



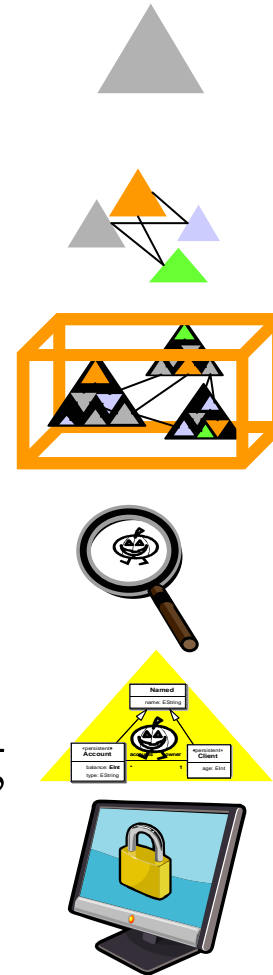
⊕ Security testing: a key-challenge for software engineering

Yves Le Traon



➔ Research domains

- Component testing
- Integration testing
- System testing
- Fault localization
- Test and Model-driven engineering
- Security testing



➤ Work (to be) presented at

- **IEEE ISSRE 2008 (Int. Symposium on Software Reliability Engineering)**
- **3rd IEEE Mutation Workshop (Mutation Testing wksp)**
- **1st IEEE SECTEST 08 (Security Testing Workshop)**
- **IEEE ICST 2008**



**First IEEE International Conference on
Software Testing, Verification and Validation**

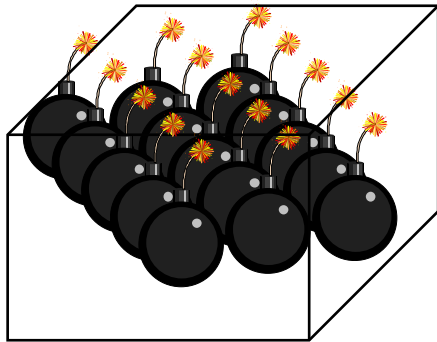
<http://www.cs.colostate.edu/icst2008/>



➔ Software testing

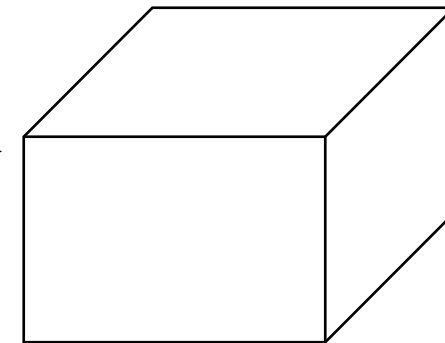
« in God we trust, for the rest we test » (A. Petrenko)

