to error messages
Description of Test Coverage Items	• Functionality that involves user input, e.g. dialogs, URLs of a web application, that might be used in a database query • User input fields • SQL injection payloads • Names of tables and rows of the database schema • Values of existing records
Discussion	SQL injection is a task that could be rather trivial but also very complex. This depends on several factors. For instance, error messages resulting from incorrect SQL constructs caused by SQL injection are very helpful in deciding whether SQL injection is generally possible. In order to detect whether table data can be modified, it is helpful to have knowledge of the database management system (different systems have little differences in SQL syntax) and the database schema (modifying existing records may require knowledge in which tables they are stored). If SQL injection is possible, the extent of SQL injection can be assessed by trying to modify existing data which requires knowledge of existing values in the database tables. This enables to determine whether existing database entries can be read, modified or deleted.
Test Data	• SQL Injection Cheat Sheet **Error! Reference source not found.** • Fuzzing library Fuzzino **Error! Reference source not found.**
Testing Tools	• Fuzzing framework Sulley **Error! Reference source not found.** • Sqlmap **Error! Reference source not found.**
Generalization of	**Error! Reference source not found.**

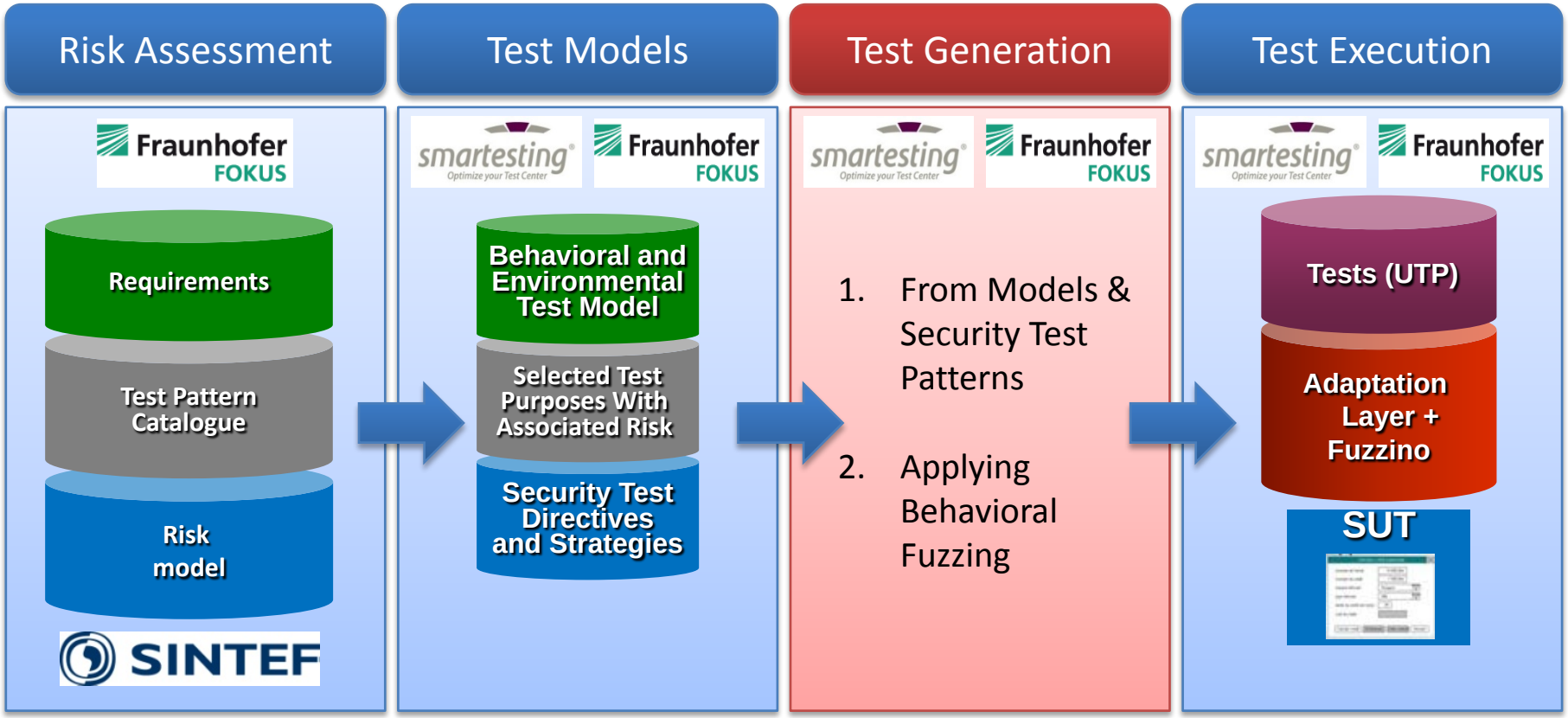
Automatic derivation from Test Pattern to Test Purpose:

- Linked to model by using keywords
- Testing directives inherited from Test Patterns



Generic

3. Security test generation



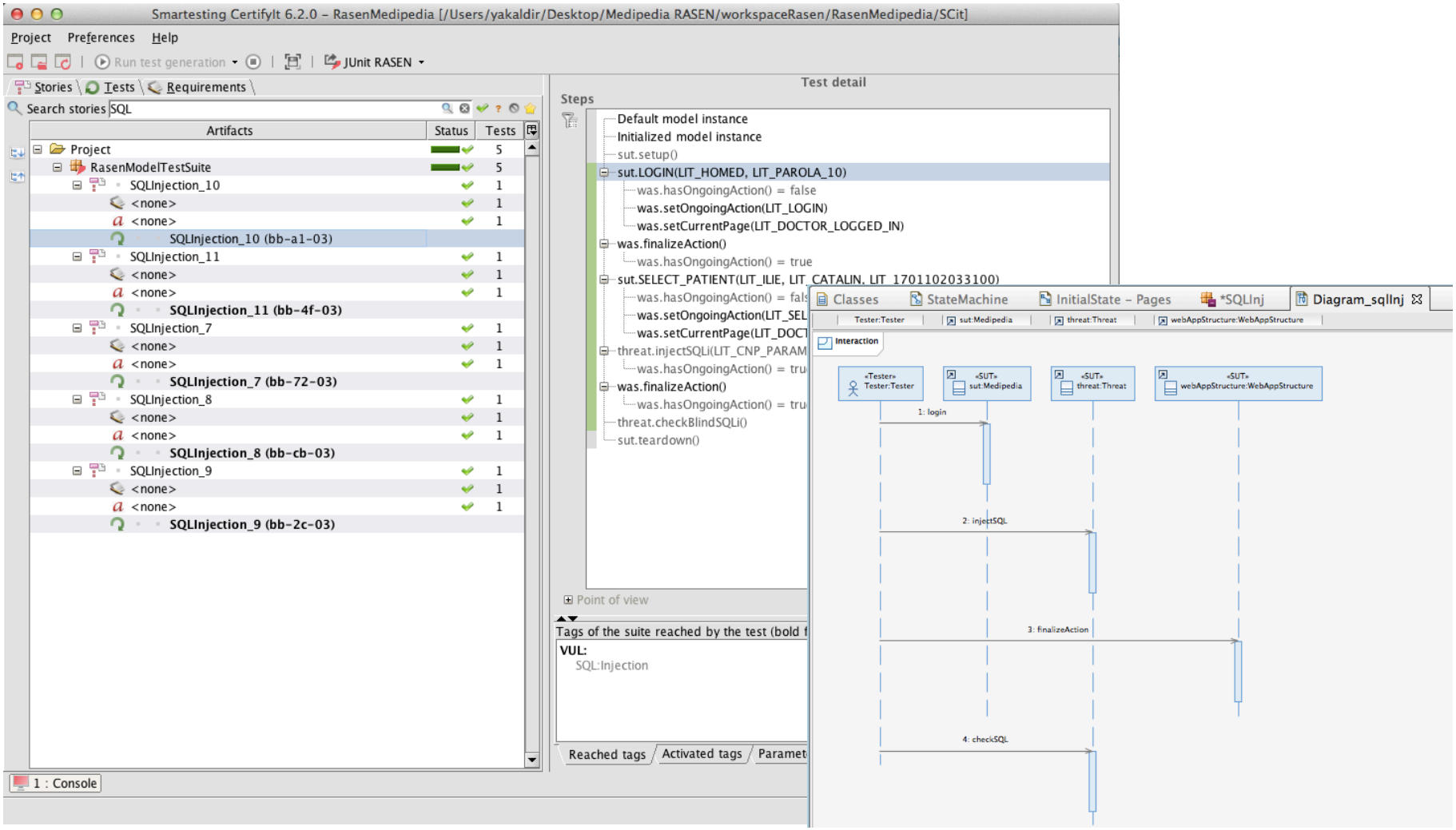
Test generation strategies

Test cases are automatically generated using Smartesting CertifyIt by composing behavioral models and test purposes:

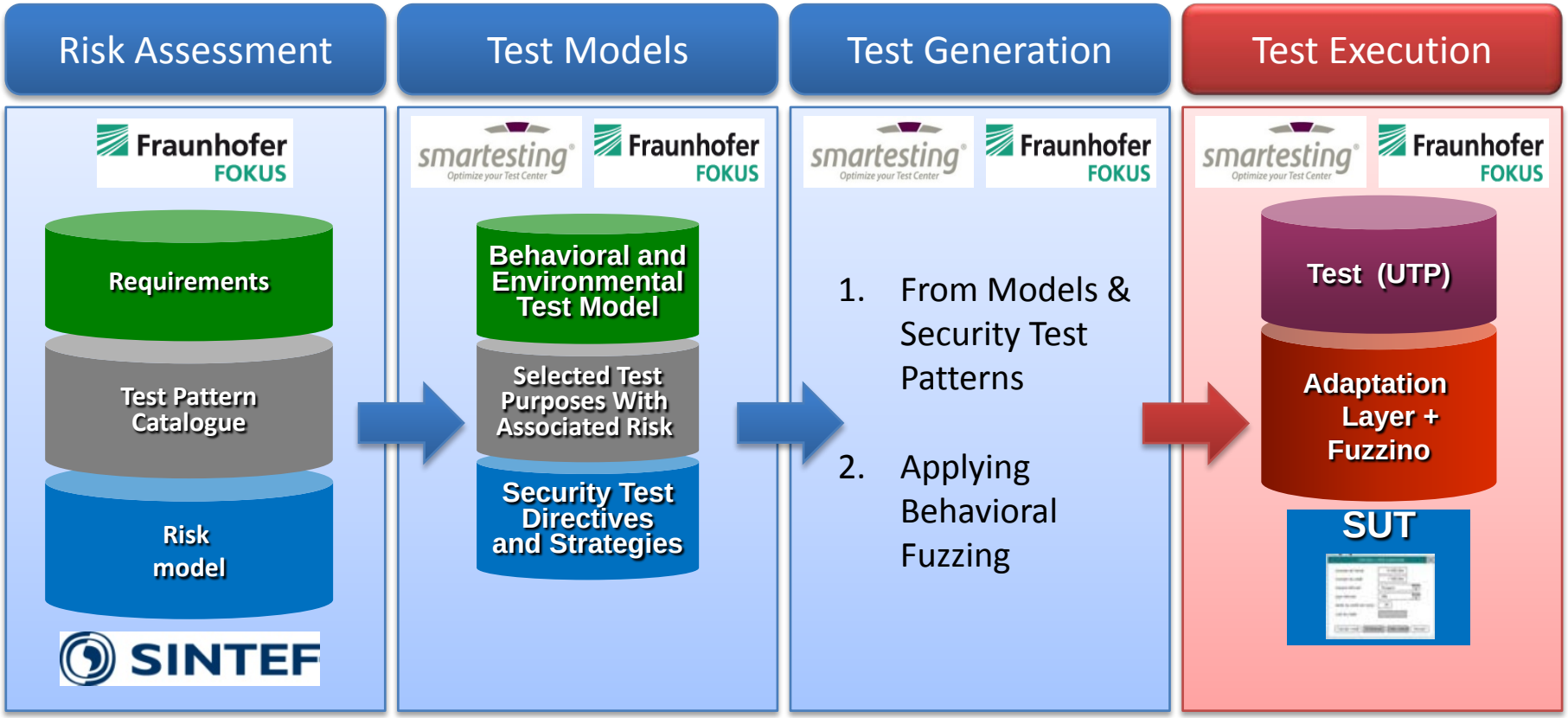
- For one Test Purpose, several (or many) test cases by:
  - Applying usual Test Purpose coverage criteria
  - Applying behavioral fuzzing strategy given from Test Patterns
- Traceability management from security requirements to generated tests is build-in

