A natural measure for software system complexity

CSDM 2010 – Parallel session 2

Jacques Printz, Professor Emeritus, Cnam

A pragmatic definition of complexity
Project management point of view

- **System A is more complex than system B if the cost to develop, maintain and operate A is greater than the cost of B**
  - **Development** = Programming + Test (Validation and Verification) + Documentation (Users and Maintenance)
  - **Maintenance** = Programming (modification of an existing code) + Test (Non regression) + Documentation (Update)
  - **Operate** = Capacity planning and system administration, in particular ⇒ Ways to recover a coherent state after a fault
    - ✓ Maintain essential data to reconstruct a coherent state of the system
    - ✓ See Autonomic computing approach and the notion of autonomic component

Physical / Mathematical analogy

- **Any program affirms something about the validity of the transformation of an input state into an output state**
  - It works like a physical law or, in some limited cases, like a mathematical theorem
    - ✓ States are related to **information** stored and managed by the IS
- **Tests of the program are like a kind of proof**
  - Experimental proof (scenarios, experiments) like in physics
    - ✓ For example the CERN LHC (4.5 Md€) for the Higg's boson
  - Formal proof (deduction from axioms or models) like in mathematics
    - ✓ For example, around 300 pages for the demonstration of Fermat's last theorem (by A.Wiles) → For a one line assertion

Measure of system complexity using tests

Tests take in account

- the **statically** aspect of programming
  - Program flow graph, coverage measurement, number of instructions, …
- the **dynamically** aspect of data transformations and control
  - Data dependencies and functional dependencies, shared data, events, ACID transactions (i.e. modules, like in the definition given by D.L.Parnas)

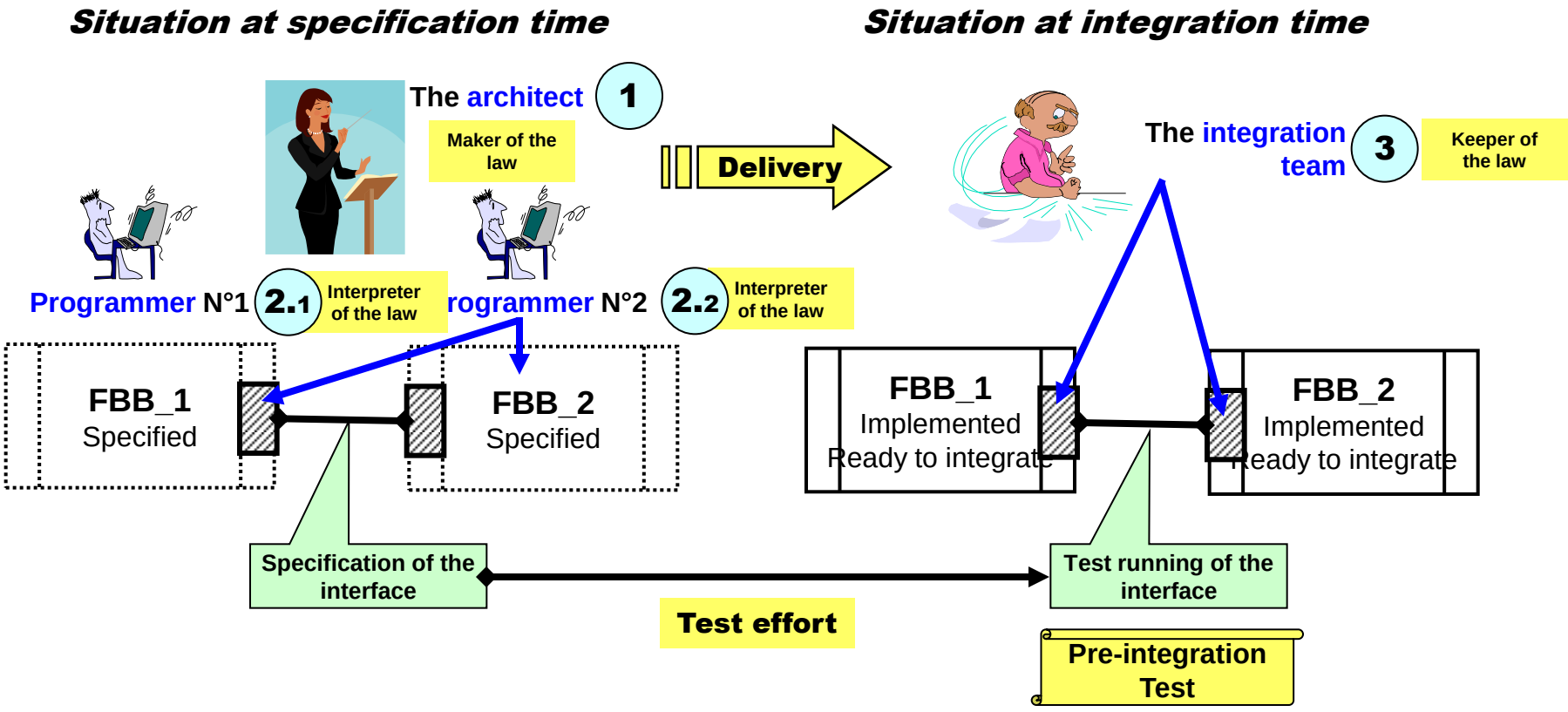
Testing activity is a dual form of programming activity