Testing

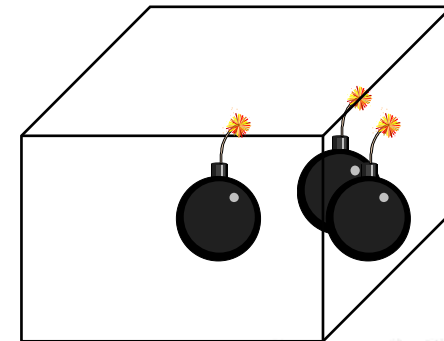


—————→ *Conformance of a product w.r.t. a specification*

Verification/proof

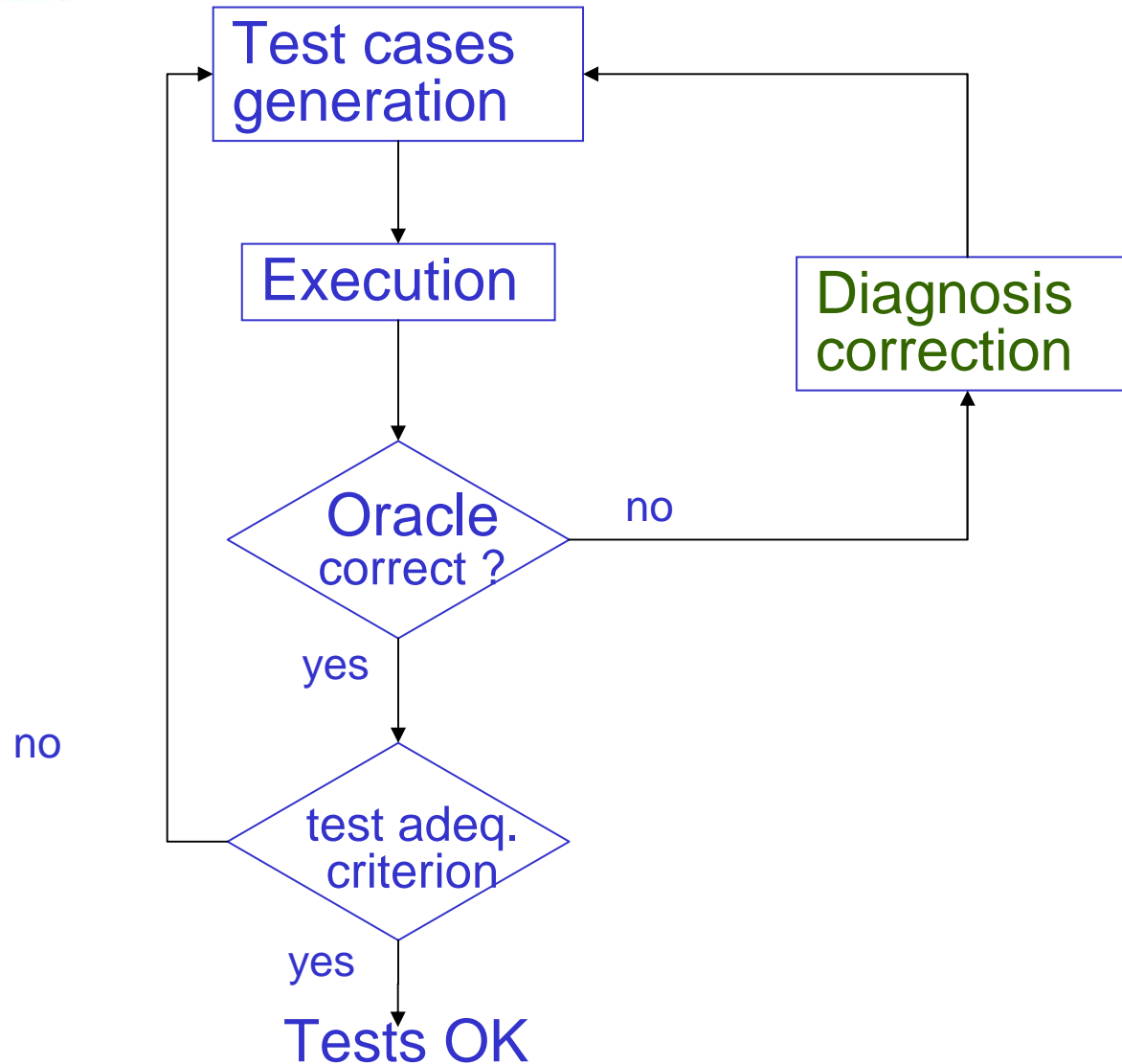


Testing



From certitudes to trust
Empirical results are needed

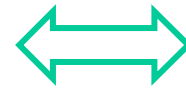
➔ Test steps





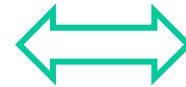
➔ Testing and security in parallel

Security



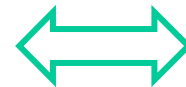
Correction

Vulnerability analysis



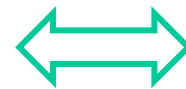
Testability analysis

Breach detection



Fault detection

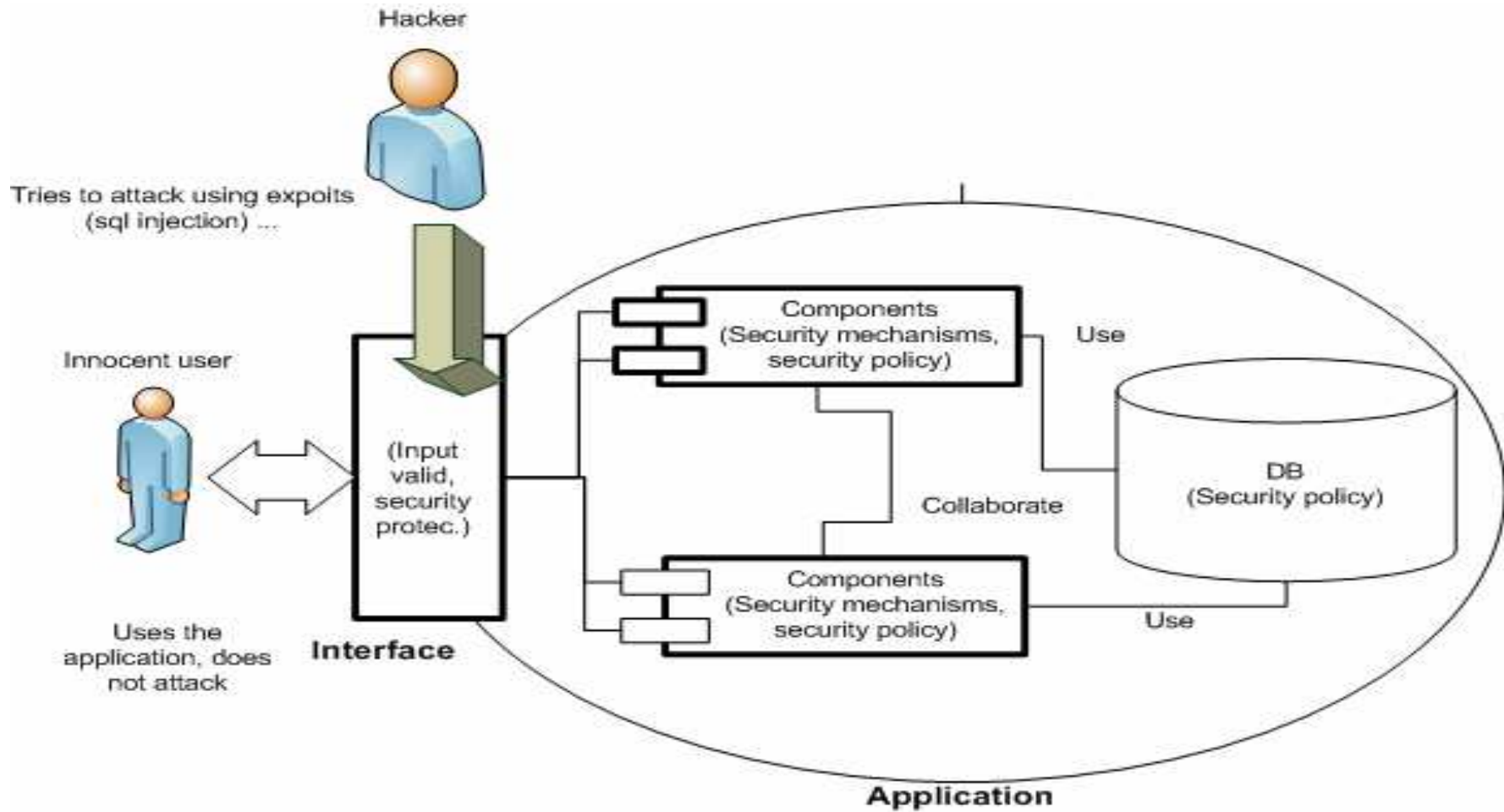
Security policy



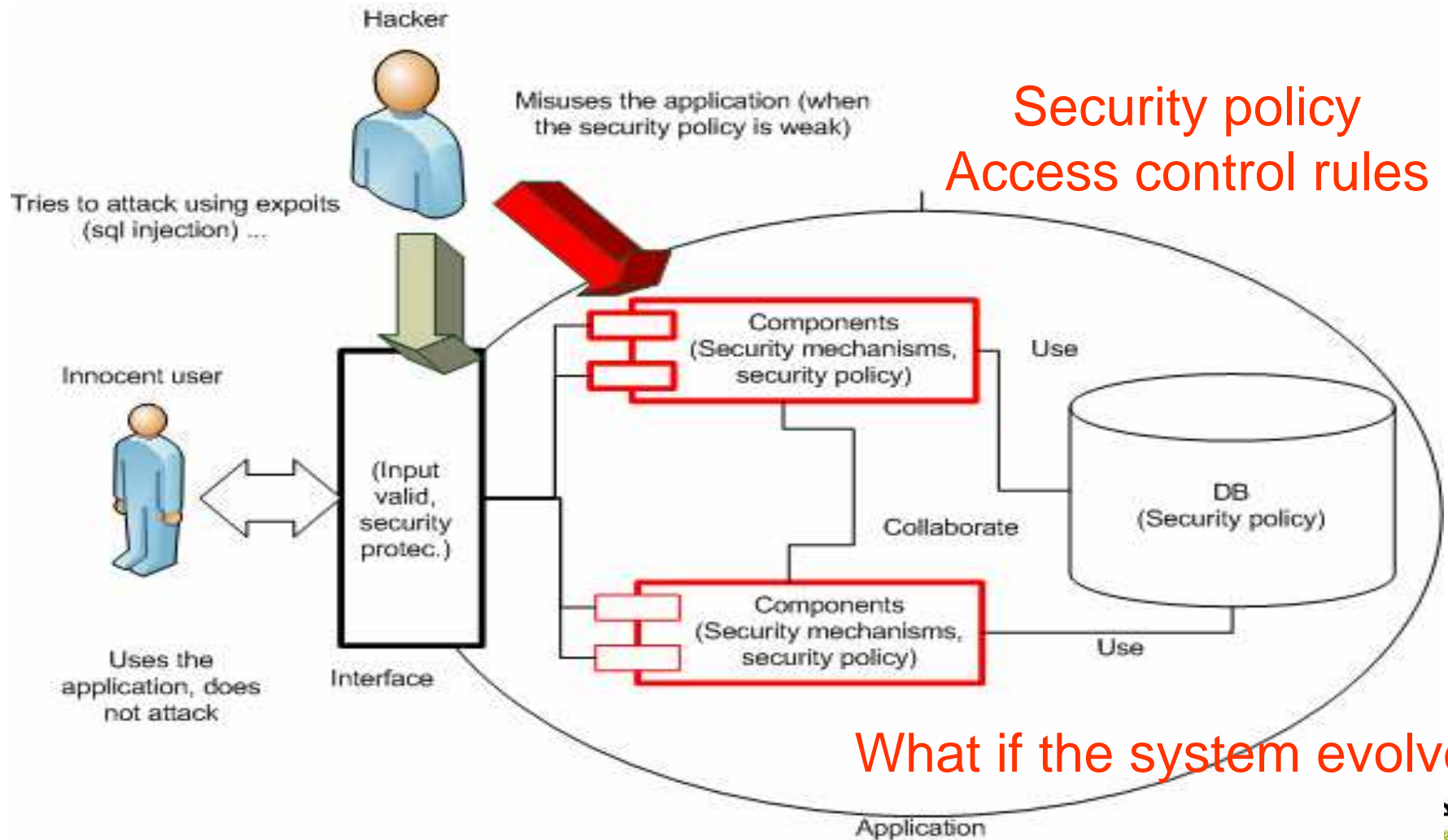
Robustness policy

Testing system's security ?

➔ Securing a web application



➔ Securing a web application



Security policy
Access control rules

What if the system evolves ?



➔ Overview

- **Security policies and access control models**
- **Testing security policies**
- **Mutation operators for comparing criteria**
- **Results analysis**
- **7 issues**
- **Conclusion**

➔ Security : general definition

- **Protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction.**
- **CIA**
 - **Confidentiality**
 - accessed, used, copied, or disclosed by persons who have been authorized to access, use, copy, or disclose the information
 - **Integrity**
 - data can not be created, changed, or deleted without authorization
 - **Availability (and correctness) of**
 - the information and the security controls (opposite of availability is denial of service - DOS)
- **Confidentiality, possession or control, integrity, authenticity, availability, and utility.**
- **Ex of approach for confidentiality: Encryption/cryptography**

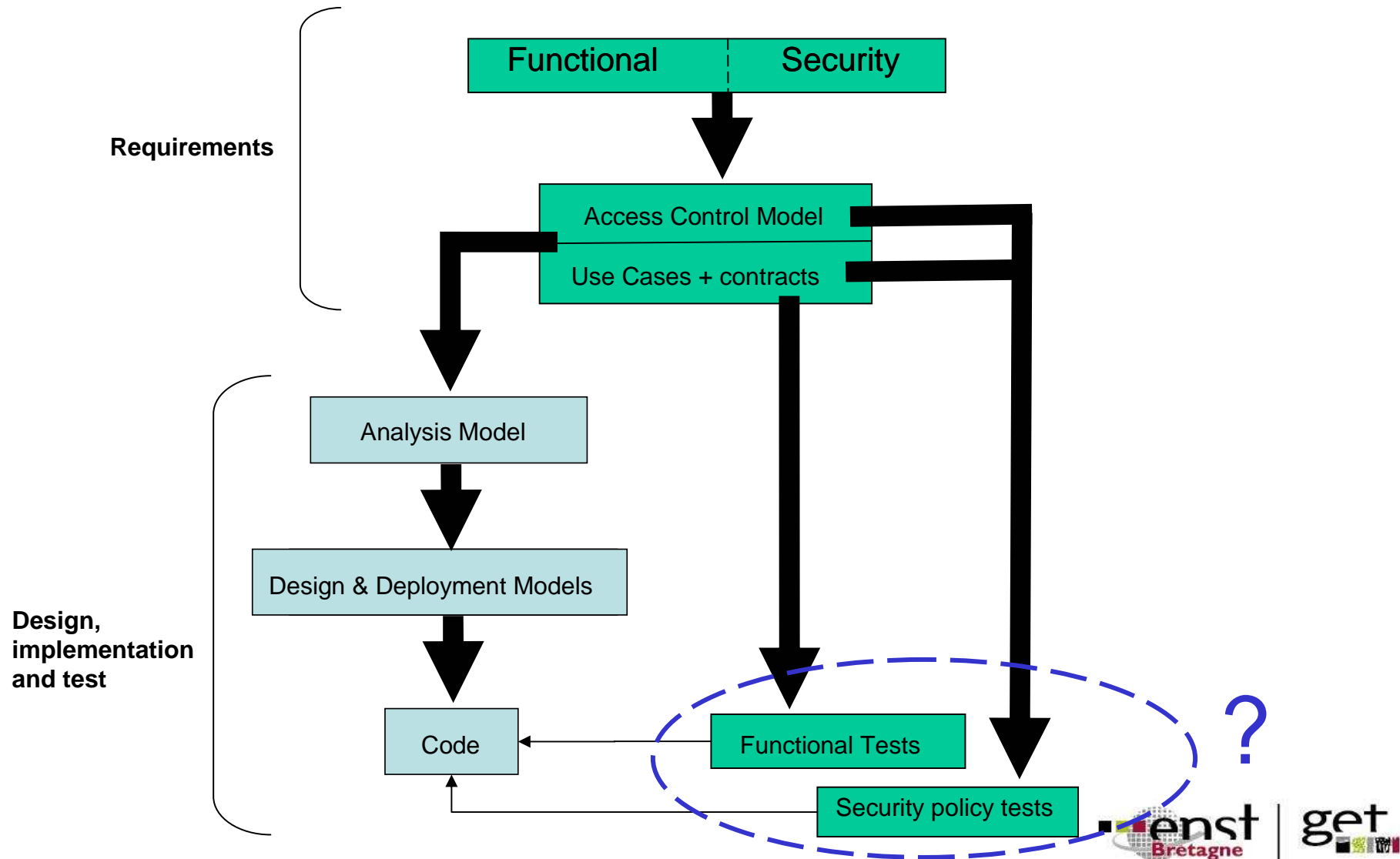


➔ Vocabulary

➤ Risk management

- **Risk** is the likelihood that something bad will happen that causes harm to an informational asset (or the loss of the asset).
 - A **vulnerability** is a weakness that could be used to endanger or cause harm to an informational asset.
 - A **threat** is anything (man made or act of nature) that has the potential to cause harm.
- **A security mechanism is the implementation of a security requirement (e.g. access control rule)**

Security in the development process



⊕ Security requirements: a library management system

- offer services to manage books in a public library
- books can be borrowed and returned on working days. When the library is closed, users can not borrow books. When a book is already borrowed, a user can make a reservation for this book. When the book is available, the user can borrow it.
- managed by an administrator (create, modify and remove accounts for new users).
- A secretary who can order books, add them in the LMS when they are delivered. The secretary can also fix the damaged books (maintenance days)
- The director of the library has the same accesses than the secretary and he can also consult the accounts of the employees
- The administrator and the secretary can consult all accounts of users.
- All users can consult the list of books in the library
- Three types of users: public users who can borrow 5 books for 3 weeks, students who can borrow 10 books for 3 weeks and teachers who can borrow 10 books for 2 months

Security is mixed with functional aspects

⊕ Security policies: Access control models

➤ Informal definition:

- Express permissions or prohibitions for people to access any of the resources of the system.

➤ May take into account the circumstances or contexts (delegation, temporal or spatial properties)

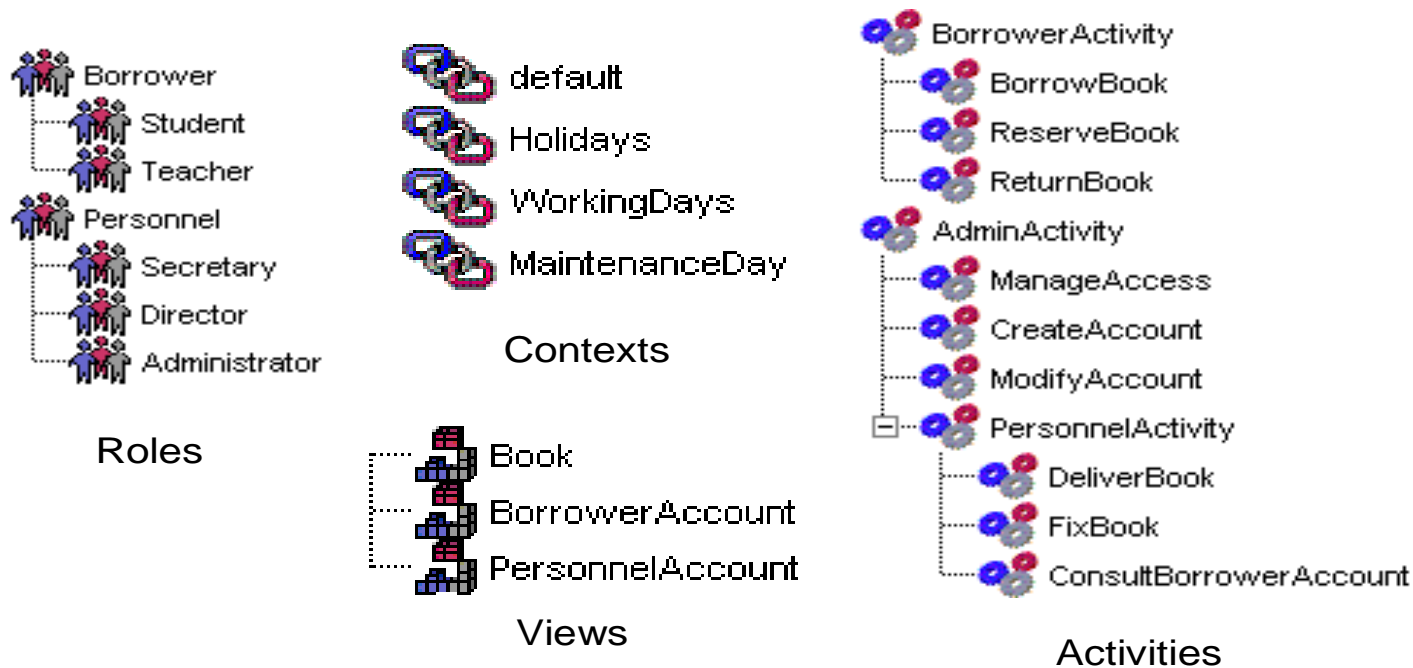
- In the LMS, we must distinguish between working days, holidays and maintenance days.

The logo for OrBAC, featuring a yellow circle with a right-pointing arrow to the left of the text "OrBAC" in a bold, yellow, sans-serif font.

