De l'Ingénierie Logicielle à l'Ingénierie des Protocoles - Introduction

Test de Conformité
en boîte noire & techniques de descriptions formelles

Stéphane Maag
Stephane.maag@telecom-sudparis.eu
TSP
Département Réseaux et Services Multimedia Mobiles

Slides&docs:
http://www-public.imtbs-tsp.eu/~maag/P7/

Historique du logiciel

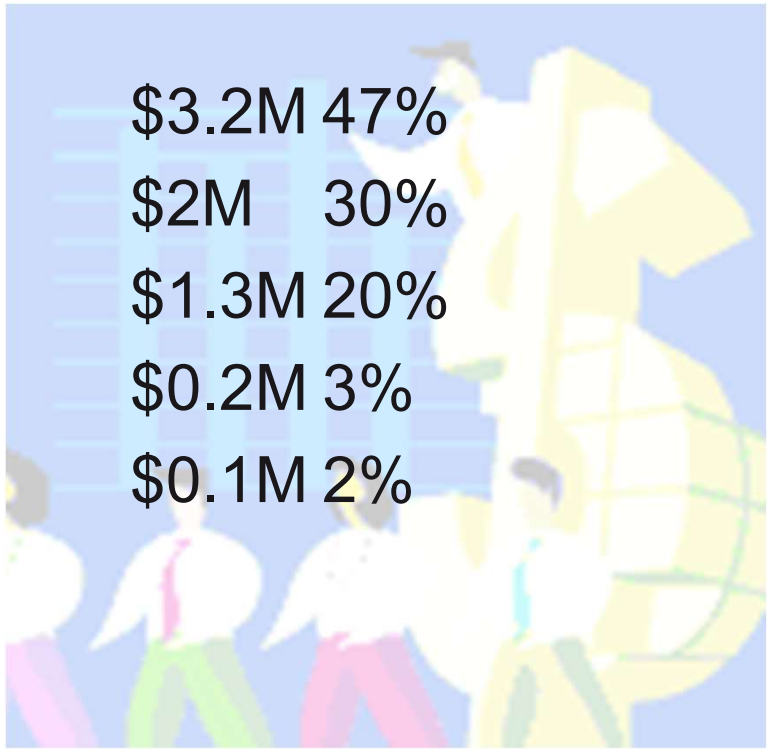
- **50's-60's: "Ancestral" programming**
  - Logiciels pour les protocoles développés "à la main"
  - "utilisateurs malins" et "gurus Kingdom"

- **Fin des 60's: "Crise du logiciel"**
  - Difficile d'écrire de gros programmes
  - Difficile de les utiliser et de les faire évoluer
  - De nombreux projets échouent

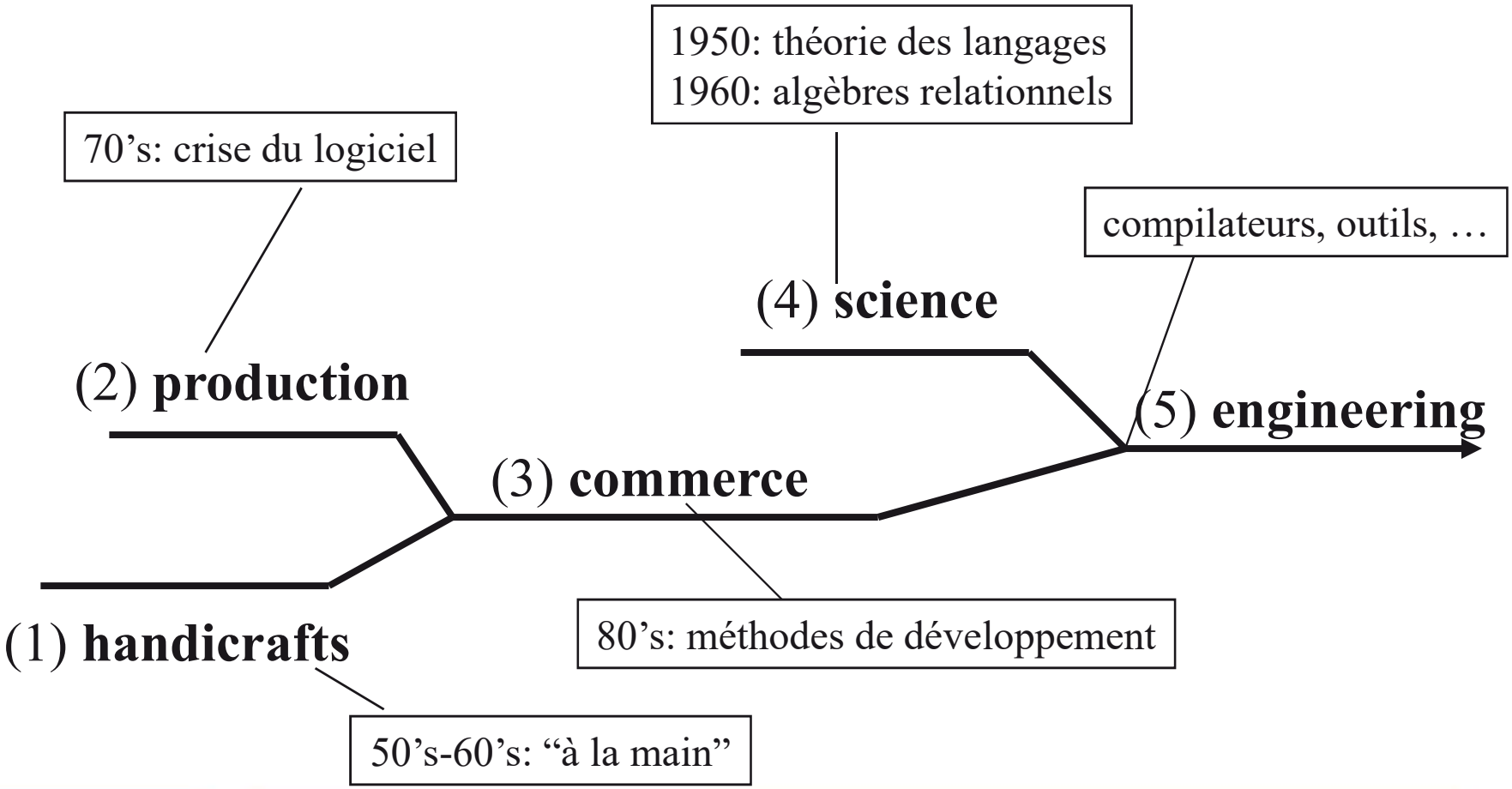
La “Crise du logiciel”