- Result of testing activity is a set of texts, like programming :
  - **test programs**
  - **test data**

Testing is now recognized as a fundamental aspect of software system engineering

- Test driven engineering

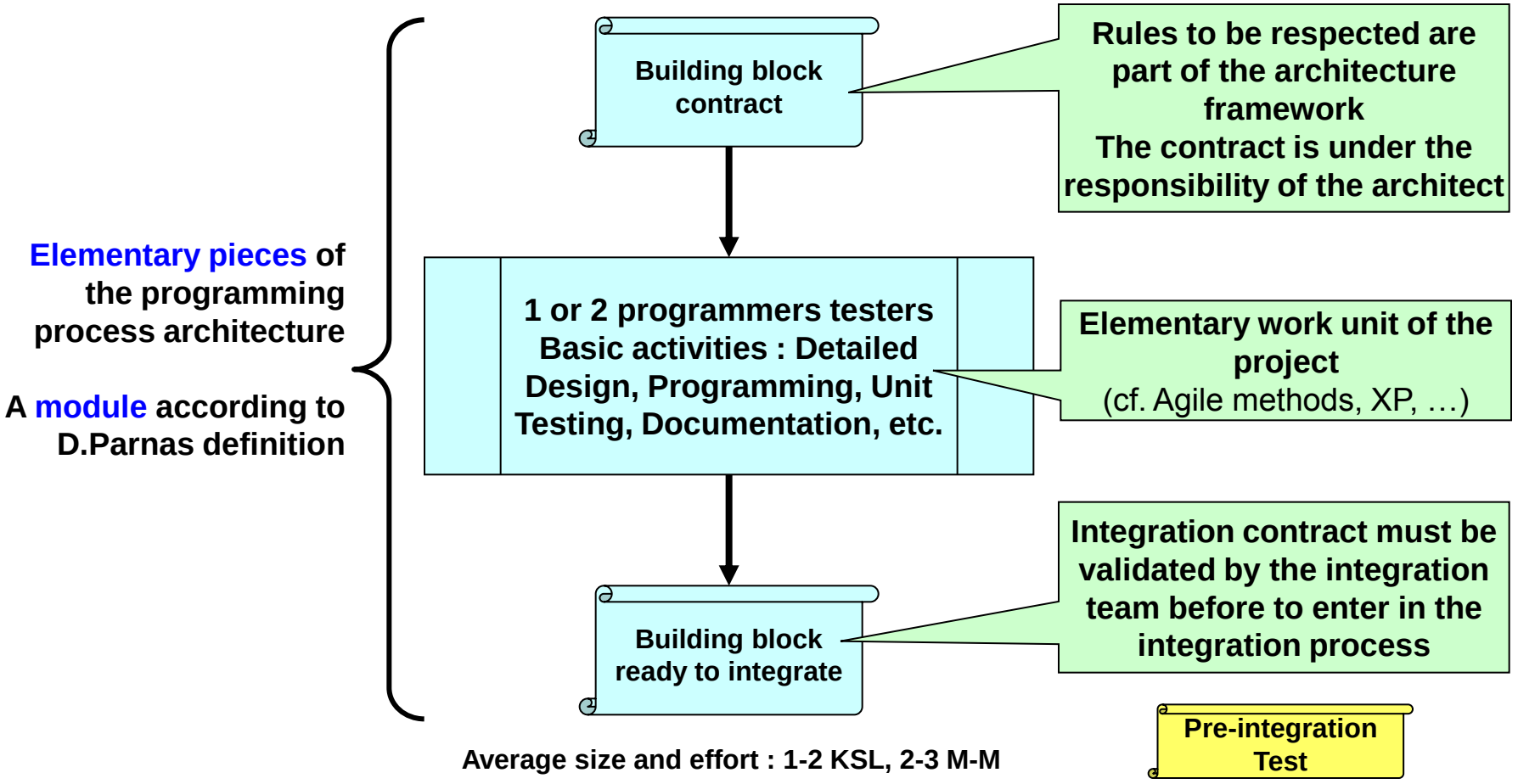
Cost of interface testing
The actors point of view



