

Test Statistique Structurel et Fonctionnel

Pascale Thévenod-Fosse, Hélène Waeselynck
{thevenod,waeselyn}@laas.fr



Journée Club SEE
"Systèmes informatiques de confiance"
Thème : Test
Paris, le 1er juin 1999

Probabilistic Generation of Input Patterns



Random testing:

uniform distribution over the input domain

→ "Blind" approach



Statistical operational testing:

simulated operational profile

→ reliability estimation



Statistical structural / functional testing:

input distribution determined according to a test criterion

→ proper coverage of the software model

Focus: fault-finding

Outline

- **Statistical structural / functional testing: Why?**
- **Statistical structural / functional testing: How?**
- **Some results from industrial case studies**
 - ➔ *Procedural programs (C, Pascal, Assembler):*
Statistical structural and functional testing versus random testing and deterministic structural testing
 - ➔ *Synchronous data flow programs (Lustre):*
Mixed test strategy (statistical + deterministic)
- **Conclusion and On-Going Work**

No Software Design Fault Model

criterion $C \rightarrow S_c = \{ \text{elements } k \text{ to be exercised during testing} \}$

Usual Practice

deterministic test set such that each k is exercised (at least) once

**Imperfect Connection
Criteria Faults**

search for more refined criteria

compensate for the weakness of C by exercising each k several times

Statistical Testing According to C

Why?

Statistical Testing

criterion C + random generation

☞ **Balanced input distribution / C** → all the elements k have the same probability of being exercised

Balanced coverage of the software model

☞ **Proper input distribution / C** → maximize the probability of exercising each element k

Proper coverage of the software model

Why?

Design of Statistical Test Sets

Criterion C

$S_c = \{ \text{elements } k \text{ to be exercised} \}$

1

Search for a probability distribution over the input domain



Purpose:
maximize the probability P_k of exercising each k

2

Assessment of the test size N



Stopping rule:
test quality q_N wrt C

$$N = \frac{\log(1-q_N)}{\log(1-P)}$$
$$P = \min \{ P_k \}$$

On average, the least likely k is exercised n times with $n \approx -\ln(1-q_N)$:

$n \approx 7$ for $q_N = 0.999$; $n \approx 9$ for $q_N = 0.9999$

How?

Search for an Input Distribution



Analytical techniques

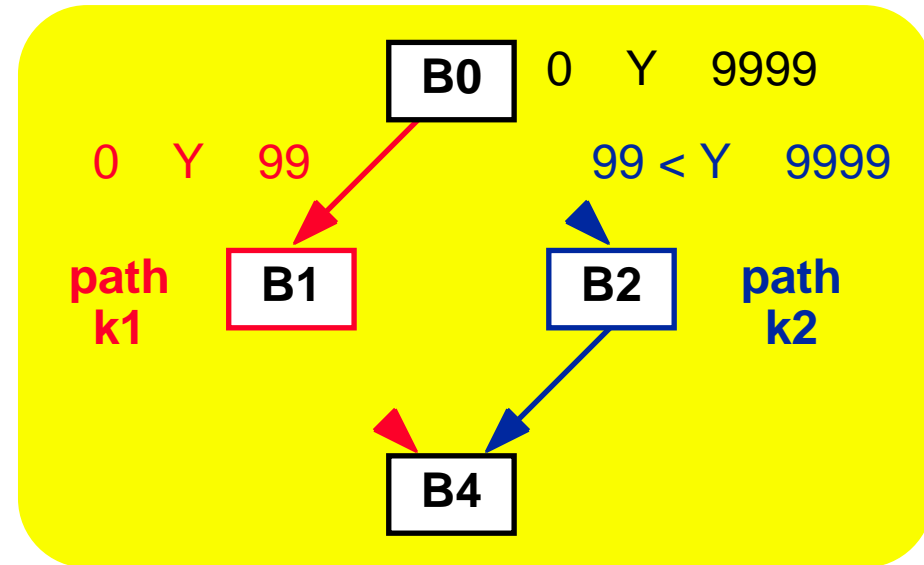
$C = \{\text{paths}\} \rightarrow S_c = \{k1, k2\}$

$p1 = \text{Prob}[0 \ Y \ 99]$

$p2 = \text{Prob}[99 < Y \ 9999]$

$p1 = p2 = 0.5$

Structural distribution / C



Empirical techniques

successive refinements of an initial distribution

How?