- **Étude du gouvernement Américain en 1979:**

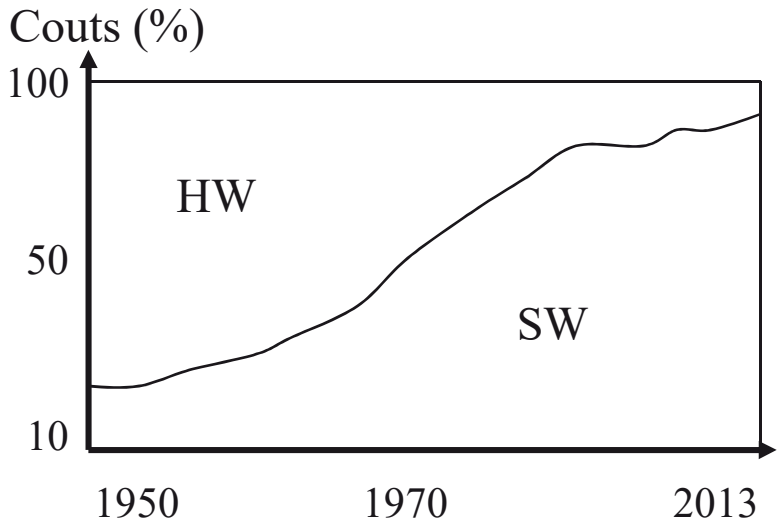
• Payés mais non délivrés	$3.2M	47%
• délivrés mais jamais payés	$2M	30%
• Stoppés	$1.3M	20%
• Utilisés après modifications	$0.2M	3%
• Utilisés tels quels	$0.1M	2%



Chronologie de l'ingénierie du logiciel



Augmentation des développements de logiciels!



La valeur du logiciel augmente chaque année.

Valeur planétaire plus importante qu'avec les puits de pétrole.

⇒ Chaque modification de cette valeur peut avoir d'importante répercussions.

⇒ **Besoin de normalisation**

Protocoles de communication

- Fonctionnement d'un réseau de machines communicantes
  - respect d'un ensemble de normes

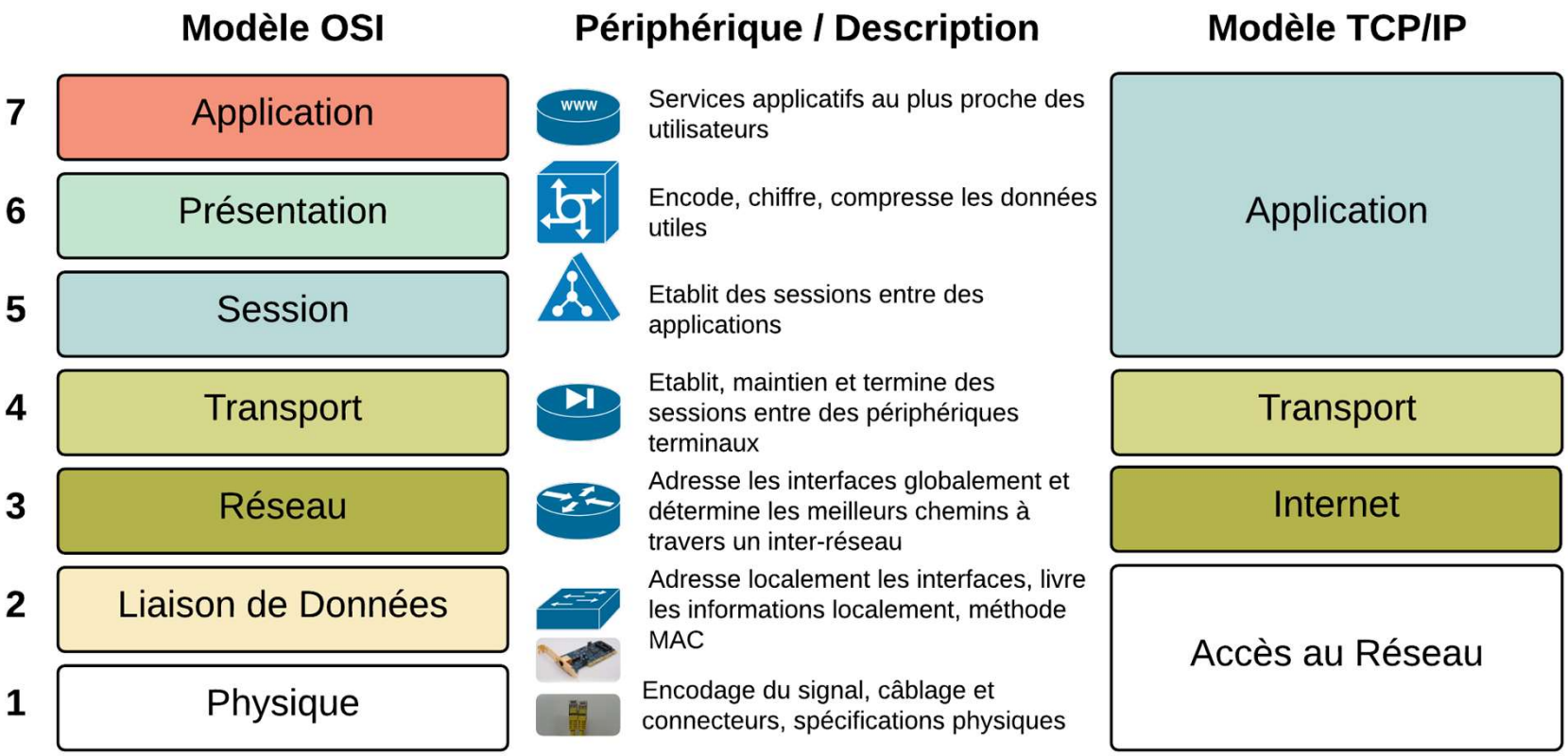
- Les **normes** permettent la communication de réseaux hétérogènes sur les plans matériel et logiciel
  - sont définies par l'intermédiaires de protocoles
  - de nombreux standards

Instituts de standardisation

- ITU, ITU-T (*ITU-T X.224,…*)
- ISO (*ISO 7498,…*)
- IEEE (*IEEE 802.11,…)*
- ETSI (ISDN, SS7)
- IETF (*routing protocol,…*)