OrBAC

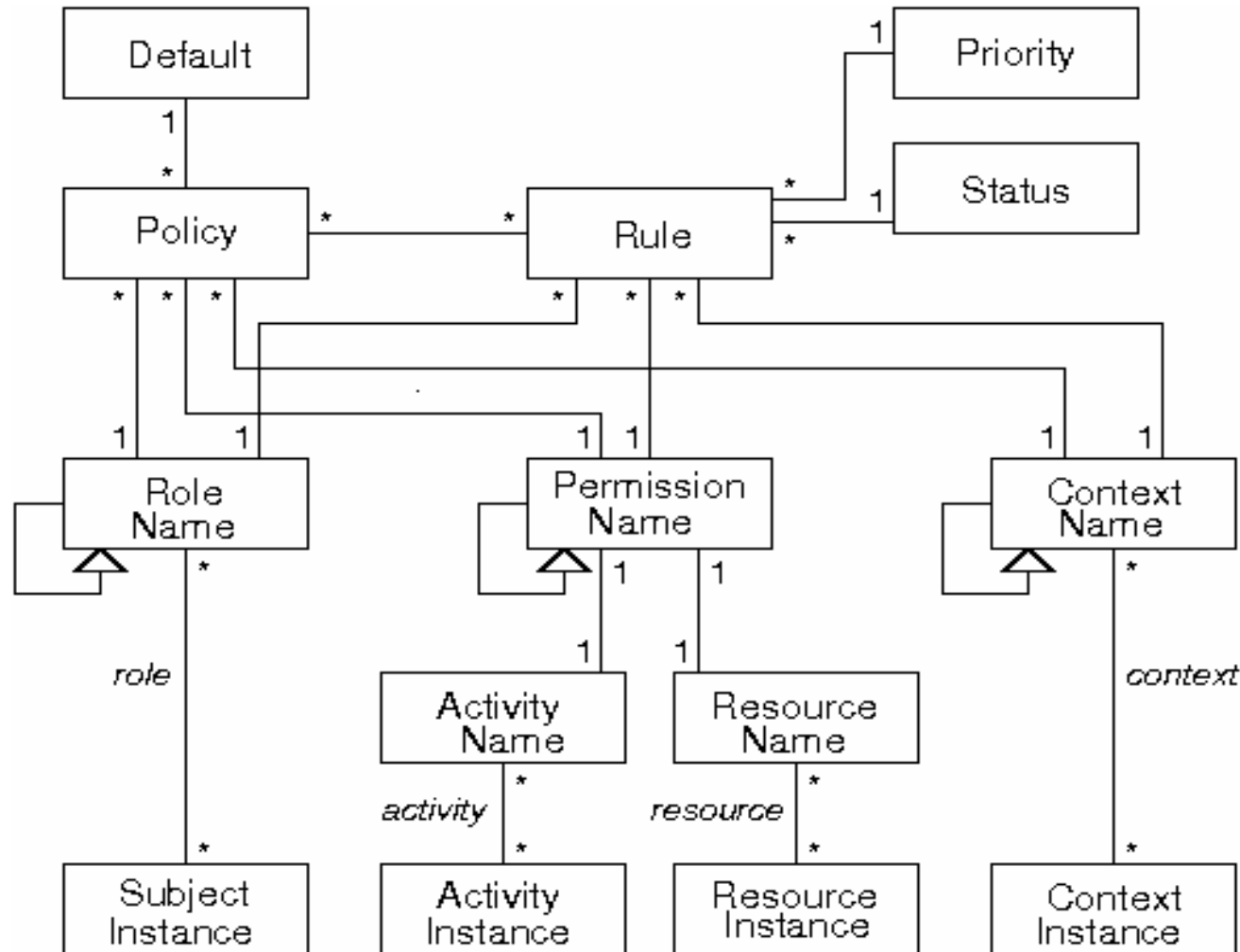
- **OrBAC : Organization based access control.**
- **Rules of permission, prohibition or obligation**
 - . Rule signature : the organization, the role, the activity, the view and the context
 - . *Permission(org,role,activity,view,context).*
 - . Example:
 - . *Permission(Library,Borrower,Borrow,Book,Holidays).*
- **Hierarchies, rules derivation and conflict management.**
- **A tool : MotOrBAC**

OrBAC

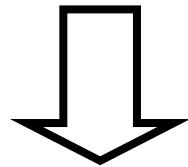
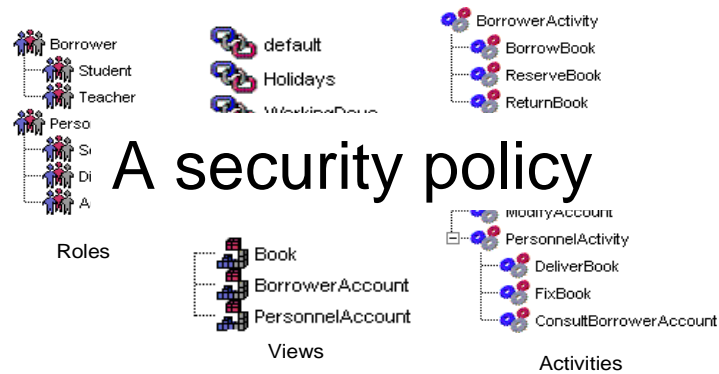


- **20 primary rules, 22 derived rules, 35 concrete rules**
- **Examples of rules**
- *Permission(Library,Administrator, ModifyAccount, BorrowerAccount, WorkingDays).*
- *Permission(Library,Personnel, FixBook, Book, MaintenanceDay).*

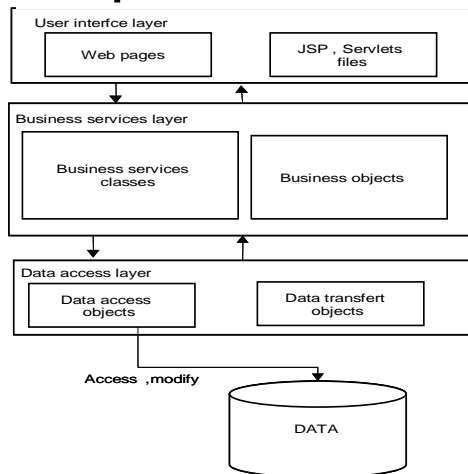
⊕ Security policy – class model



➔ Security policy testing



Implementation



Test cases

➤ Test case: an example

➤ *Intent*

- permission for a borrower to borrow a book the working days.

➤ *Test sequence*

- create the sequence reaching the context of a working day,
- make a borrower borrow a book.

➤ *Oracle*

- interrogate the security mechanism and check that the permission has been computed and given to the borrower.

➔ Test criteria

➤ **Functional test cases**

- system tests, generated based on the use cases.

➤ **CR1**

- a test case for each primary access control rule of the security model.

➤ **CR2**

- a test case is generated for each primary and secondary rule of the security model, except the generic rules.

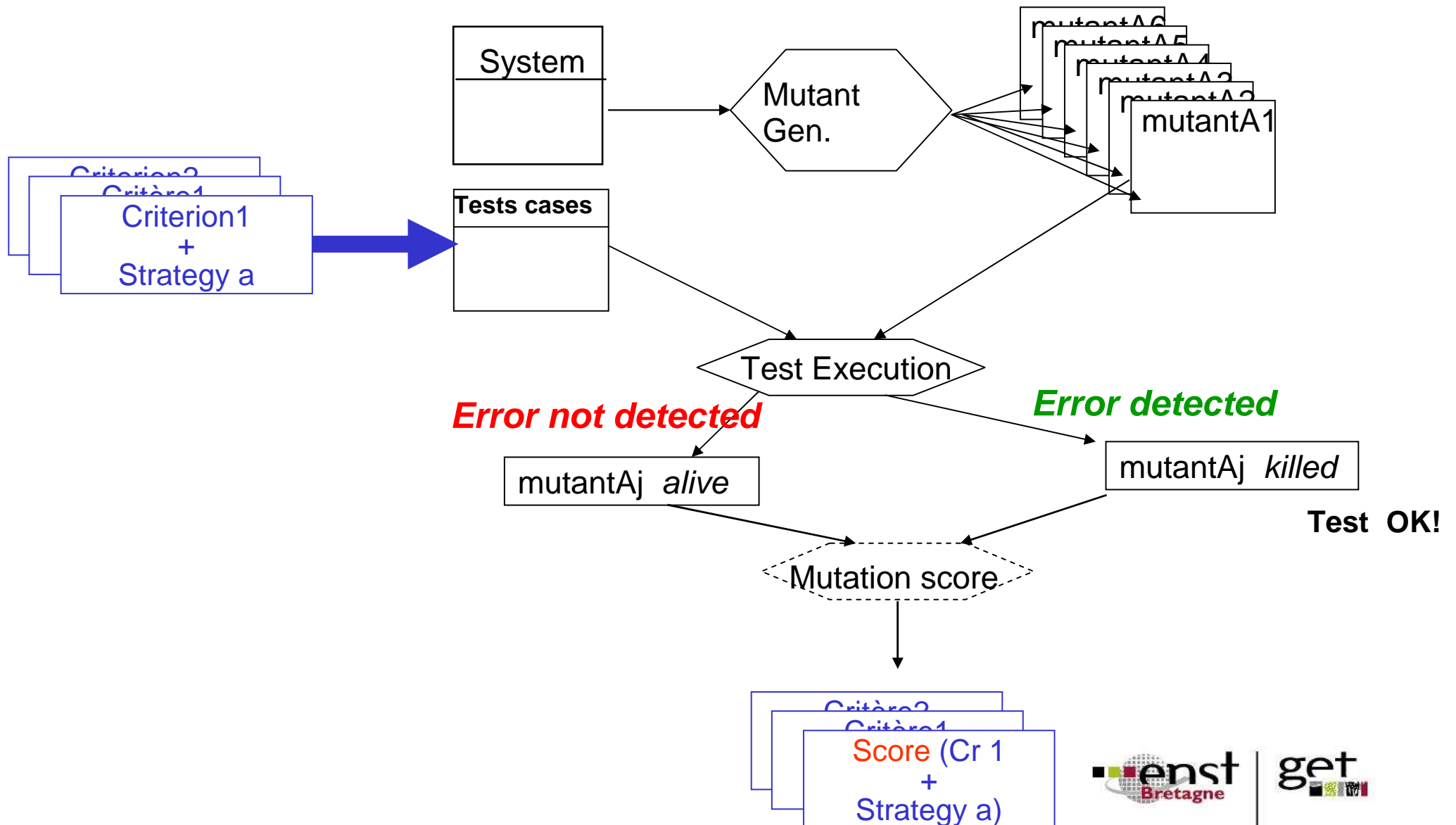
➤ **Advanced SP test cases**

- exercise the default/non specified aspects of the security policy.

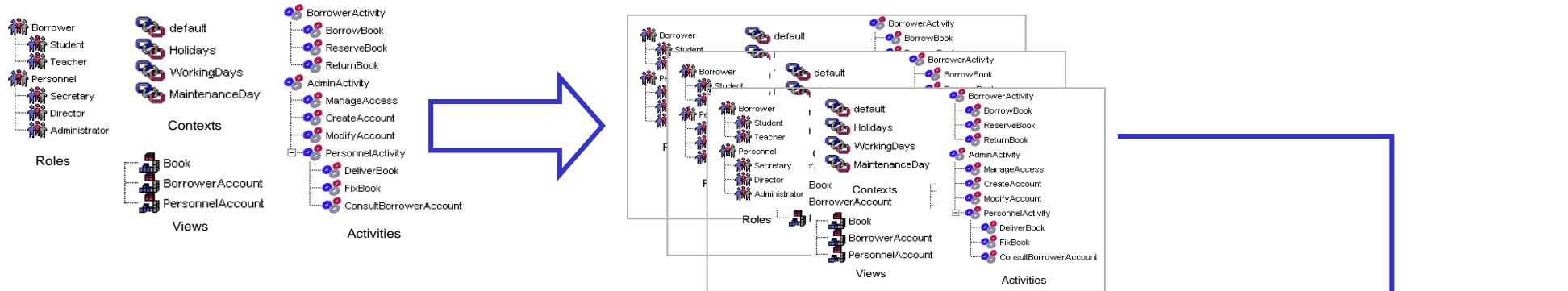
➔ Reusing functional test cases for security purpose ?