Case Studies / French Industry

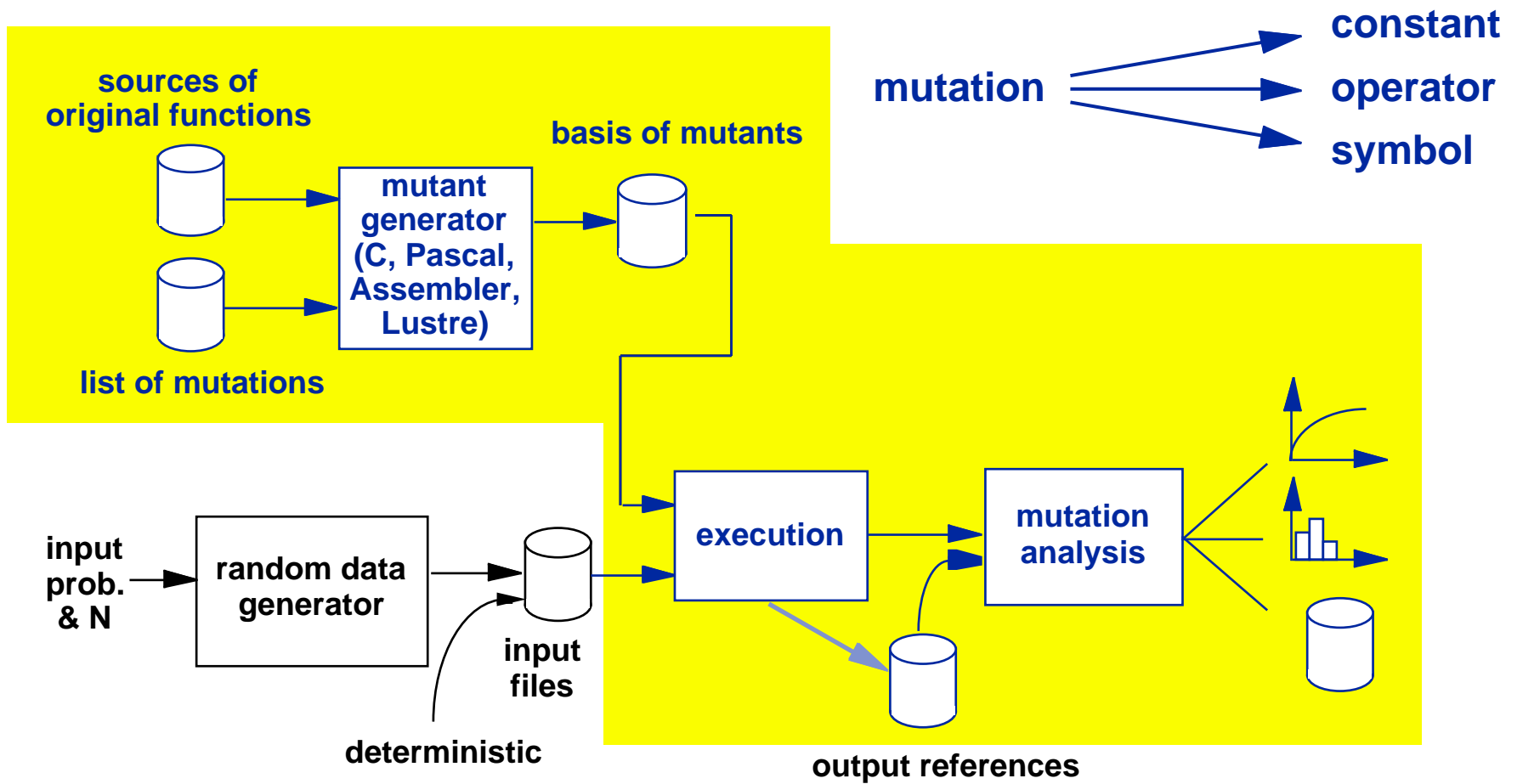
Field	# programs / testing level			Testing techniques
	Unit	Component	Integration	
Nuclear	24	2	11	Statistical structural Deterministic structural Statistical functional Random (uniform)
Avionics	6	–	–	Statistical structural Statistical functional Random
Spatial	–	1	–	Statistical functional Deterministic functional