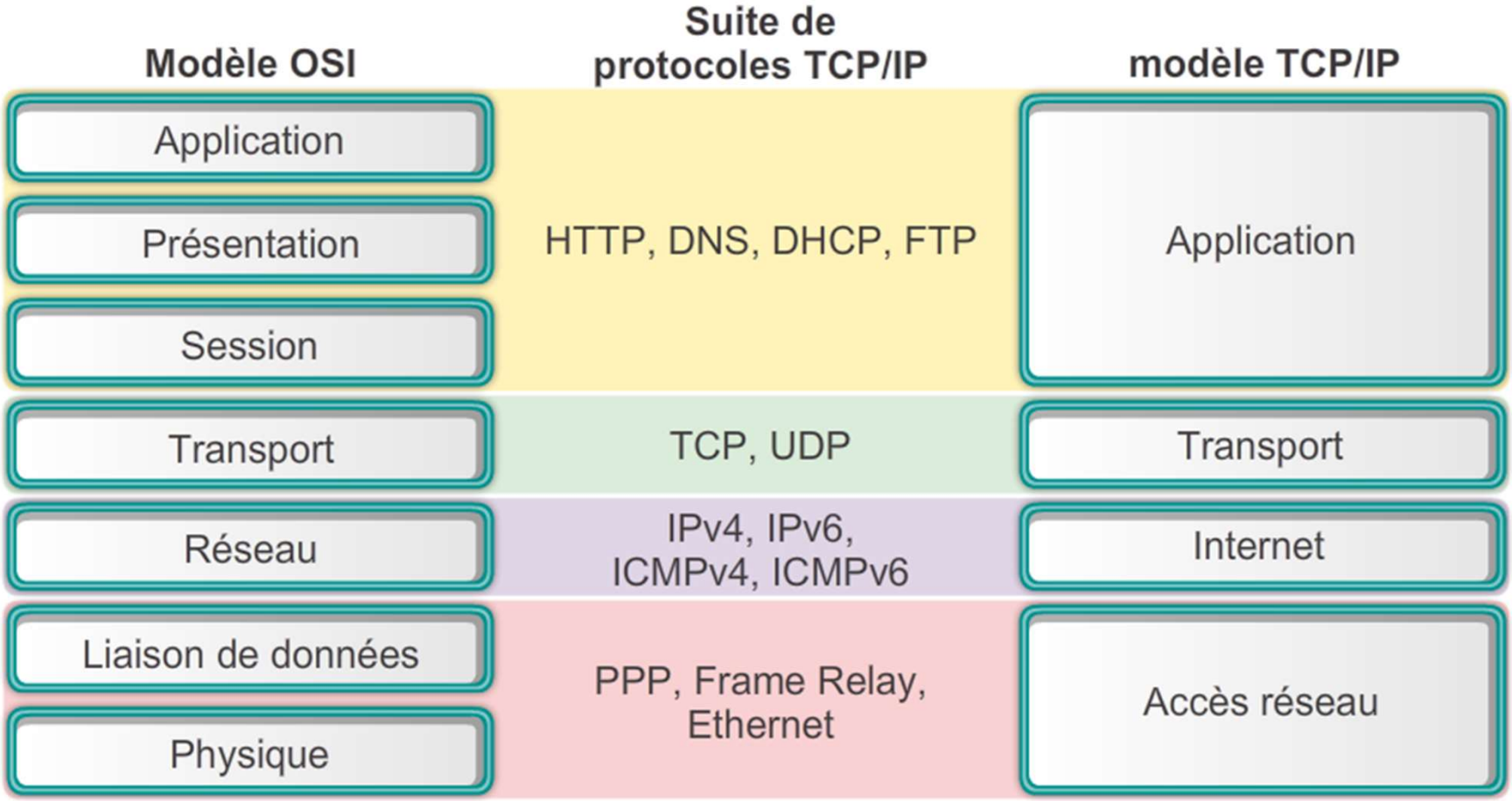
…

Modèles OSI et TCP/IP



https://cisco.goffinet.org/ccna/fondamentaux/modeles-tcp-ip-osi/

Modèles OSI et TCP/IP



https://linux-note.com/modele-osi-et-tcpip/

Modèles OSI et TCP/IP

Application

- Elle est la couche de communication qui s'interface avec les utilisateurs.
- S'exécute sur les machines hôtes terminales.

Transport

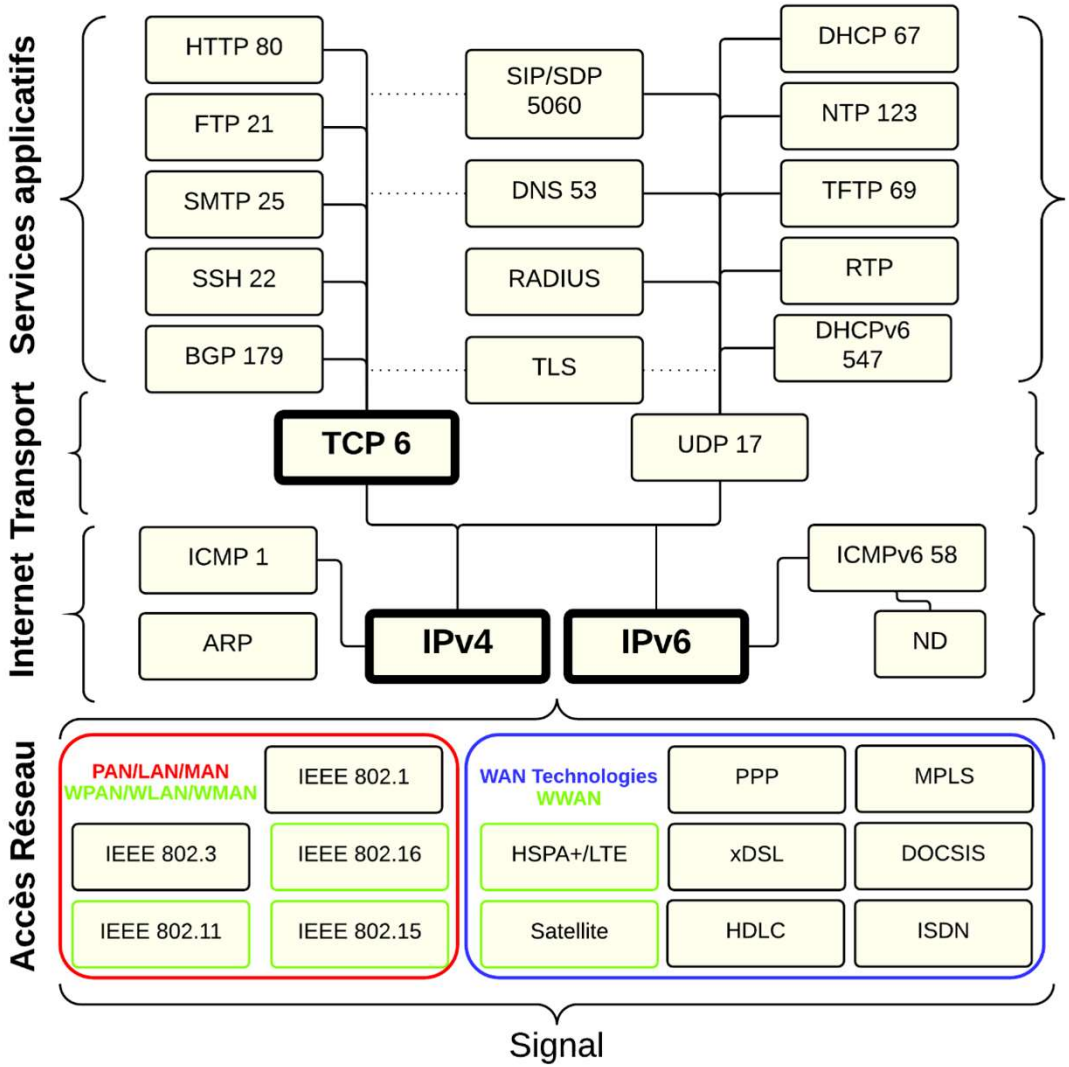
- Elle est responsable du dialogue entre les hôtes terminaux d'une communication.
- Les applications utiliseront TCP pour un transport fiable et UDP sans ce service.
- Les routeurs NAT et les pare-feu opèrent un filtrage au niveau de la couche transport.