➘ **Binary relations may be represented simply using 2×2 MATRIX**

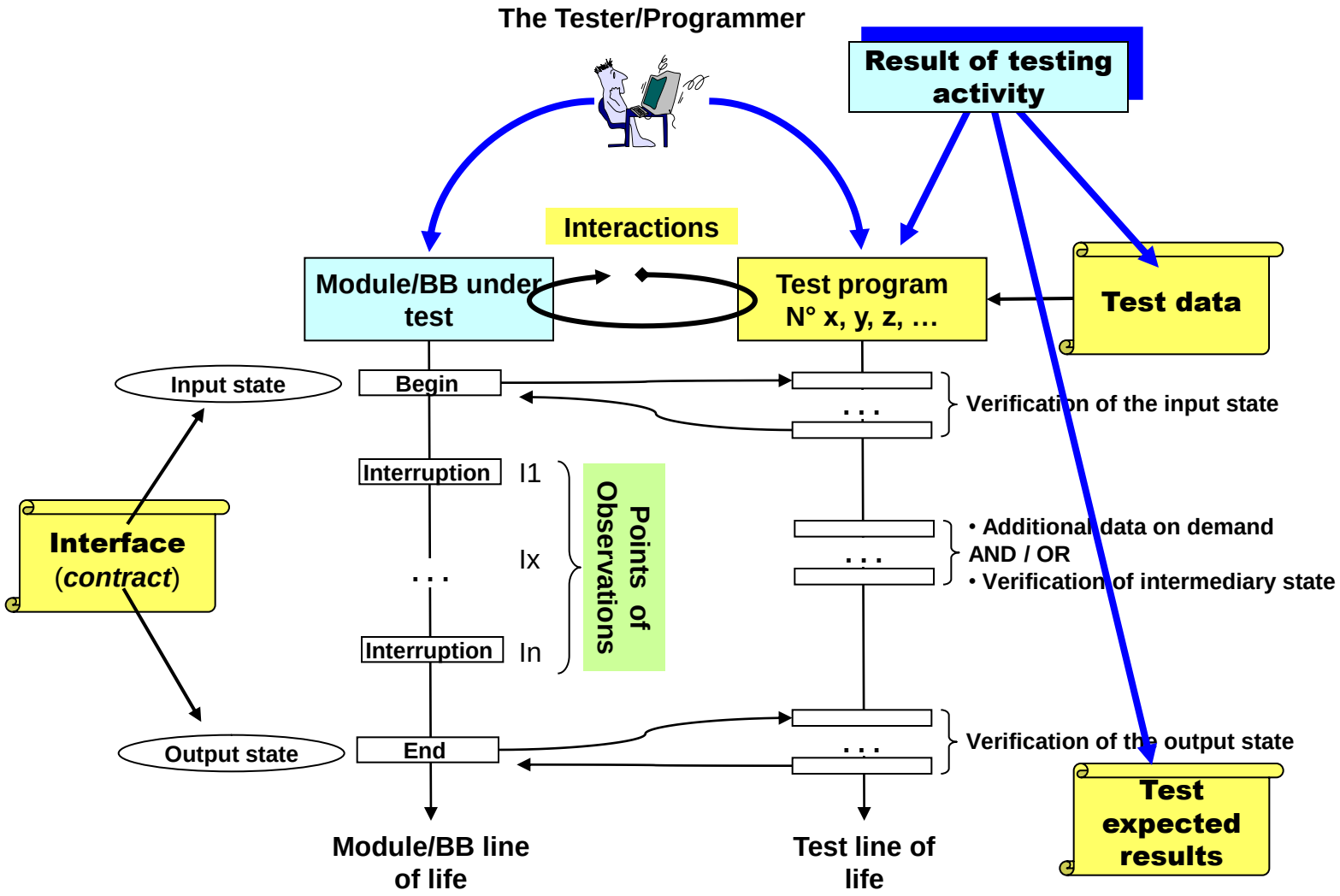
➘ **But more complex relationships may exist (Ternary, …)**