Languages: C, Pascal,
Assembler,
Lustre

+ industrial test sets
(when available)

Results

Mutation Tool: SESAME

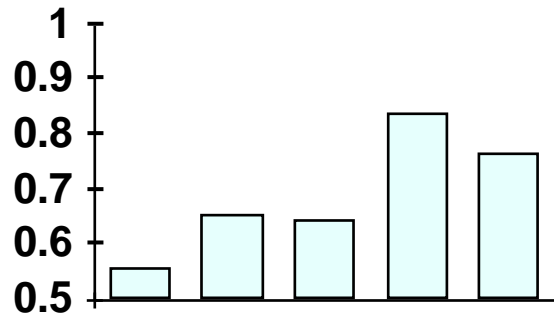


Results

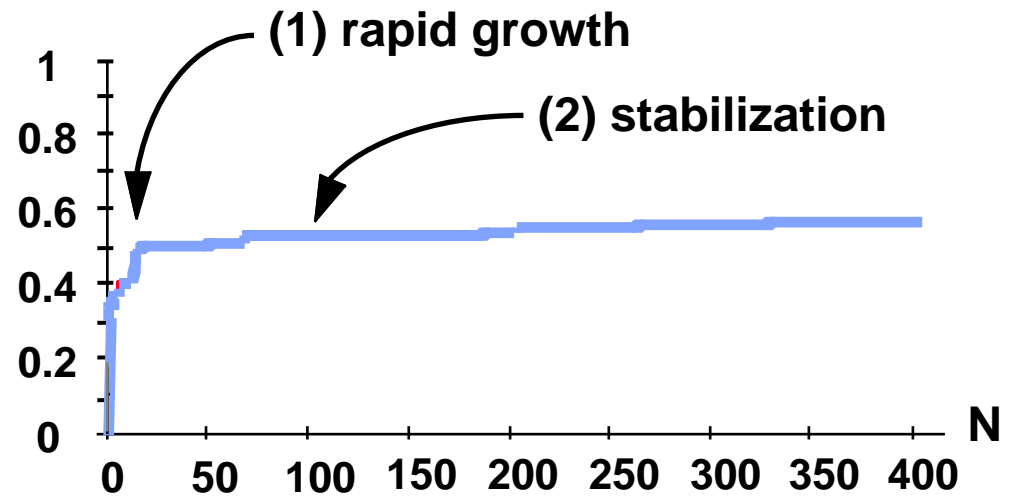
Random Testing: a Poor Approach

FCT3: 1416 mutations, N = 405

% revealed



Five different test sets



Component: 13 genuine faults, N = 5300

● 5 faults revealed

✗ 8 faults not revealed

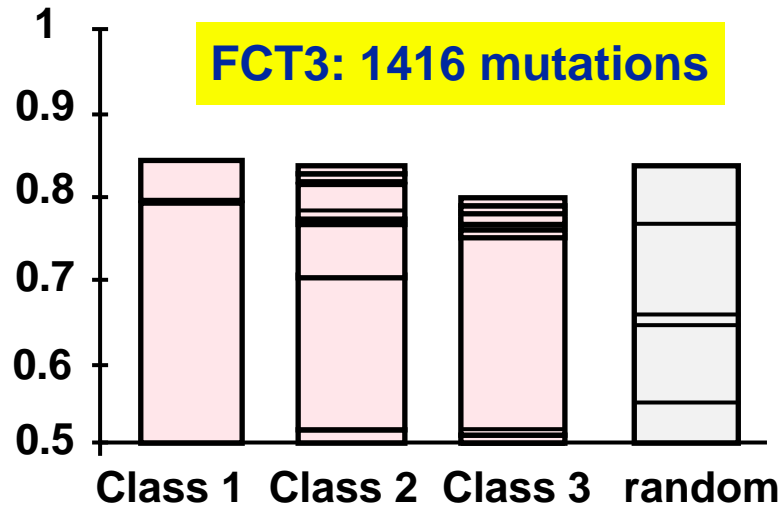
A	B	C	D	E	F	G	H	I	J	K	L	Z
●	✗	✗	●	✗	●	●	●	✗	✗	✗	✗	✗