Internet

- Elle permet de déterminer les meilleurs chemins à travers les réseaux
- Identifie globalement les interfaces
- Les routeurs transfèrent le trafic IP qui ne leur est pas destiné.

Accès réseau

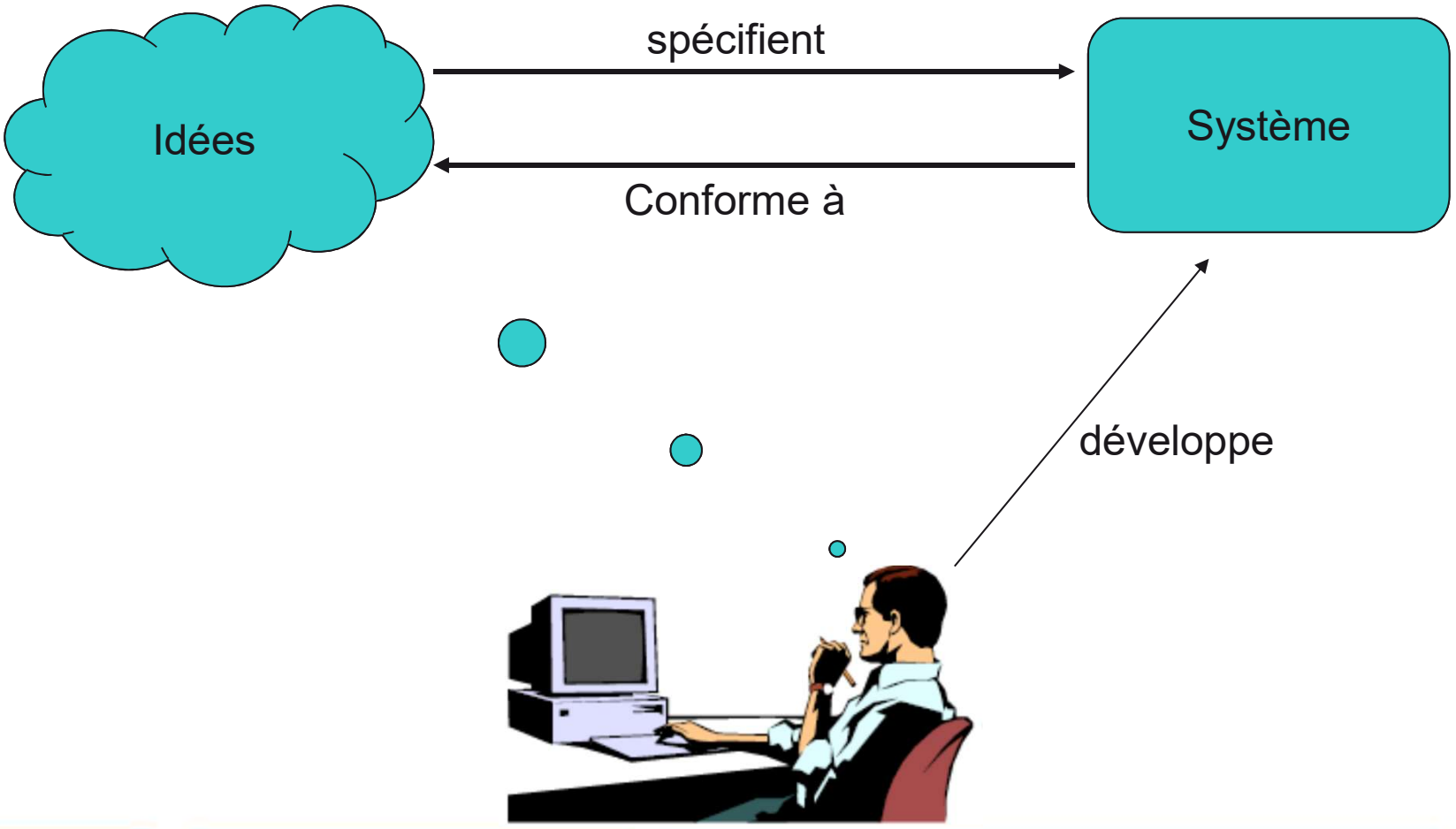
- Elle organise le flux binaire
- identifie physiquement les interfaces
- Elle règle la méthode d'accès au support
- Elle place le flux binaire sur les support physique
- Commutateurs, câbles, NIC,