Programming activity : Building blocks
Functional blocks – Service blocks

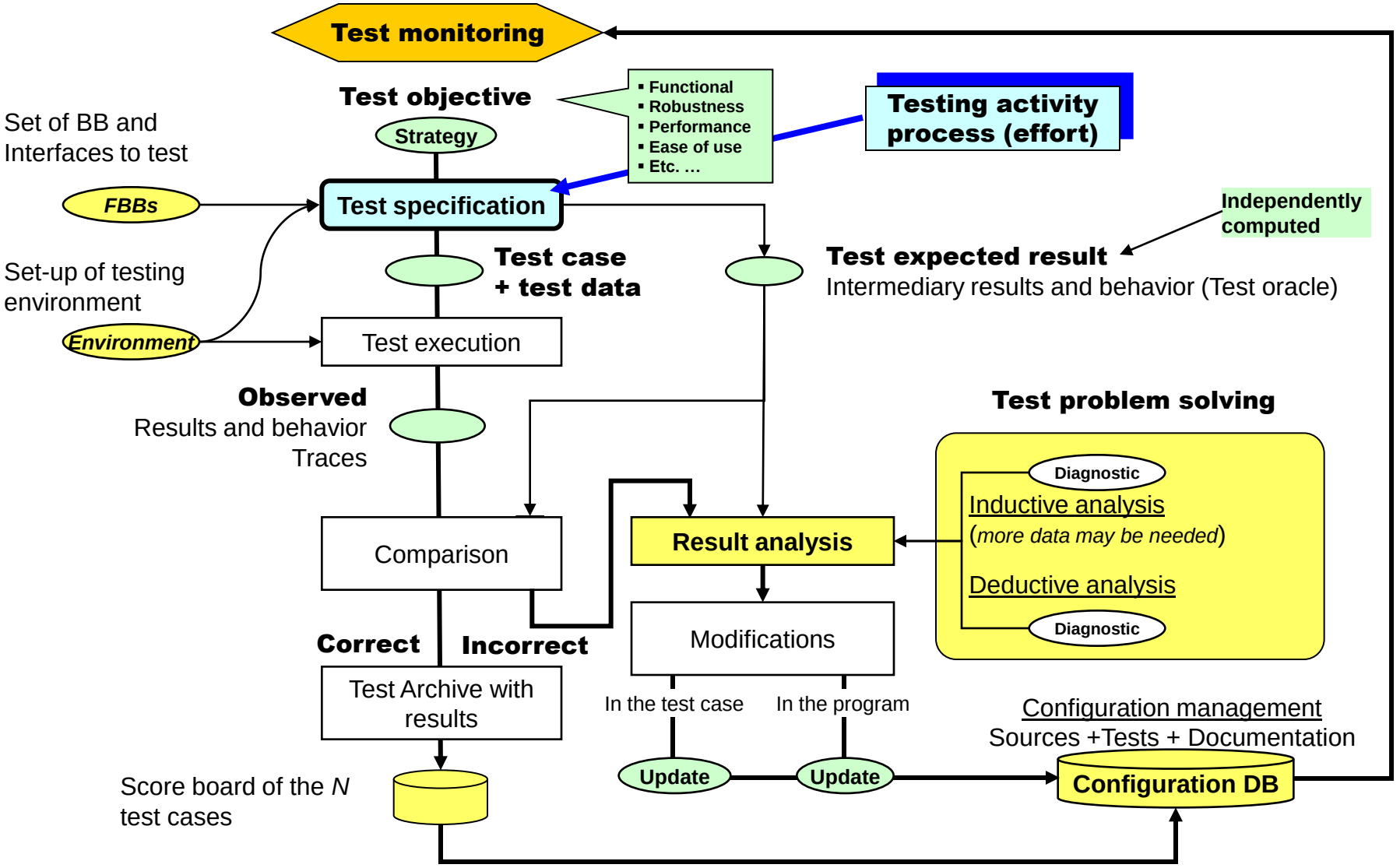


Testing activity

Test driven development

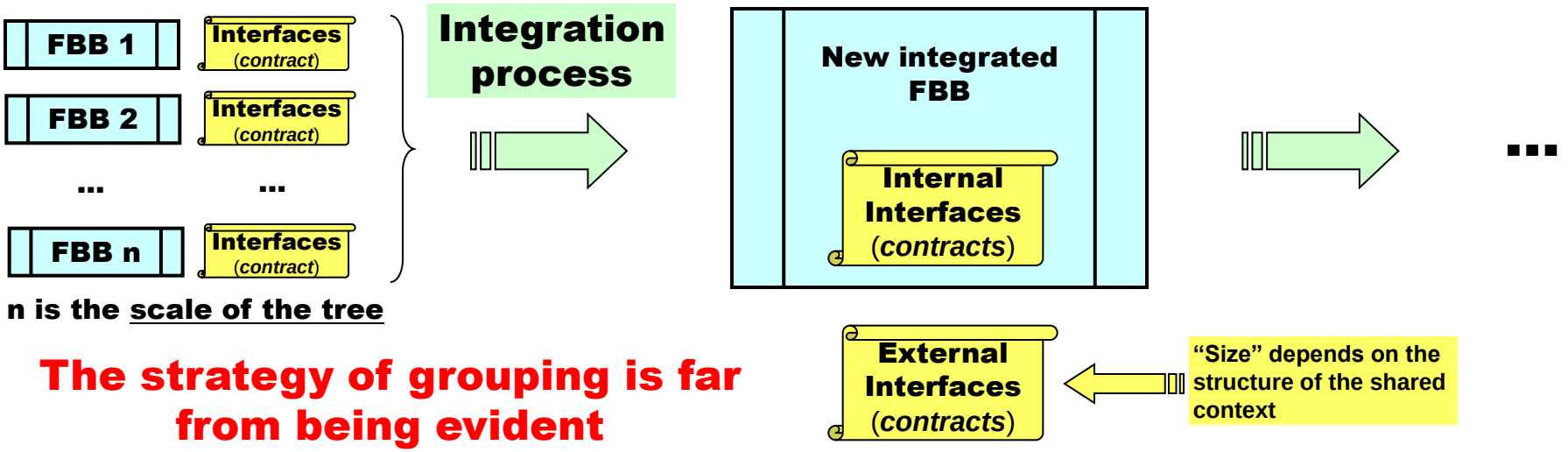