No new fault revealed after N = 633

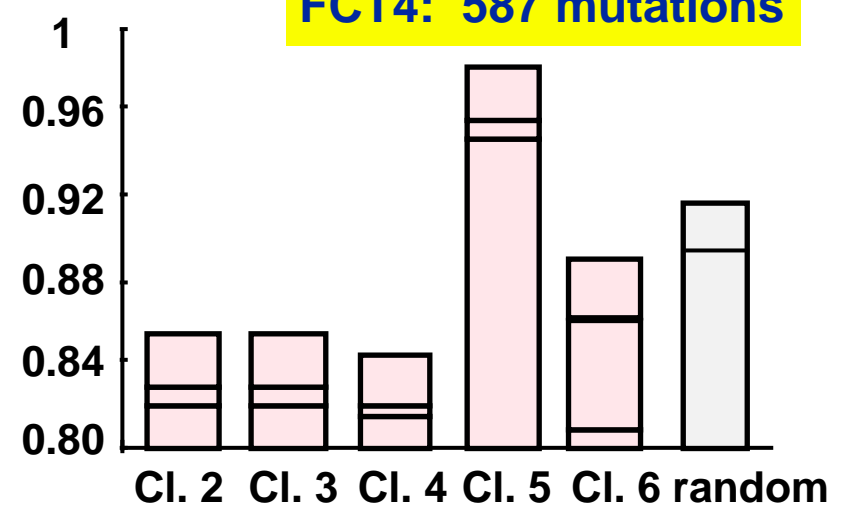
Results

Deterministic Structural Testing

% revealed



% revealed



- All-Paths (Class 1): score < 100%
- Strong impact of the input values chosen to cover the criteria
- Criterion stringency → no guarantee
- Not significantly better than random testing

Results

Statistical Structural Testing

4 programs FCTi 2816 mutations

Test data sets	# Mutations not revealed	% Mutations revealed
Deterministic structural	292 — 825	70.7% — 89.6%
Random (uniform)	278 — 687	75.6% — 90.1%
Statistical structural	5 — 49	98.3% — 99.8%

statistical
structural testing
+
deterministic testing
of extremal values

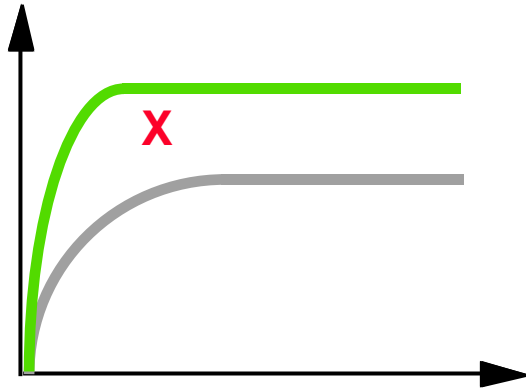
Cost-effective design:

- adopt weak test criteria (e.g. branches)
- require high test quality (e.g. $q_N = 0.9999$)

Results

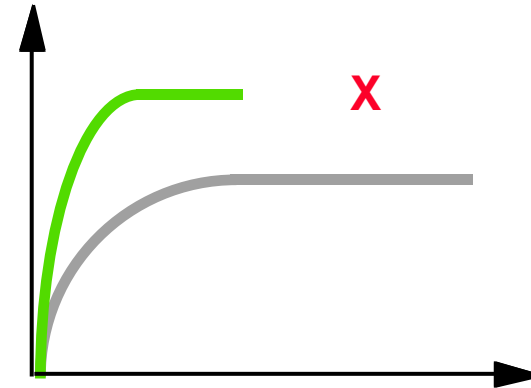
Comparison / Industrial Practice

% mutations revealed



Size of the test sets

% mutations revealed



Size of the test sets

**Safety critical software
(unit testing)**

X deterministic (industrial)