- Reusing the test input sequence
- Modifying the intent and the oracle
- *Incremental strategy: It denotes the strategy for producing security test cases which reuse existing test cases.*
- *(in the other case, the strategies are independent)*

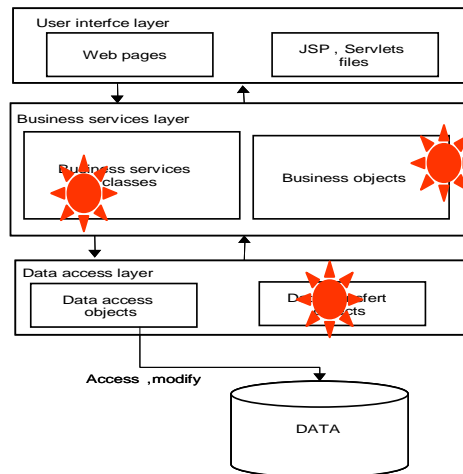
➔ Comparing criteria with mutation analysis



➔ Security policy mutation analysis



Mutants implementation

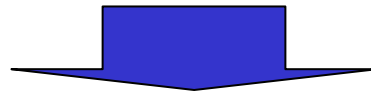


➔ Mutation operators

➤ Simulating access control flaws

- Permission and prohibition – Rule's type errors (PRP-PPR)

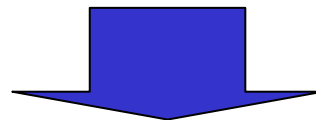
Permission(Library, Administrator, ModifyAccount, BorrowerAccount, WorkingDays)



Prohibition(Library, Administrator, ModifyAccount, BorrowerAccount, WorkingDays)

- Role and context – Parameter errors (RRD-CRD).

Permission(Library, Administrator, ModifyAccount, BorrowerAccount, WorkingDays)

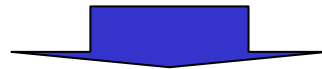


Permission(Library, Teacher, ModifyAccount, BorrowerAccount, WorkingDays)

➔ Mutation operators

- **Hierarchy errors** on roles and activities (RPD ADP)

Permission(Library, Student, BorrowerActivity, Book, WorkingDay)



*Permission(Library, Student, **Reserve**, Book, WorkingDay)*

- **Rule addition** for checking tests robustness (ANR)

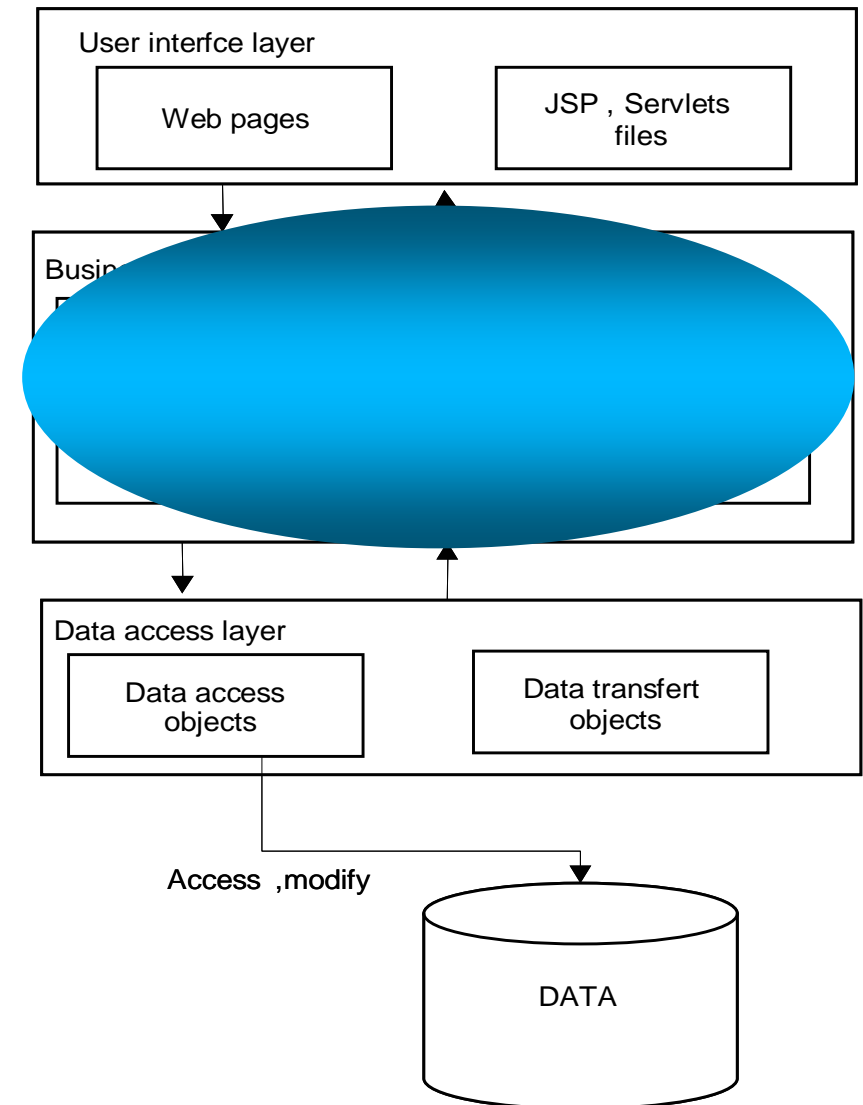


*Prohibition(Library, **Personnel**, reserve, Book, WorkingDay)*

- **Property** : A mutated rule has a higher priority than the other rules

➔ Case studies

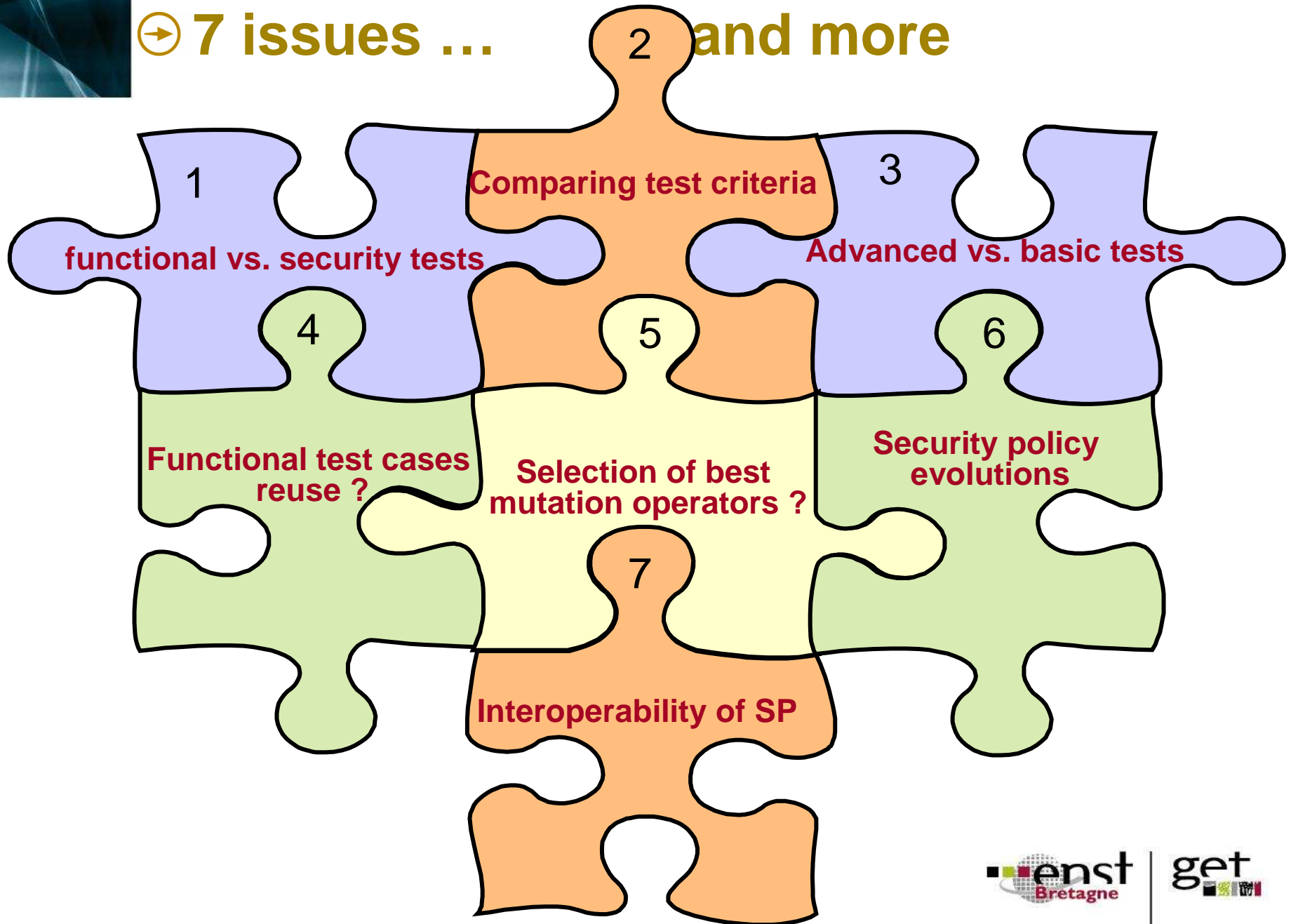
	#classes	#methods	#LOC
LMS (Library Management System)	62	335	3204
VMS (Virtual Meeting Server)	134	581	6077
ASMS (Auction Sale Management System)	122	797	10703



➔ Number of mutants

Operator category		Op.	LMS	ASMS	VMS
Basic Mutation operators	Type changing	PPR	22	89	36
		PRP	19	41	70
	Parameter changing	RRD	60	650	530
		CRD	60	520	318
	Hierarchy changing	RPD	5	20	20
		APD	5	0	20
Rule adding operator		ANR	200	736	432
Total			371	2056	1426

➔ **7 issues ... and more**



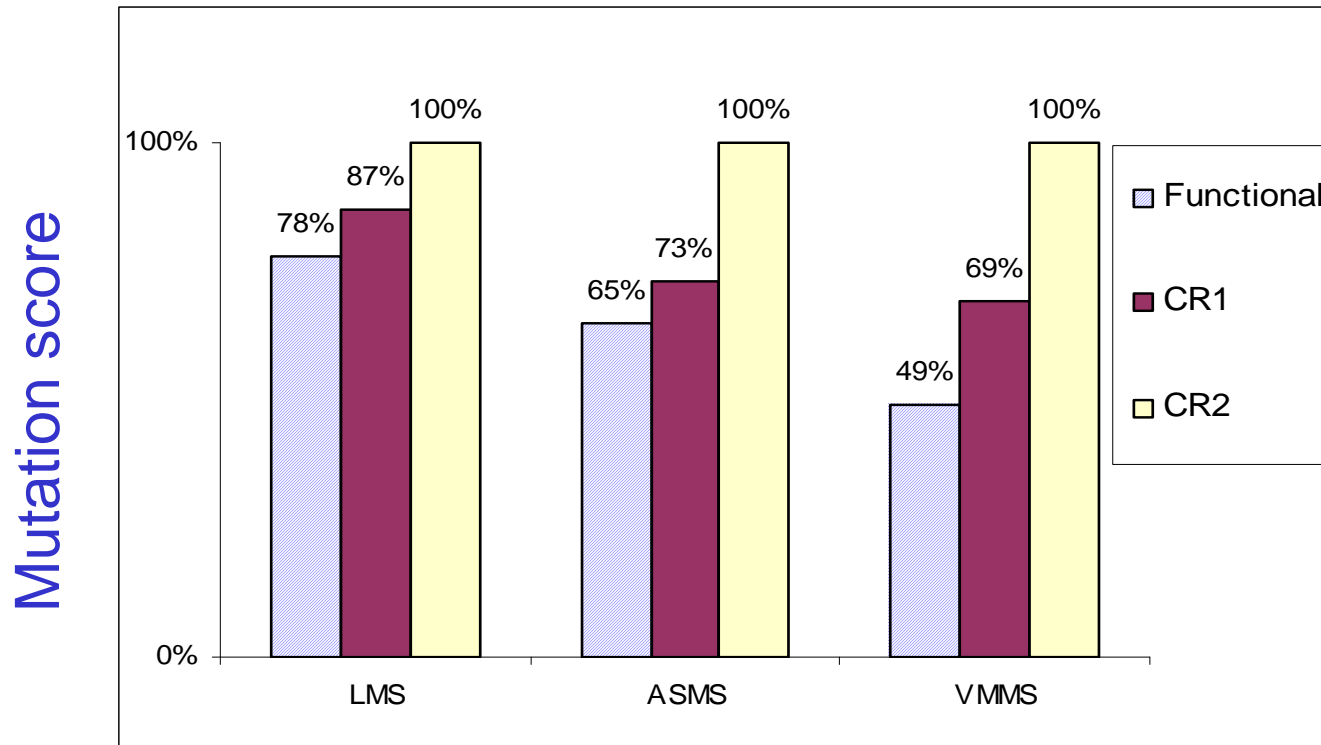


➔ Issues

- **Issue 1: Differences between functional and security tests**
- **Issue 2: Comparing test criteria**
- **Issue 3: Advanced vs. basic security tests**
- **Issue 4: Reusing functional test cases ?**
- **Issue 5: Towards selective mutation**
- **Issue 6: Dealing with security policy evolutions**
- **Issue 7: Interoperability of SP**