Test process life cycle
The "machine" for testing



Constructing the integration tree
The art of grouping FBB

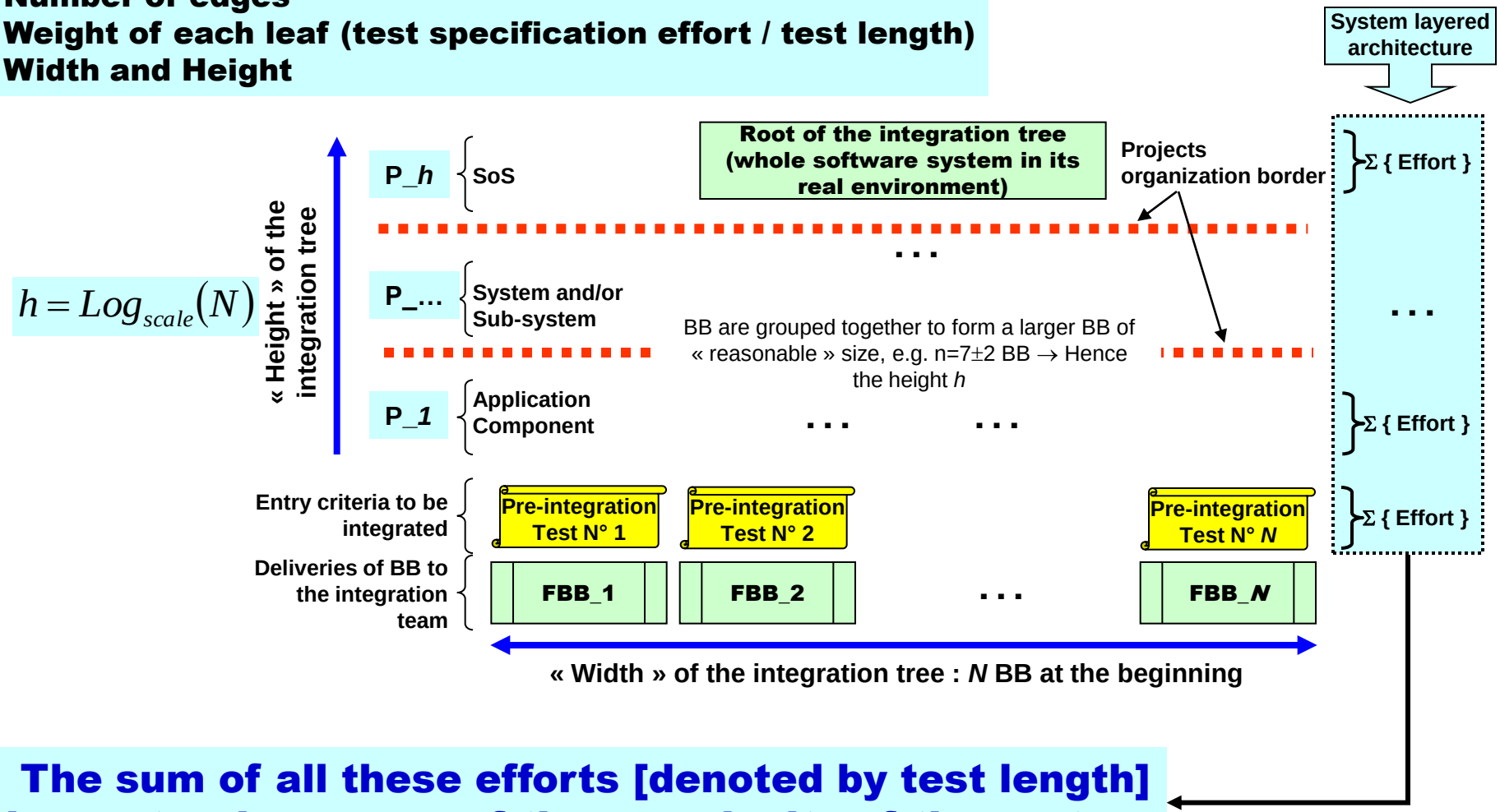
↘The scale of the integration tree: FBB are grouped together to form a larger FBB of « reasonable » size, e.g. n=7±2 BB → Hence the height *h*
n depends on the type of coupling (flow of control [synchronous/asynchronous]**, shared data, events)**