— statistical structural

— Random uniform

Results

Statistical Functional Testing



Behavior models

- Finite state machines, Decision tables (SA/RT)
- Statecharts
- etc



Hierarchical modeling (top-down)



Complexity

- weak criteria (e.g. state coverage)
- several input distributions, each one focusing on the coverage of a subset of functional hierarchy

Results

Component: 13 genuine faults

Student version: 12 faults (A, ..., L)

Industrial version: 1 fault Z (to be compensated by hardware checks)

- structural: A, G, J
- misunderstanding: B, ..., F, I
- initialization: H, K, L, Z

1 random test set
(N = 5300)

SA/RT: 5 test sets
(N = 441)

Statecharts: 5 test sets
(N = 441)

	A	B	C	D	E	F	G	H	I	J	K	L	Z
1 random test set	●	×	×	●	×	●	●	●	×	×	×	×	×
SA/RT: 5 test sets	●	●	●	●	●	●	●	●	●	●	●	×	④
Statecharts: 5 test sets	●	●	●	●	●	●	●	●	●	●	●	●	●

● always revealed

④ revealed by i sets out of 5

× not revealed

Results

Conclusion / Procedural Programs

Mixed test strategy



Statistical testing = criterion C + random generation

Proper input distribution,
sound probe of the software model

Compensate for the weakness of C,
automatic generation of large test sets



Deterministic testing = extremal/special values + selective choice

identified from the software model
(structural or functional)

→ Boundary values
→ Transient modes
(e.g. initialization)

Results

Mixed Strategy for Lustre Programs

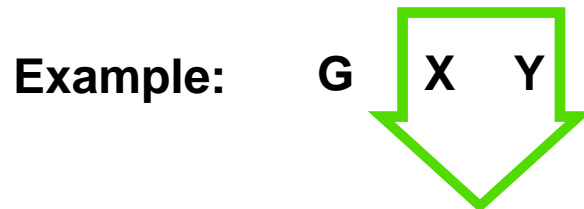
Statistical structural testing

Each transition one cyclic execution

- Test criterion: **All-transitions**, except for the initialization operating mode

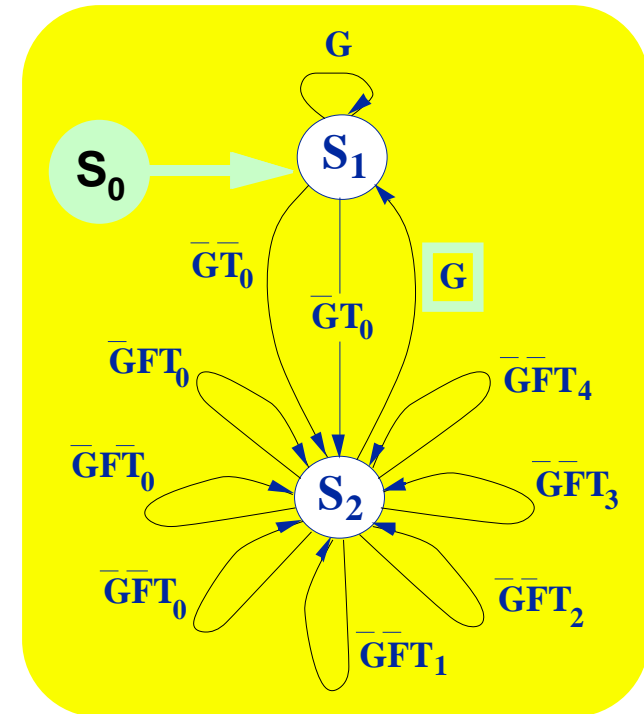
Deterministic testing

- Initial state S_0 and its output transition(s)
- Boundary values / transition conditions