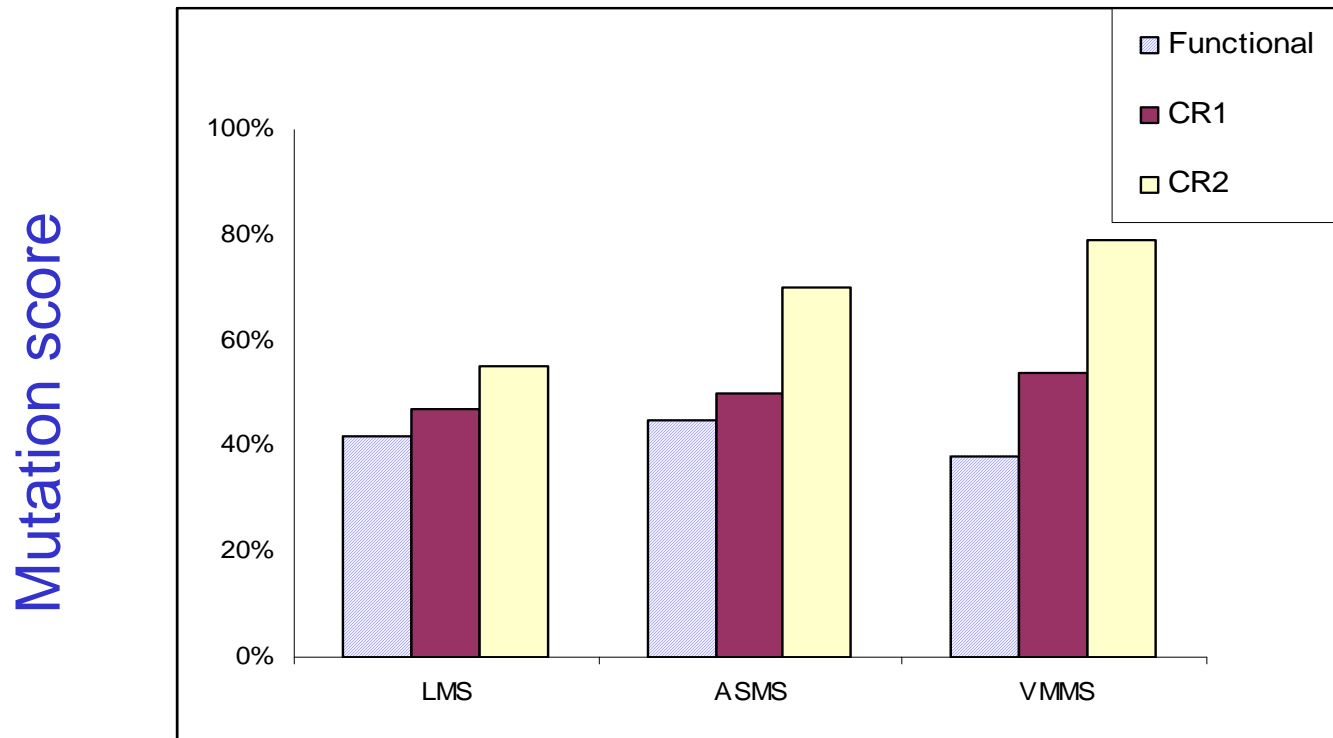


Results



Overall mutation scores with basic security mutants

➔ Results



Mutation scores with all mutants

➔ First issues

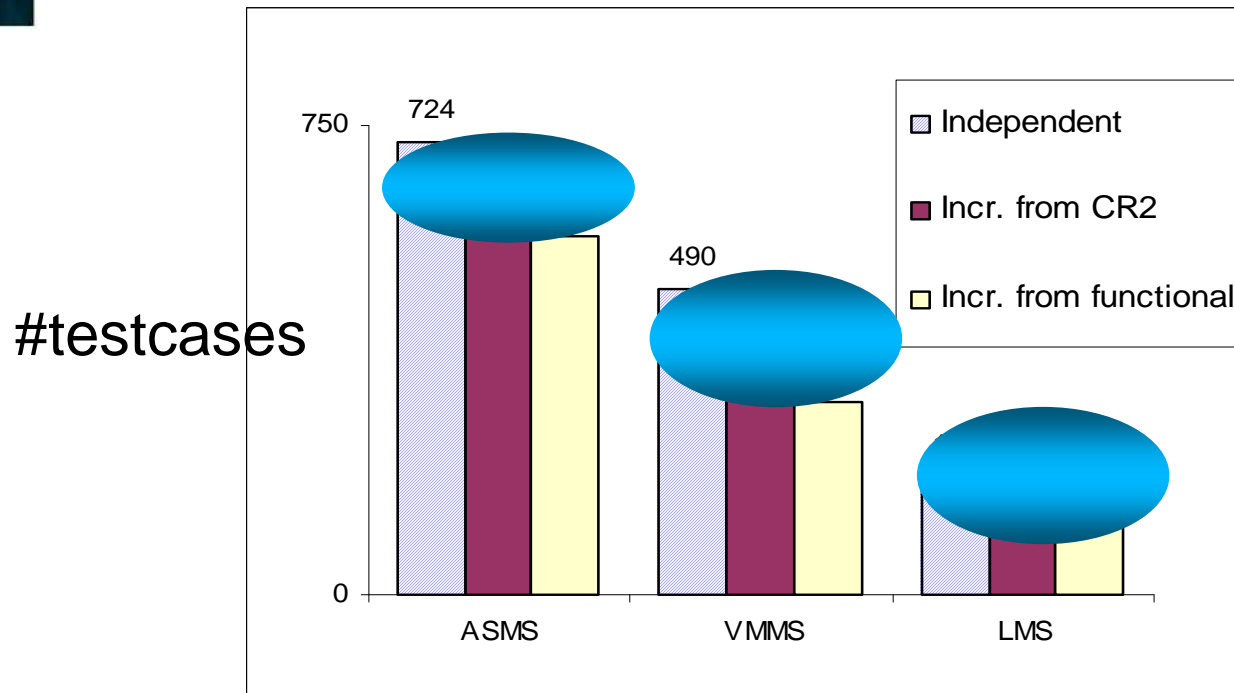
- **Issue 1: Relationships/differences between functional and security tests**
 - For LMS
 - 78% of basic security mutants
 - 11% of ANR
 - 7 of the 42 test cases do not trigger security mechanisms
 - Functional tests \leftrightarrow Security Policy tests
- **Issue 2: Comparing test criteria**
 - $CR2 > CR1$:
 - 100% MS with basic mutation operators
 - None of these criteria are sufficient to kill ANR mutants

➔ Issue 3: Advanced vs. basic security tests

		#test cases	Basic security mutants	ANR
LMS	CR2	35	100%	17%
	Adv. tests	154	59%	100%
ASMS	CR2	110	100%	16%
	Adv. tests	614	69%	100%
VMS	CR2	106	100%	32%
	Adv. tests	384	72%	100%

- Advanced security test cases kill all ANR mutants
 - **a costly task**
- But fail in killing all basic security mutants
- Both criteria are thus needed to be fully efficient in testing the security mechanisms

➔ Issue 4: Reusing functional test cases ?



- Incr. from functional : No clear conclusion
- Incr. from CR2 : 15% of effort saved compared to independent test generation

➤ Issue 5: Towards selective mutation



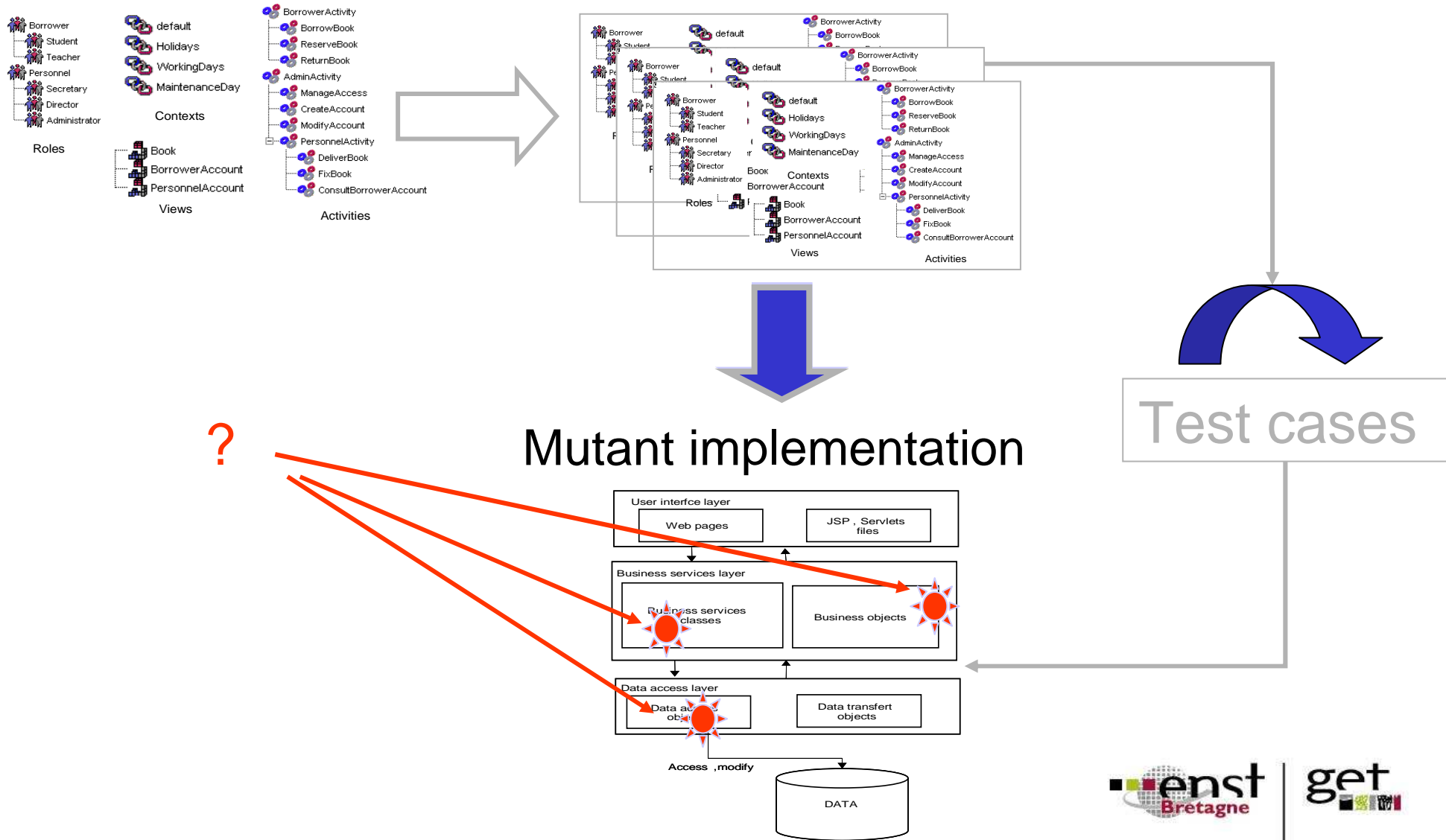
Manual implementation of mutation operators



Test execution

- **Are all operators necessary ?**
- **Is there a « subsumption » relationship between mutation operators ?**

➔ Issue 5: Generating mutants: Tedious !