Result: a suite of abstract security test cases

Test generation results using CertifyIt



4. Test concretization for execution



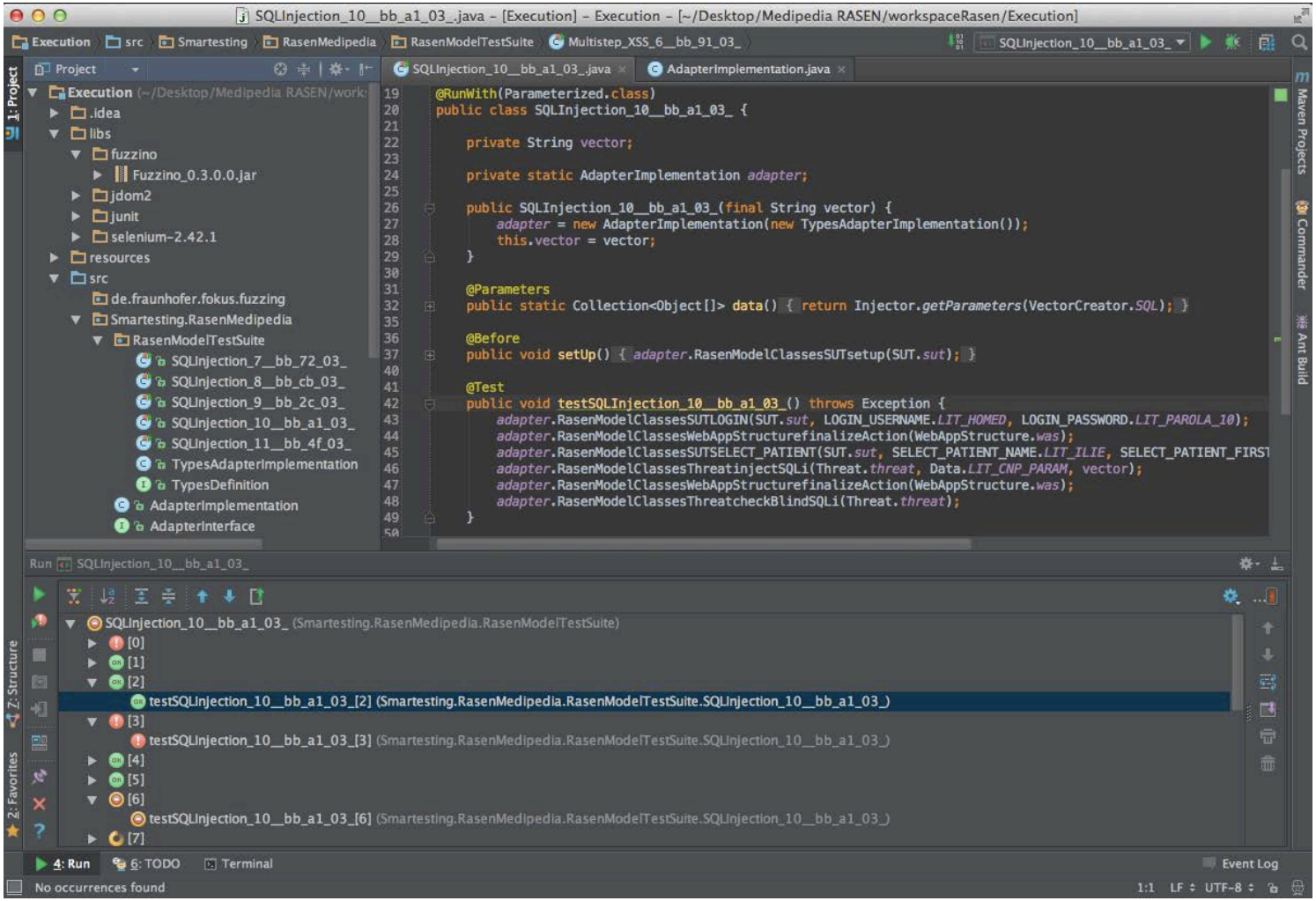
Generation of executable test scripts

JUnit test scripts are automatically generated by CertifyIt using an adaptation layer concretizing abstract data into concrete values:

- For one abstract test case, several (or many) executable test cases by:
  - Using a set of selected test data given from Test Patterns
  - Applying data fuzzing strategy given from Test Patterns
- Traceability management from security requirements to executable tests is build-in

Result: a set of executable security test scripts

Tests Execution in JUnit environment



Conclusion and future work

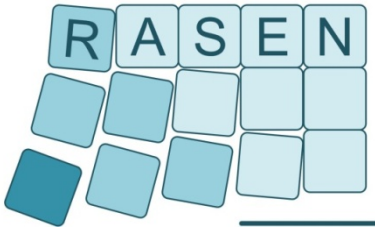
- Extended security test patterns for risk-based test case generation
- Formalization of security test patterns into test purpose language to drive the risk-based test generation
- Risk-based testing approach combining RASEN partners risk assessment and testing techniques:
  - Risk identification and prioritization using CORAS method
  - Import of risk assessment results from CORAS tool into CertifyIt
  - Test purpose generation method (CertfyIt)
  - Behavioral and data fuzzing strategies (Fuzzino)

- Definition of more accurate testing strategies regarding risk prioritization
- Extension of security test patterns and related test purposes
- Improvements of the tool integration (especially Test Purpose / fuzzing)
- Deeper use case evaluation, especially to validate the approach regarding large scale systems

Thank you for your attention!

Questions and Comments?

http://www.rasenproject.eu/



Compositional Risk Assessment and Security Testing of Networked Systems

The project can also be followed on Twitter & LinkedIn:
@RASENProject
#RASENProject
http://www.linkedin.com/groups?home=&gid=7429037