$X = Y$ is a boundary value

Automaton produced by the Lustre compiler



Example: Node ctrl

Results

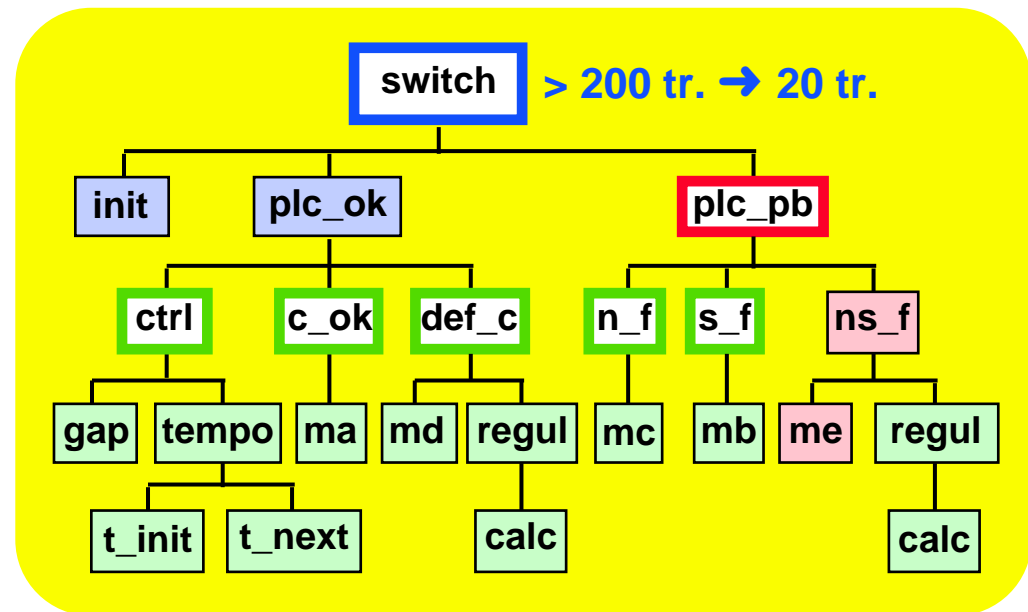
Unit and Integration Testing Levels

Principle:

- Bottom-up exploration of the program call graph
- Integration levels: simplified automata

Example: SWITCH
21 different nodes → 3 levels

- unit testing → □
- 1st integration testing → □
- 2nd integration testing → □



Results

Effectiveness of the Mixed Strategy

Nodes	Other nodes tested	# Mutations	# Mutations not revealed	
			Statistical	Stat.+Deter.
ctrl	gap, tempo, t_init, t_next	311	4	0
c_ok	ma	503	0	—
def_c	md, regul, calc	493	3 – 8	0 – 2
n_f	mc	506	0 – 1	—
s_f	mb	504	0 – 1	—
plc_pb	ns_f, me	317	0 – 10	—
switch	init, plc_ok	173	7 – 8	0 – 3

- High error detection power of statistical testing
- Deterministic test inputs are complementary
- But, at the top level, deterministic testing does not provide a sound probe of the initialization process
 - ↳ statistical testing based on a specific test profile

Results

Conclusion



**Criterion analysis for the input profile:
complexity deterministic testing**

➔ weak criterion + empirical search tractable



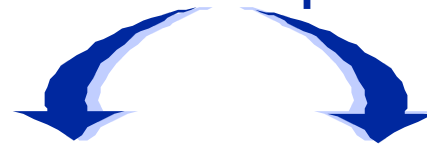
Automatic generation of test inputs



High fault revealing power



Separate testing of extremal and special values



**deterministic
testing**

**statistical testing with
specific test profile(s)**



Large test sets → oracle problem

how to determine the correctness of the outputs?

➔ executable specification or prototyping facilities

Conclusion

Challenging Issues: OO Paradigm

- **Basic modular unit: class**
 - ➔ only objects (class instances) can be executed ... and tested
- **Structure**
 - ➔ distributed among several classes
- **Behavior**
 - ➔ complex dependencies between objects
 - ➔ highly dependent on the object states

- ① How to define the testing levels (unit, integration, ...)?
- ② Which software models & associated test criteria?



On-going work