https://cisco.goffinet.org/ccna/fondamentaux/modeles-tcp-ip-osi/

Conception/Ingénierie de nouveaux protocoles/services/apps/systèmes/…

Idées et Systèmes

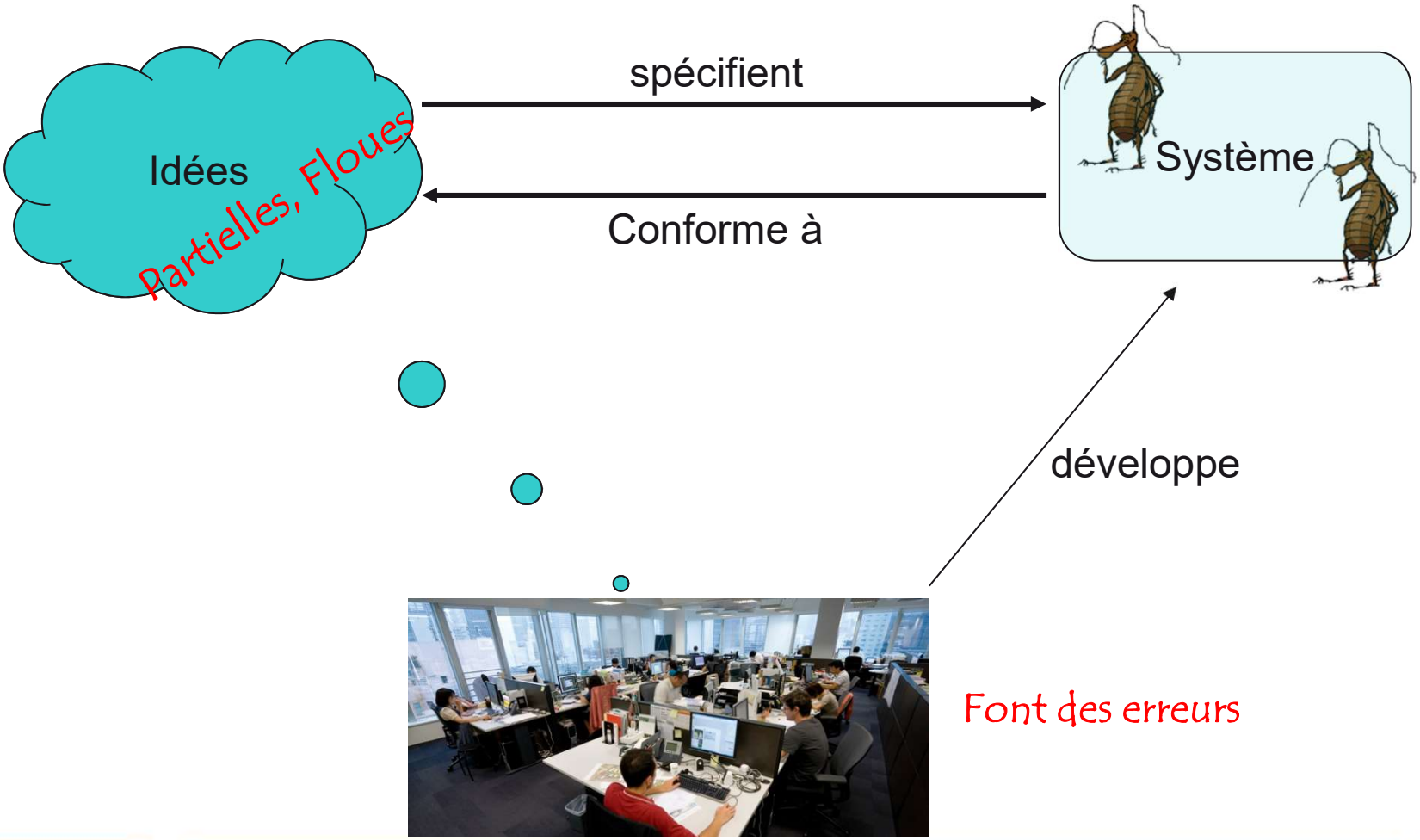


Platon a dit …

The world that appears to our senses is in some way defective and filled with error, but there is a more real and perfect realm, populated by entities (called "forms" or "ideas") that are eternal, changeless, and in some sense paradigmatic for the structure and character of our world.

© Stanford Encyclopedia of Philosophy

Idées et Systèmes



Panne ??

- Que signifie « Panne d'un système » ?
- Quelques termes:

L'Homme fait des Erreurs

menant à une Faute dans le système

qui mène à une Panne du système

- C'est quoi une « panne » ?

Terminologie

- **Une Erreur signifie que Monsieur X a écris du code qui a une Faute.**
- **Une Faute est une partie du code qui quand elle est exécutée peut mener à une Panne.**
- **Une Panne est un comportement observable du système qui n'est pas conforme au** cahier des charges.

Le Test, c'est quoi?!

Testing??

- **Qu'est-ce que le Test ??!**
  - Ce n'est **pas**:

    "I've searched hard for defects in this program, found a lot of them and repaired them. I can't find any more, so I'm confident there aren't any. " (Richard Hamlet, 1994, Portland State Univ.)

  - Le test, c'est plutôt :

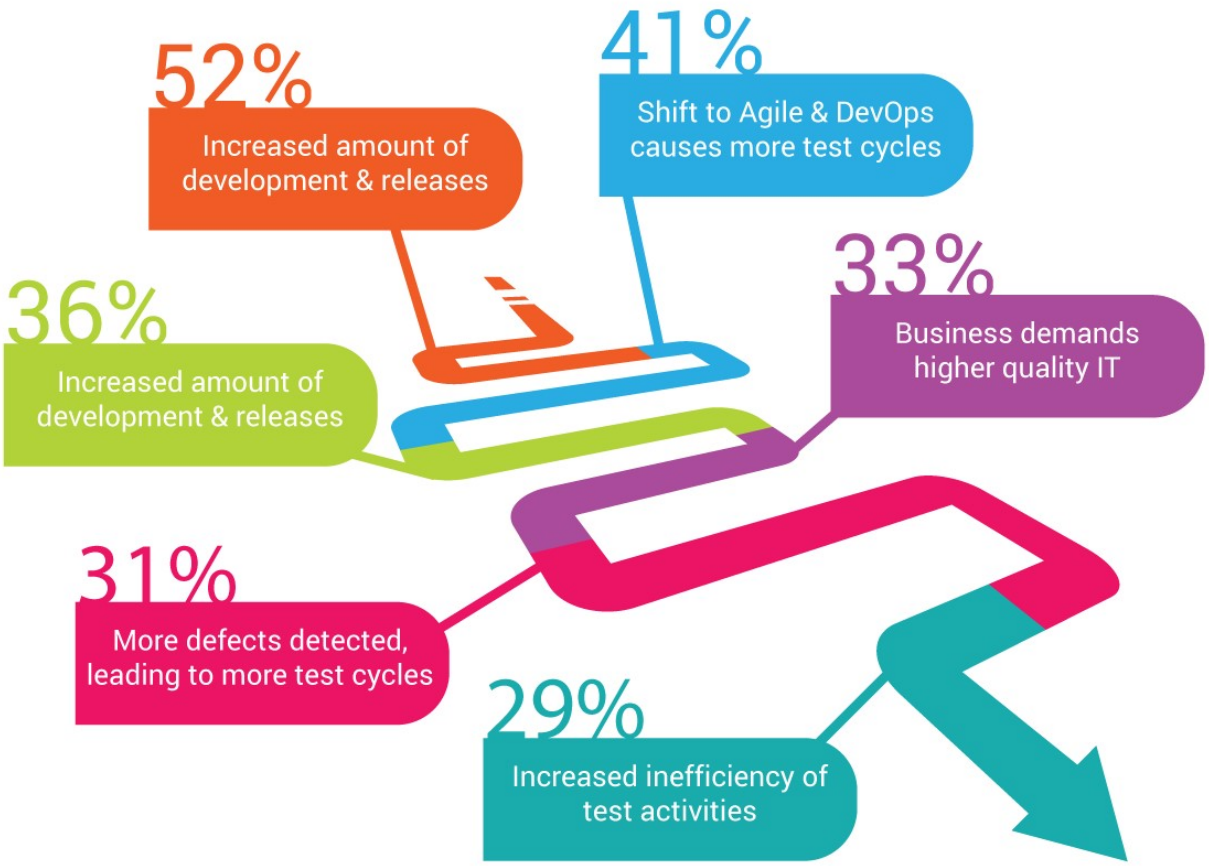
    "A process of analyzing an item to detect the differences between existing and required conditions, and to evaluate the features of the item." (ANSI/IEEE Standard 1059)

Lié aux pannes: Le Test permet la détection des Pannes par l'exécution du système – processus intégrant les activités de sûreté de fonctionnement (SdF ou 'dependability').