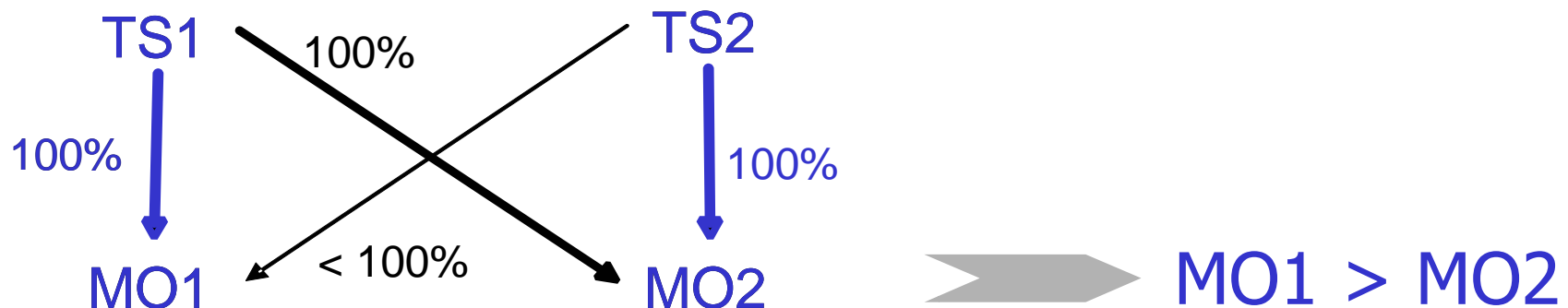


➤ Issue 5: Ranking Mutation Operators

➤ Minimal test suite (TS)

- For a set of mutants
- 100% mutation score
- Removing a test cases makes the mutation score decrease

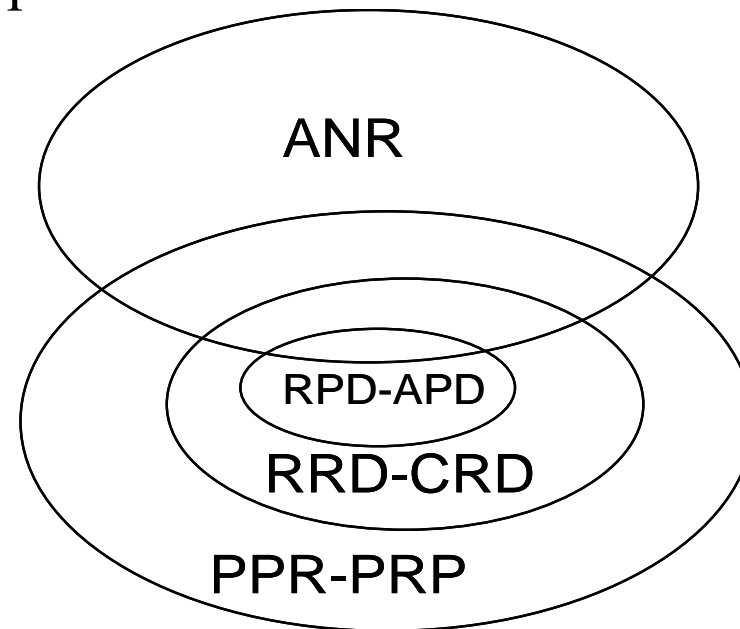
➤ Ranking



➔ Issue 5: Ranking results

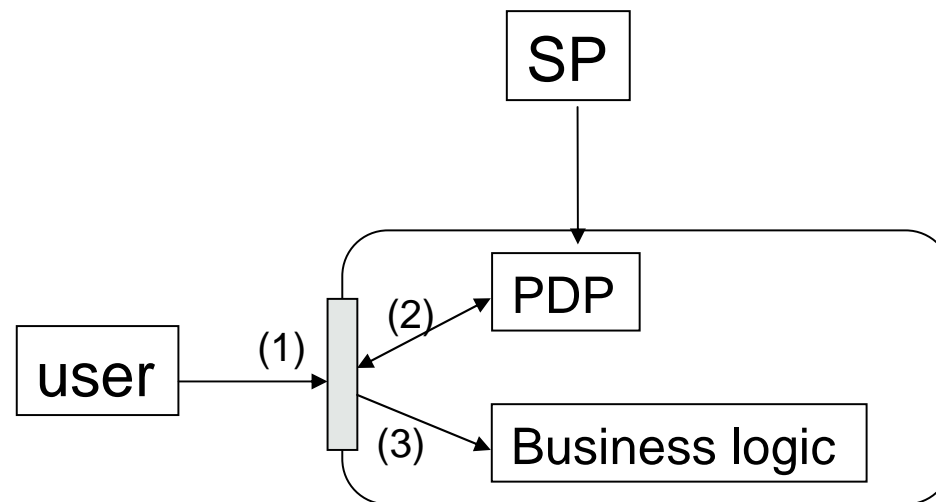
- Ranking
 - PRP (type) subsumes RRĐ (parameter)
 - RRĐ (type) subsumes RPĐ (parameter)
 - ANR (adding new rule) is independent.

- Most useful operators
 - PRP (type), ANR



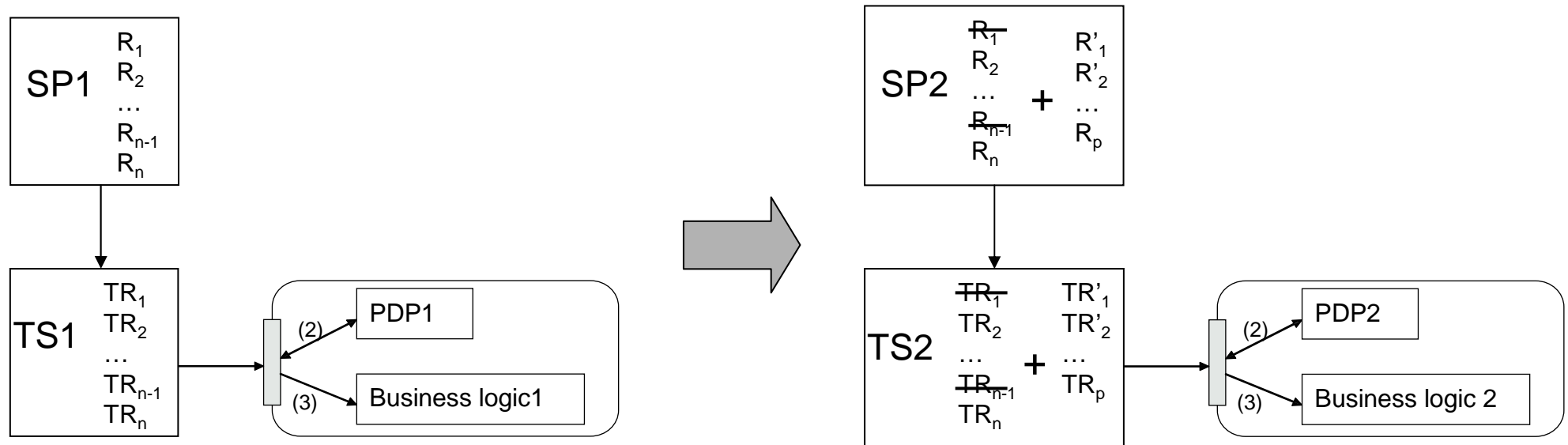
⊕ Issue 6: dealing with security policy evolutions

- Design and architecture
- Conceptual representation in three parts



SP: Security Policy
PDP: Policy Decision Point

➔ Issue 6: dealing with security policy evolutions



Ideal regression testing scheme

SPi: Security Policy i
 TSi : Test suite i
 PDPi: Policy Decision Point i
 TRi: Test rule i

➔ Issue 6: Ideal architecture for a secure system

- **Low coupling between security mechanisms and business objects**
- **Traceability between security requirements and security mechanisms**
- **Separate security components:**
 - Control and correction
 - More evolvable
 - BUT : the business logic design must be flexible

➔ Issue 6: Explicit/hidden security mechanisms

```
public void borrowBook(User user, Book book) throws SecurityPolicyViolationException {
    // call to the security service
    ServiceUtils.checkSecurity(user,
        LibrarySecurityModel.BORROWBOOK_METHOD,
        LibrarySecurityModel.BOOK_VIEW,
        ContextManager.getTemporalContext());
    // call to business objects
    // borrow the book for the user
    book.execute(Book.BORROW, user);
    // call the dao class to update the DB
    bookDAO.insertBorrow(userDTO, bookDTO);}