Effort to integrate the *n* FBB depends on the interfaces architecture of the corresponding integration step and on the functional size of the FBB

Width / Height of the integration tree

Tree properties:
- **Number of leaves (nodes)**
- **Number of edges**
- **Weight of each leaf (test specification effort / test length)**
- **Width and Height**

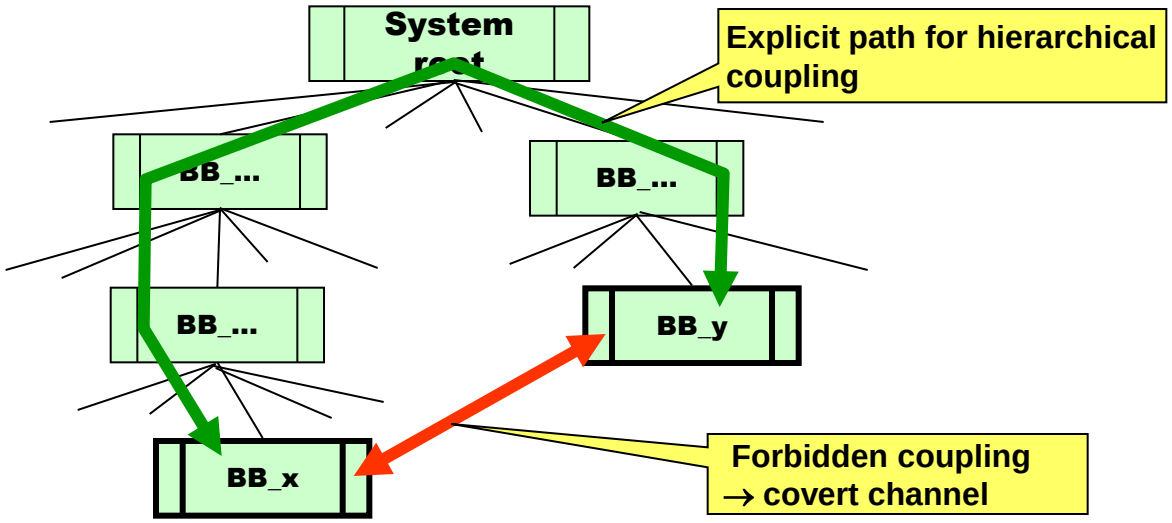


The sum of all these efforts [denoted by test length] is a natural measure of the complexity of the system