- **D'une autre manière** (rapidement!)**:**
  - Suit les besoins/requirements,
  - Fonctionne comme attendu,
  - Est implémenté avec les mêmes caractéristiques.

  ➔ de nombreuses étapes sont nécessaires …

Pourquoi tester ??

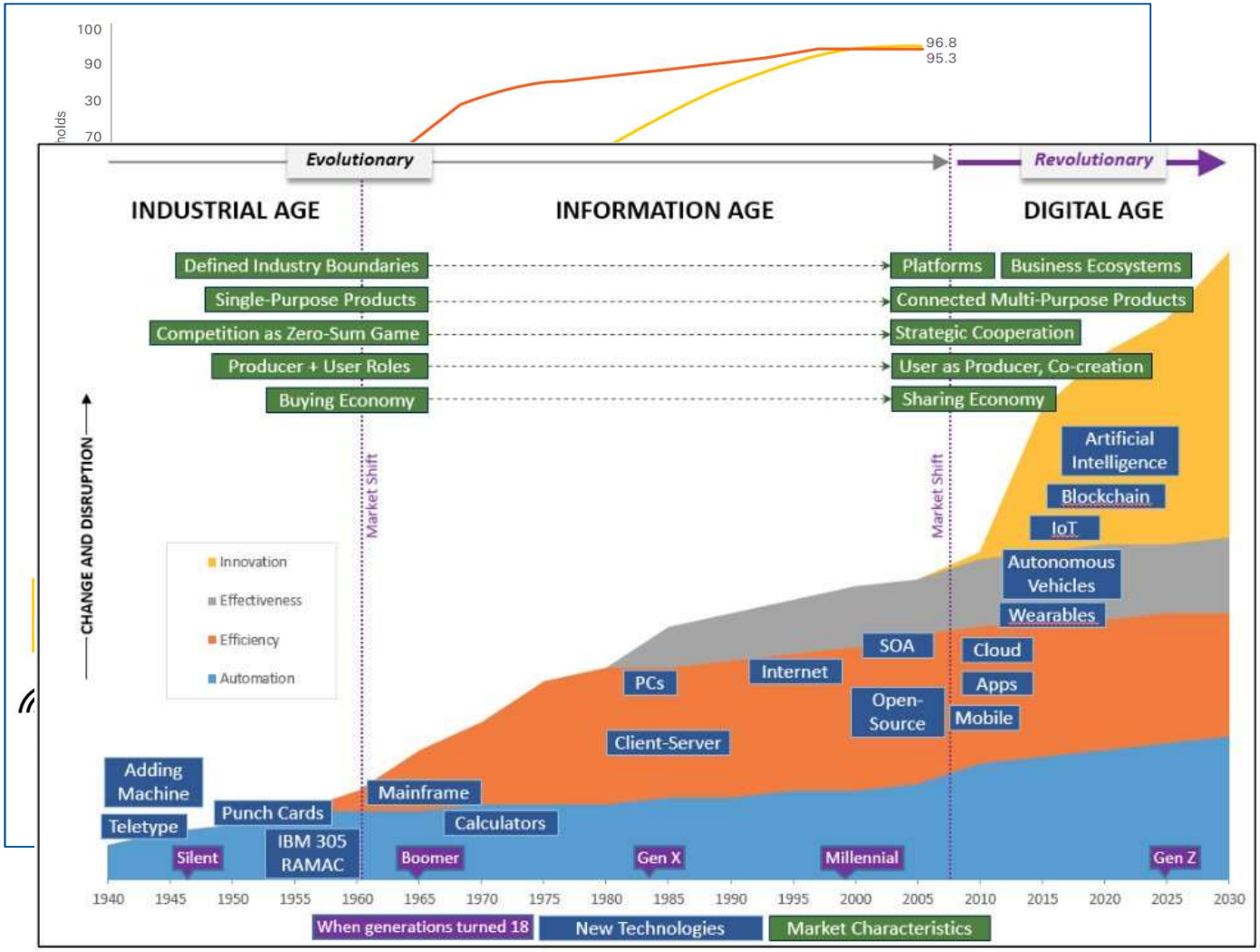


Source: World Quality Report 2016-17

- Inefficient processes
- Underproductive people
- Inadequate tools and technologies
- Dynamic and unstable business and technological landscape
- Poor decision-making or implementation errors