```

explicit

hidden

```
Public void borrowBook(Book b, User user) {
    // visible mechanism, call to the security policy service
    SecurityPolicyService.check(user,
        SecurityModel.BORROW_METHOD, Book.class, SecurityModel.DEFAULT_CONTEXT);
    // do something else
    // hidden mechanism

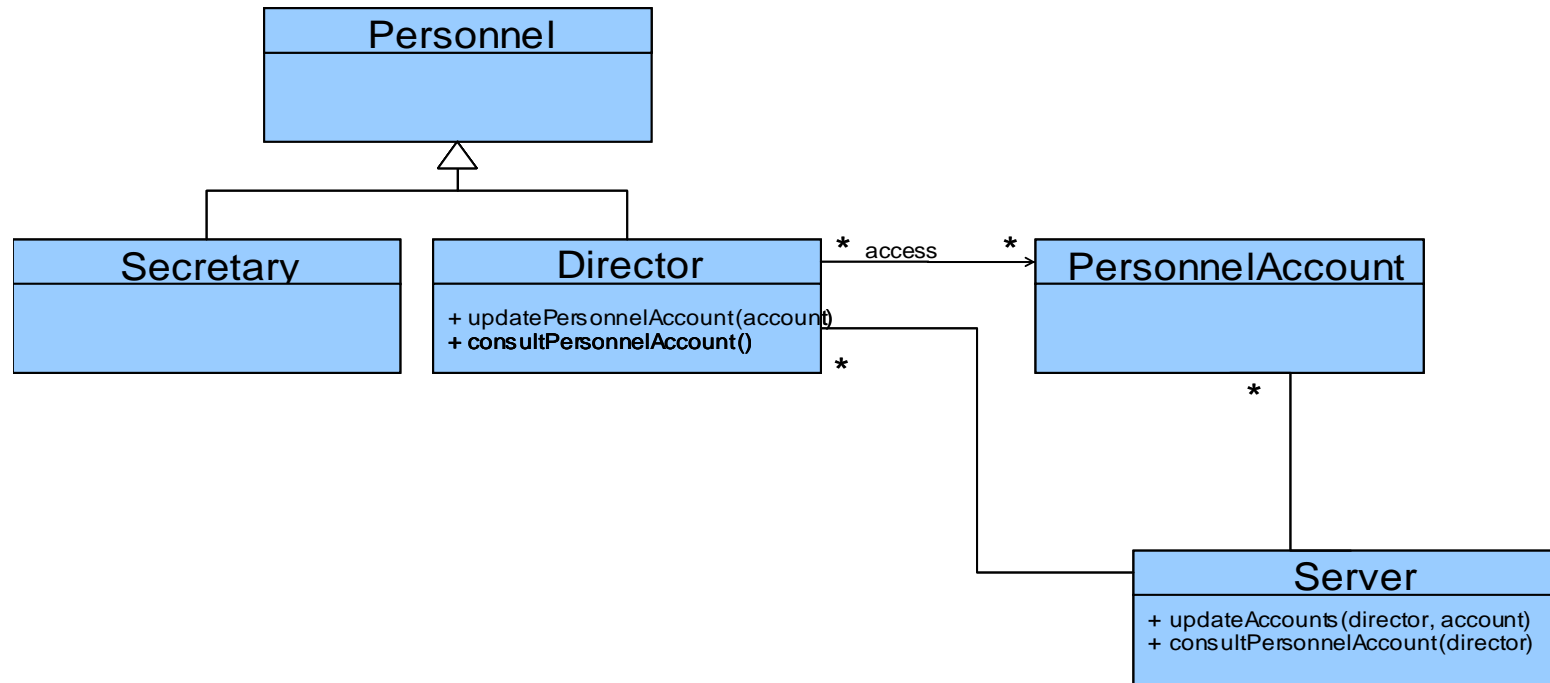
```

```
If(getDayOfWeek().equals("Sunday") || getDayOfWeek().equals("Saturday")) {
    // this is not authorized throw a business exception
    Throw new BusinessException("Not allowed to borrow in week-ends);} ...}

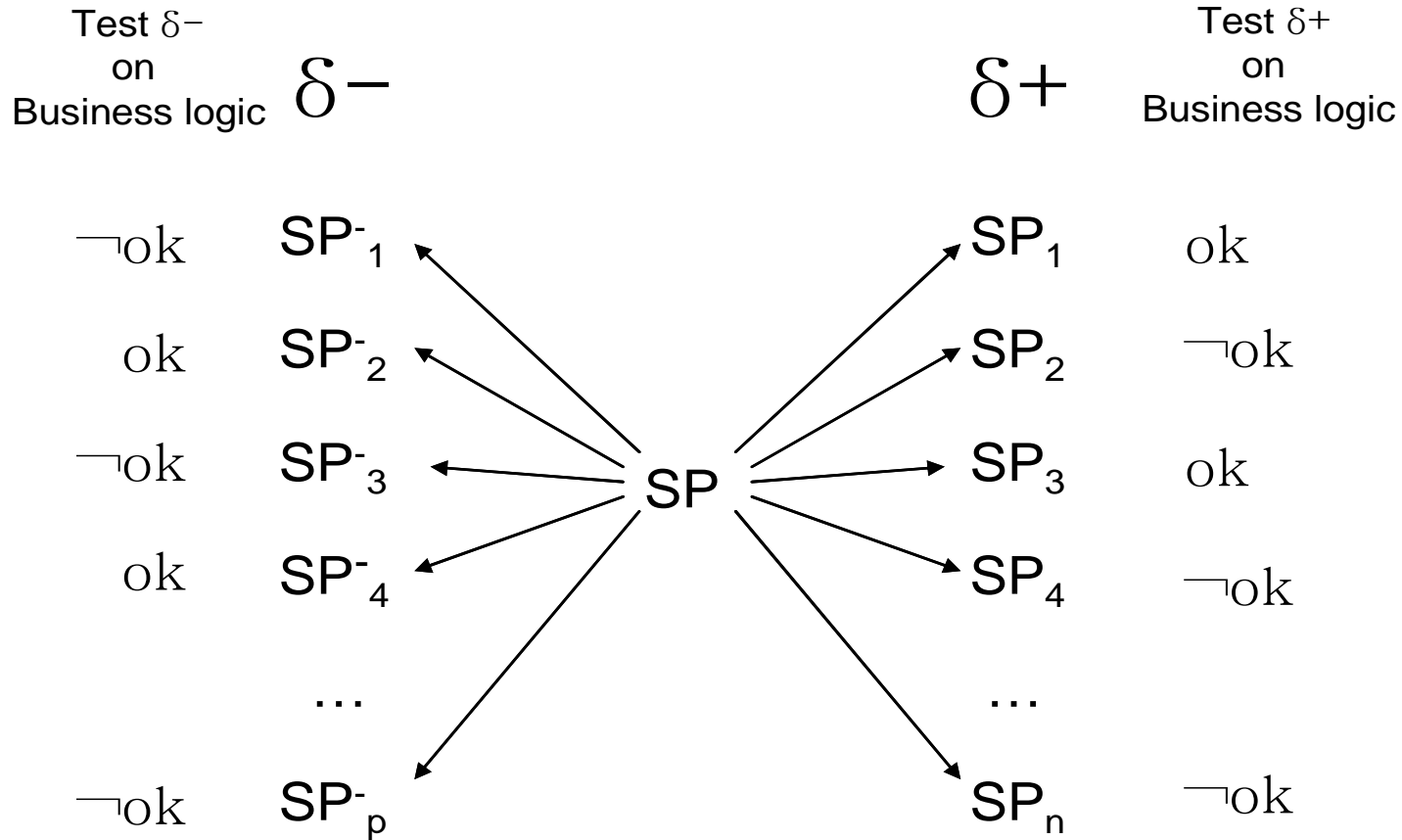
```

➔ Issue 6: Explicit/hidden security mechanisms

Hidden design constraint

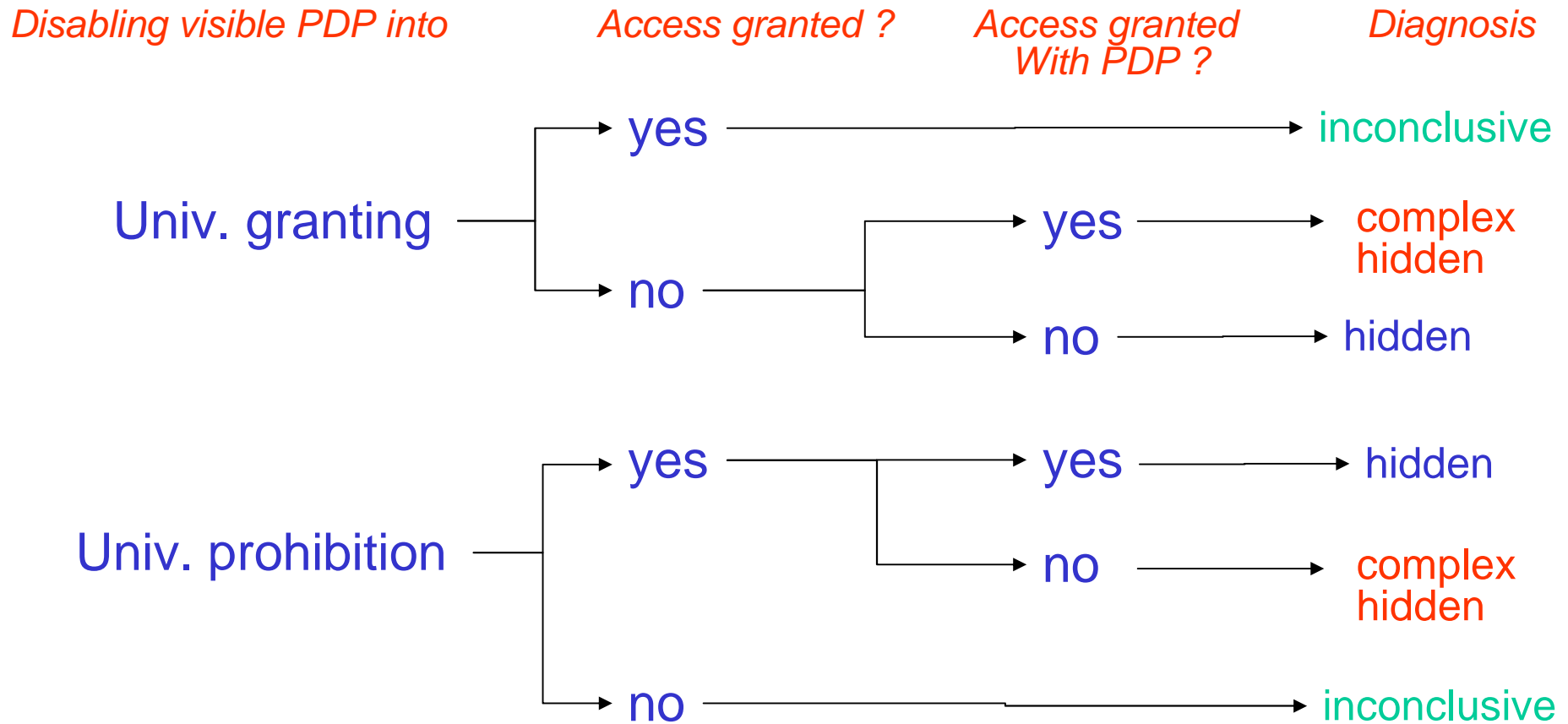


➔ Issue 6: Micro-evolutions of a legacy system access control policy



δ^- = restrict the access control
 δ^+ = relax the access control

➔ Analysis of the test-driven evolution





➔ Methodology

for each micro evolutions :

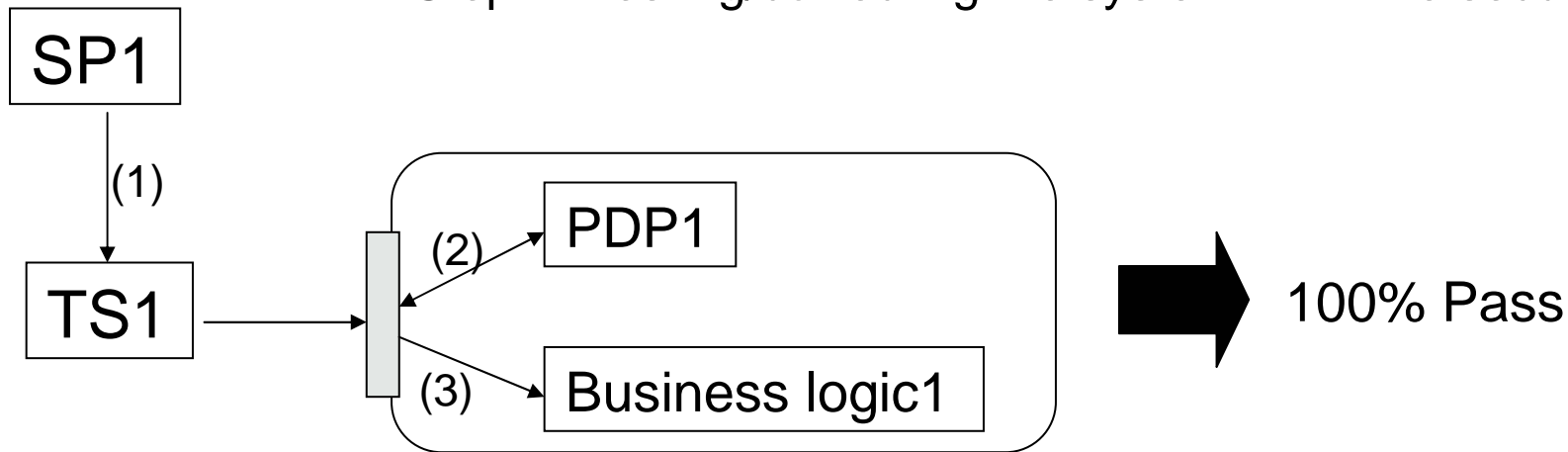
- 1) the associated visible mechanisms are disabled,
- 2) a test case is generated and applied on the business logic,
- 3) if the test case fails (detects an error), it means that the micro-evolution cannot be supported by the existing business logic without analysis and refactoring.

$$flexibility(SP, TSP) = \frac{\left| \{tsp \in TSP^+ / tsp(pass)\} \cup \{tsp \in TSP^- / tsp(pass)\} \right|}{\left| \bigcup_{i=1..n} SP_i^+ \right| + \left| \bigcup_{j=1..p} SP_j^- \right|}$$

$$rigidity = 1 - flexibility$$

➔ Issue 6: Test driven audit of the current system

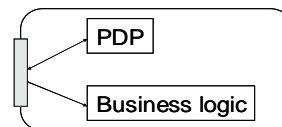
Step 1 : testing/correcting the system w.r.t. the security policy



SPi: Security Policy i

TSi : Test suite i

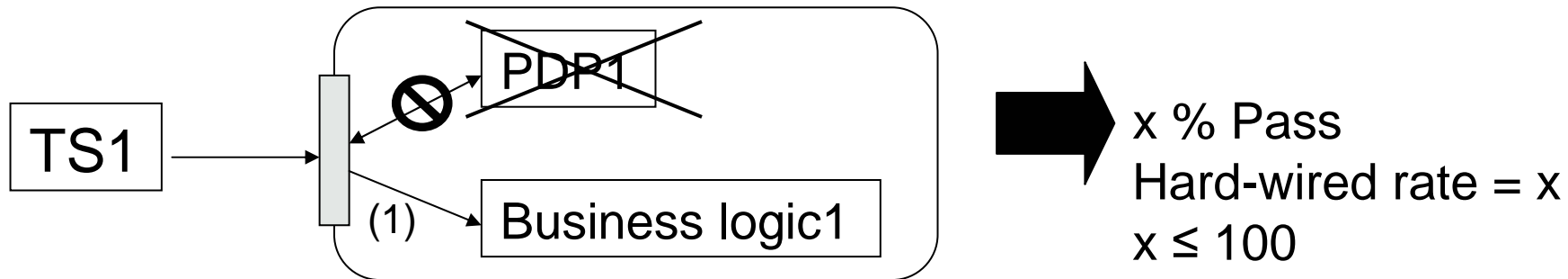
PDPi: Policy Decision Point i



: a legacy system

➔ Issue 6: Test driven audit of the current system

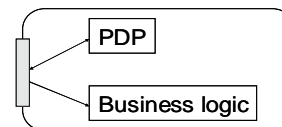
Step 2 : reapply test cases to check the security policy « hard-coding » rate