Non hierarchical coupling

Additional complexity

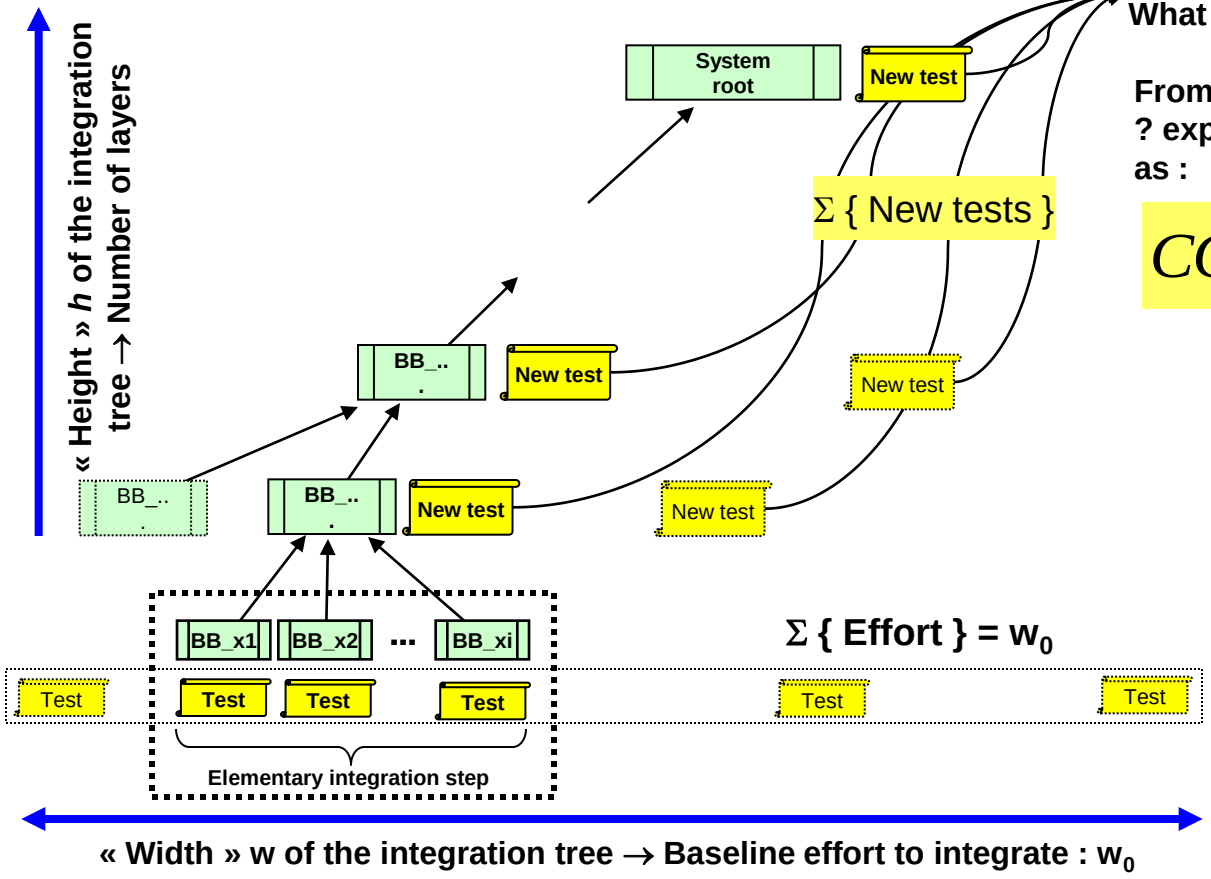
↘ Hierarchical complexity is the minimum complexity → Depends on number of edges
If D.Parnas modularity rules have been violated, for any reason, the effective complexity will be higher



↘If no rules have been specified, or if the usage of rules have not been respected (no quality assurance, no review of interfaces, ...), the effective integration complexity will be much higher :
For example, if *N* BB have a large shared context, each BB may interact with any other, then the complexity will be $O(N^2)$

if the ordering is significant, then the set of parts will have to be considered $O(2^N)$

To summarize



Fundamental problem of the integration process: What is the law of growth of these new tests?

From a practical point of view: how much time w_0 ? expressed by a Complexity Cost Function such as :

$$CCF \rightarrow k \times (w_0)^{1+f(coupling)}$$

Case 1: no coupling, FBB are totally independent (truly transactional)

- f(coupling) =0

Case 2: some coupling, Integrated FBB requires new tests which depends linearly of w_0

- f(coupling) <1
- f(coupling) >1

Case 3: tightly coupling, FBB are dependent each others, execution order is significant (cache effect, context dependencies, covert channels, …); new tests must be developed at each layer

- f(coupling) = f(h, w_0, tree_structure, time)

Cannot be avoid, if no rules have been given and controlled → Organizational entropy