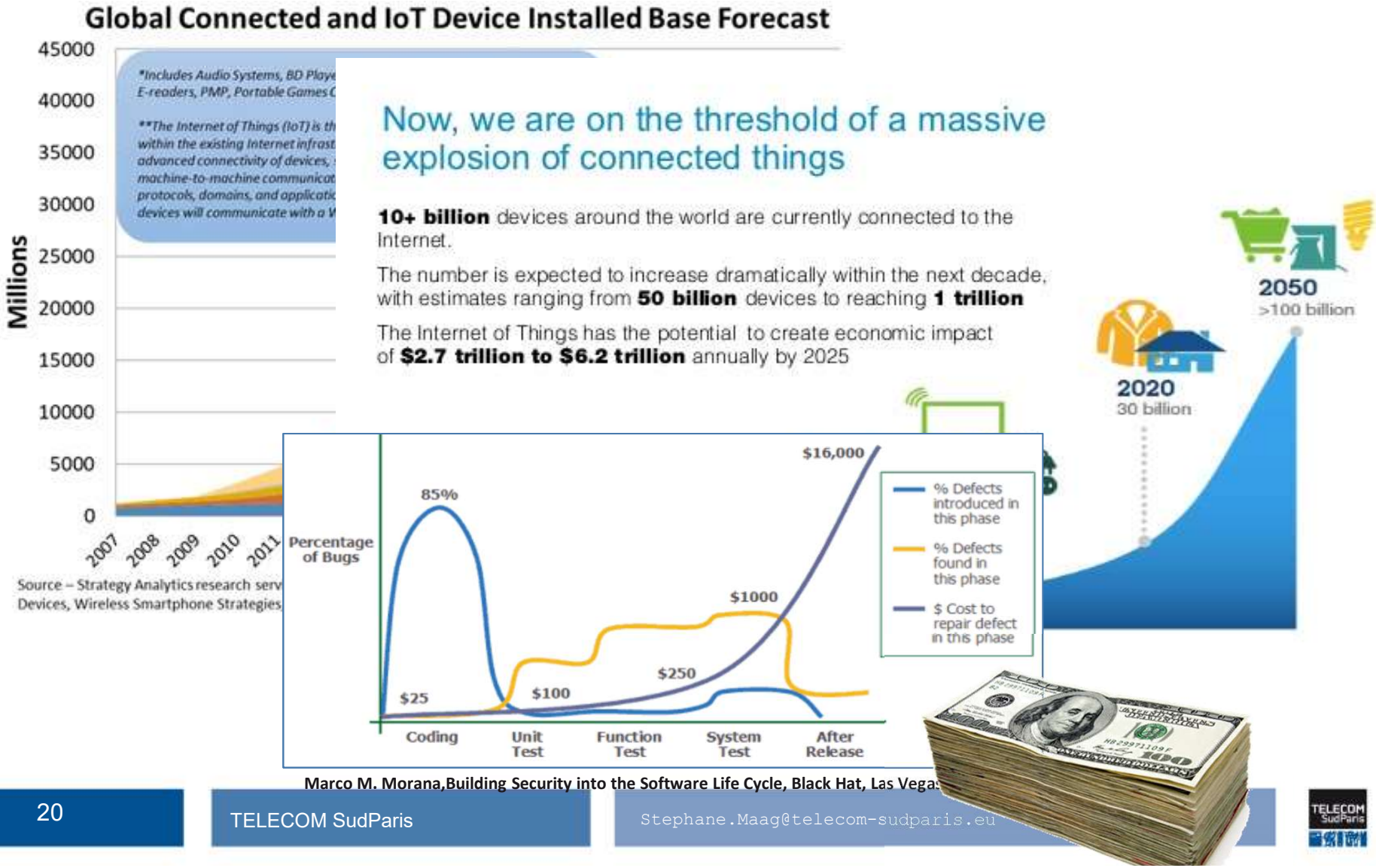
Source: EuroStar Huddle, 2018

Pourquoi tester ??



Majesco, 2017

Pourquoi tester ??

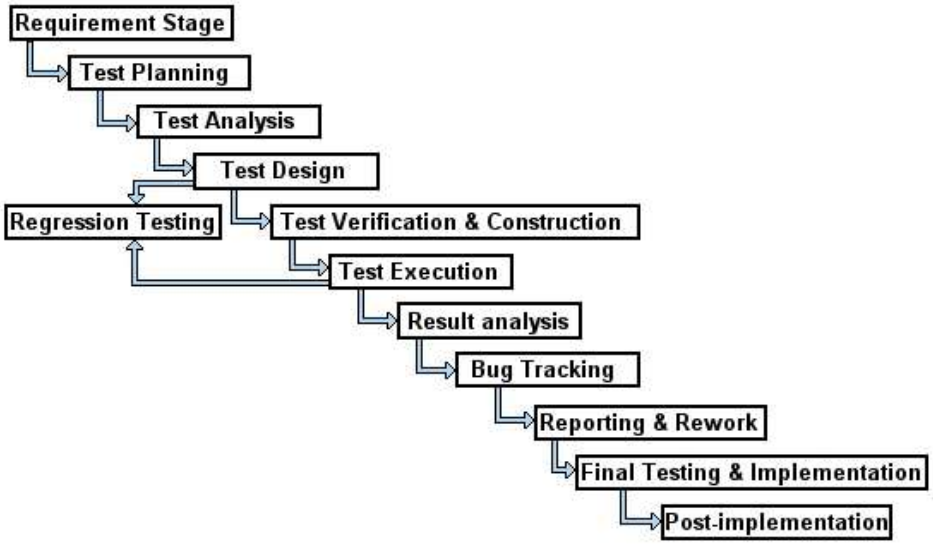


Marco M. Morana,Building Security into the Software Life Cycle, Black Hat, Las Vegas

Pourquoi tester ??

- Selon “quoi” et “quand” nous testons, les raisons du test sont nombreuses !

- Ce peut être pour:
  - Checking the design
  - Bug tracking
  - Conformance
  - “Security”
  - Interoperability
  - Performance
  - Reporting
  - Trust, confidence
  - … thousands of reasons ..
  - Making money?



Pourquoi tester ??

- **Un système est tester pour :**
  - Trouver des bugs *(debug testing)*:
    - Principalement par **Unit** et **Integration** testing, mais aussi
      - Acceptance, Conformance testing
      - Robustness testing
      - Coverage testing, etc.

  - Mesurer la qualité (cf. ISO9126 et ISO25000) et la fiabilité (inférence statistique),
  - Mesurer sa performance (QoS, Markov ...),
  - S'assurer de l'acceptance du client (acceptance test),
  - …

- **Bien sûr, il existe de très nombreuses approches!**

Comment Tester ??

- Selon “quoi”, “quand” et “pourquoi” nous testons, des centaines de techniques sont aujourd’hui utilisées! [Hierons et al 2009]

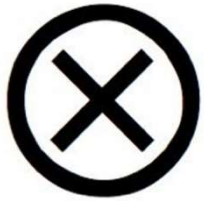
- Les techniques **formelles** de test des systèmes complexes sont cruciales à travers l’analyse des contraintes fonctionnelles de ces systèmes avec des méthodes dites **actives** ou **passive**.
  - Techniques de Description Formelles.
  - Architectures de test actif.
  - Test passif/monitoring.