SPI: Security Policy i

TSi : Test suite i

PDPi: Policy Decision Point i



: a legacy system

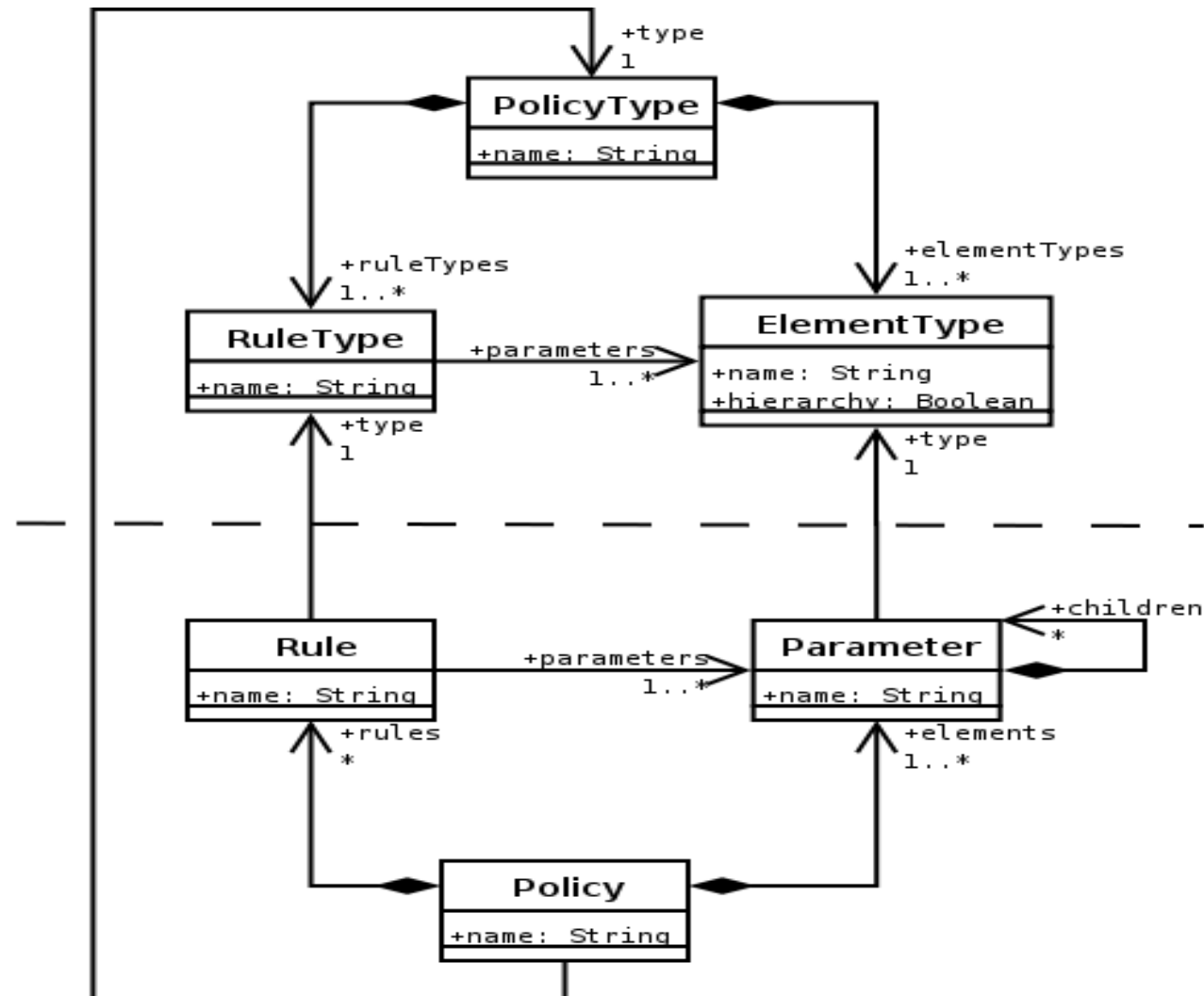
➔ Issue 6: Flexibility of the VMS case study

	Flexible	Rigid rules	System flexibility
results	20	36	0.35

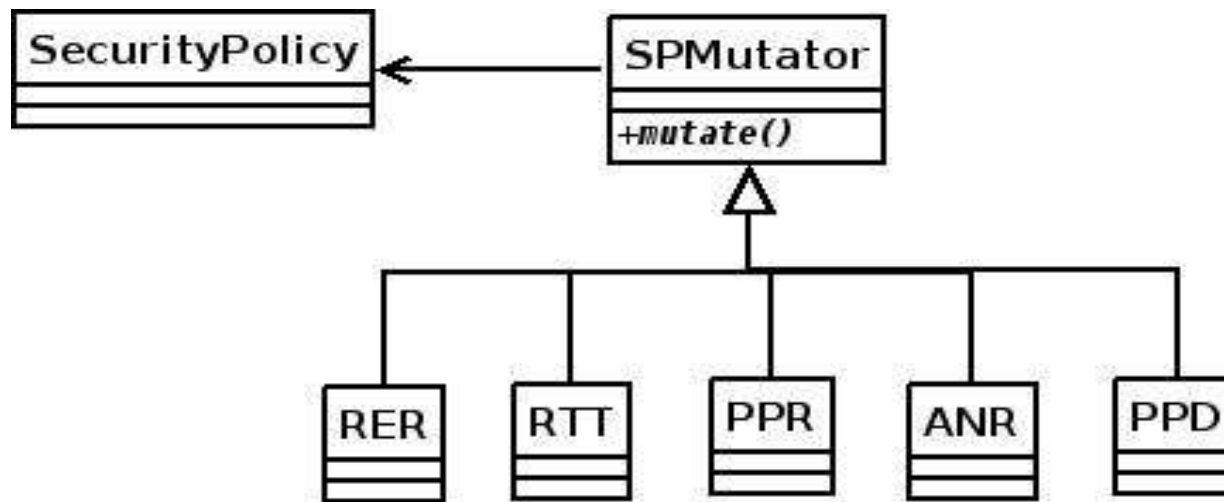
Resource\View	Flex. rules	Hard-wired rules	All	Flexibility
Meeting	12	36	48	0.25
PersonnelAccount	6	0	6	1
UserAccount	2	0	2	1

Function/Activity	Flexibility
updatePersonnelAccount	1
updateUserAccount	1
askToSpeak	0.13
leaveMeeting	0.14
overSpeaking	1
closeMeeting	1
setMeetingAgenda	0.14
setMeetingModerator	0.14
speakInMeeting	0.14
setMeetingTitle	0.14
deleteUserAccount	1
openMeeting	1
handOver	1
deletePersonnelAccount	1

➔ Issue 7: Interoperability of SP A MDE framework for security policies



➔ Application to mutation testing





➔ Generic mutation operators

Operator Name	Definition
RTT	Rule type is replaced with another one
PPR	Replaces one rule parameter with a different one
ANR	Adds a new rule
RER	Removes an existing rule
PPD	Replaces a parameter with one of its descending parameters

➤ Implementation in Kermeta

```
class RER inherits SPMutator {  
  
  method mutate(p : Policy) : set Policy[*] is do  
    var mutant : Policy  
    result := Set<Policy>.new  
    // loop on rules  
    p.rules.each{ r |  
      // create mutated policy  
      mutant := p.copy  
      mutant.name := p.name + "-RER-" + r.name  
      // remove one rule  
      mutant.rules.remove(mutant.rules.detect{x | x.name == r.name})  
      // adds the mutant policy  
      result.add(mutant)  
    }  
  end  
}
```

➤ Implementation in Kermeta

```

class PPR inherits SPMutator {

  method mutate(p : Policy) : set Policy[*] is do
    var mutant : Policy
    result := Set<Policy>.new
    // loop on rules
    p.rules.each{ rule |
      // loop on paramters
      rule.parameters.each{ param |
        // select a diffrent parameter having the same type
        p.elements.select{ e | e != param and e.type == param.type}.each { e |
          mutant := p.copy
          mutant.name := p.name + "-PPR-" + rule.name + "-" + param.name + "-" + e.name
          var r : Rule init mutant.rules.detect{ x | x.name == rule.name }
          // the rule has now a diffrent parameter
          r.replaceParameter(r.parameters.detect{ u | u.name == param.name}, e)
          result.add(mutant)
        }
      }
    }
  }
end
}

```

➔ Outputs

POLICY LibraryOrBAC (OrBAC)

R1 -> Permission(Library Student Borrow Book WorkingDays)

R2 -> Prohibition(Library Student Borrow Book Hollydays)

R3 -> Prohibition(Library Secretary Borrow Book Default)

R4 -> Permission(Library Personnel ModifyAccount UserAccount WorkingDays)

R5 -> Permission(Library Director CreateAccount UserAccount WorkingDays)

Examples of mutants:

RER Mutant

POLICY LibraryOrBAC-RER-R1 (OrBAC)

R2 -> Prohibition(Library Student Borrow Book Hollydays)

R3 -> Prohibition(Library Secretary Borrow Book Default)

R4 -> Permission(Library Personnel ModifyAccount UserAccount WorkingDays)

R5 -> Permission(Library Director CreateAccount UserAccount WorkingDays)

RTT mutant

POLICY LibraryOrBAC-RTS-R4-Prohibition (OrBAC)

R1 -> Permission(Library Student Borrow Book WorkingDays)

R2 -> Prohibition(Library Student Borrow Book Hollydays)

R3 -> Prohibition(Library Secretary Borrow Book Default)

R4 -> Prohibition(Library Personnel ModifyAccount UserAccount WorkingDays)

R5 -> Permission(Library Director CreateAccount UserAccount WorkingDays)

➔ Outputs

POLICY LibraryRBAC (RBAC)

R1 -> UserRole(romain Student)

R2 -> UserRole(yves Director)

R3 -> UserRole(alice Secretary)

R4 -> RolePermission(Student BorrowBook WorkingDays)

R5 -> RolePermission(Personnel ModifyUserAccount
WorkingDays)

R6 -> RolePermission(Director CreateAccount AllTime)

Here are some examples of the generated mutants

RER mutant:

POLICY LibraryRBAC-RER-R5 (RBAC)

R1 -> UserRole(romain Student)

R2 -> UserRole(yves Director)

R3 -> UserRole(alice Secretary)

R4 -> RolePermission(Student BorrowBook WorkingDays)

R6 -> RolePermission(Director CreateAccount AllTime)

PPR mutant:

POLICY LibraryRBAC-RDD-R1-Student-Personnel (RBAC)

R1 -> UserRole(romain Personnel)

R2 -> UserRole(yves Director)

R3 -> UserRole(alice Secretary)

R4 -> RolePermission(Student BorrowBook WorkingDays)

R5 -> RolePermission(Personnel ModifyUserAccount
WorkingDays)

R6 -> RolePermission(Director CreateAccount AllTime)

➔ Conclusion

- **A first result :**
 - specific testing techniques are needed to guarantee the coverage of access control aspects
- **A qualification process**
- **Promising results about system SP evolution**
 - Flexibility analysis
 - Detecting hidden mechanisms
 - Test driven evolution

➤ Other works in progress

- **More complex aspects**
 - Not only binary
- **Security contracts for IS vigilance**
 - 'Design by' security contracts
 - Component based approach
 - AOP
- **MDE framework for SP interoperability**
 - Automatic synthesis of PDP
 - System evolutions
- **Code injection counter-measures**
- **SOA security testing and software IDS inference**



➔ **Thank you !**

Questions ?