Approach	Type of Testing	Manual Testing		Automated Testing on Device
		Using Device	Using Emulators	
Standard Testing	Unit Testing	No	Yes	No
	Integration Testing	No	Yes	No
	System Testing	Yes	No	No
	Regression testing	Yes	No	Yes
	Acceptance testing	Yes	No	No
Special type of testing to address specific challenges	Compatibility Testing	Yes	No	Yes
	GUI Testing	Yes	No	No
Type of testing more relevant for enterprise mobile business application	Performance Testing	Yes	No	Yes
	Security Testing	Yes	No	Yes
	Synchronization Testing	Yes	No	No

Testing techniques

There are hundreds of techniques !! Depends on:

- **Testing methods**
  - Static vs. dynamic testing
  - The box approach
    - White-box testing
    - Black-box testing
      - Visual testing
    - Grey-box testing
- **Testing levels**
  - Unit testing
  - Integration testing
  - Component interface testing
  - Protocol testing
  - System testing
  - Operational Acceptance testing
- **Testing process**
  - Traditional waterfall development model
  - Agile or Extreme development model
  - Top-down and bottom-up
  - Sample testing cycle



- **Testing types**
  - Installation testing
  - Compatibility testing
  - Smoke and sanity testing
  - Regression testing
  - Acceptance testing
  - Fuzzy testing
  - Alpha testing
  - Beta testing
  - Functional vs non-functional testing
  - Continuous testing
  - Destructive testing
  - Software performance testing
  - Usability testing
  - Accessibility testing
  - Security testing
  - Internationalization and localization
  - Development testing
  - Data testing
  - Concurrent testing
  - Conformance testing
  - Sandwich testing
  - Big Bang testing
  - .etc.

“Formelle”, i.e.??

Avez vous déjà tester votre production vous-même?!

1. Vous développez votre système,
2. Vous vérifiez quelques fonctionnalités bien choisies,
3. Si tout est ‘OK’, alors vous décidez que c’est testé!

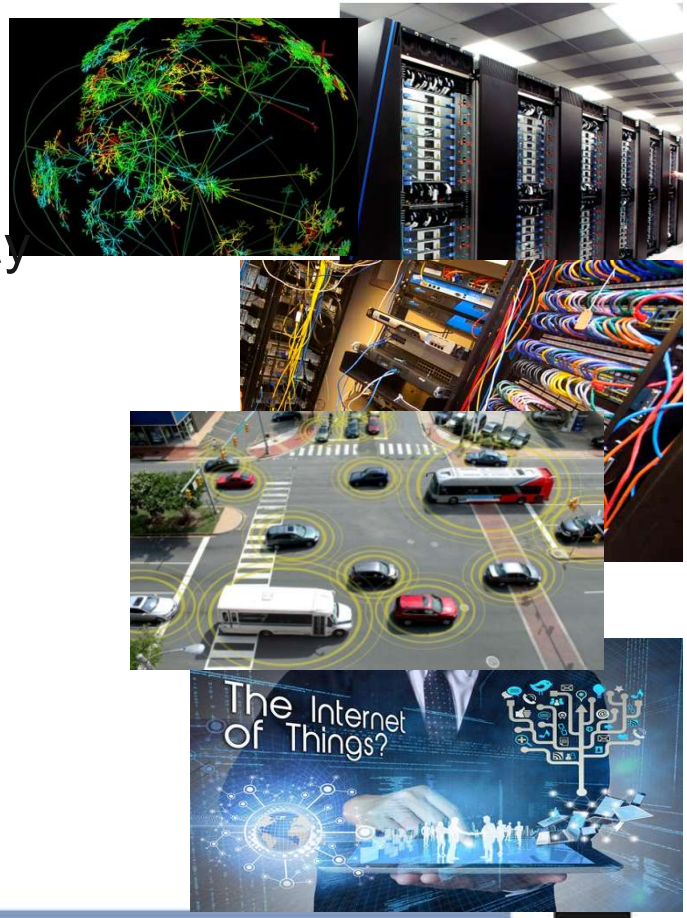
➔ obsolète !!

➔ MC Gaudel, ***Testing Can Be Formal, Too***

Proceedings of the 6th International Joint Conference CAAP/FASE on Theory and Practice of Software Development, Pages 82-96, Springer-Verlag, 1995

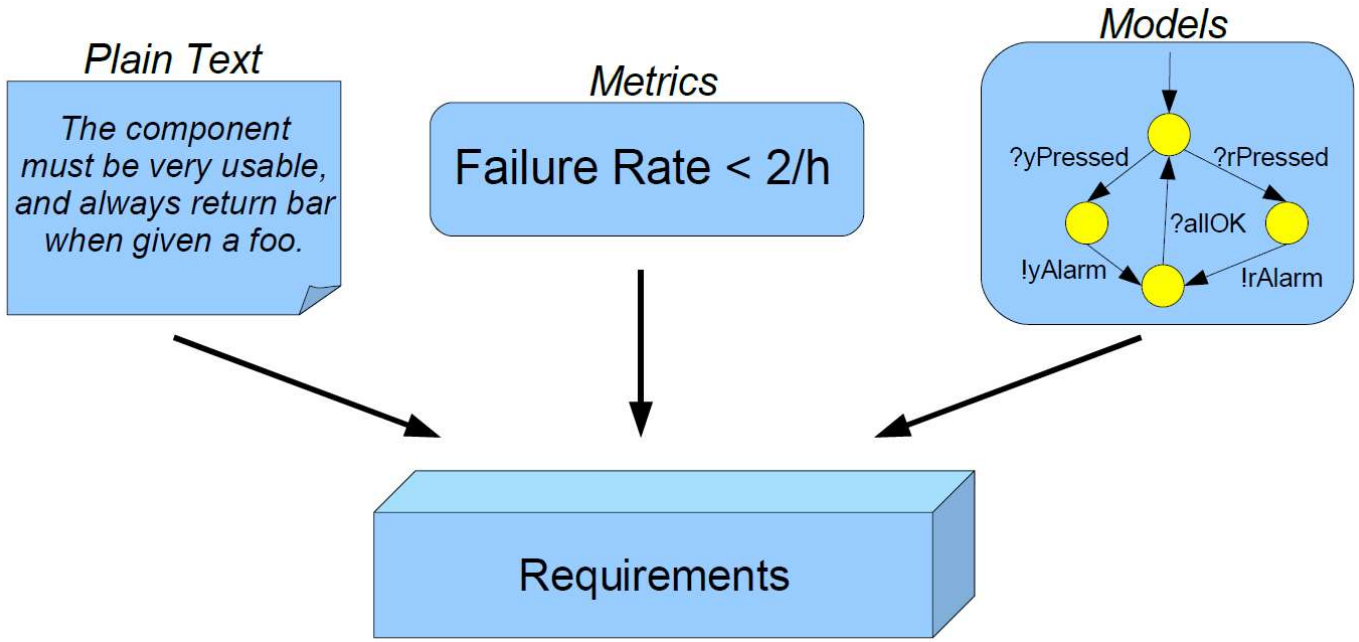
“Formelle”, i.e.??

- **En effet, dans les systèmes complexes actuels, le ‘test manuel’ n’est pas ‘efficace’!**
- **L’automatisation est devenue cruciale.**
  - Conception fonctionnelle in a “smart” way
  - Conception des contraintes in a “clear” way
  - Définition de Testing architectures
  - Test suites generation/execution
  - Testing verdicts generation
  - Diagnostic, réactions,
  - …

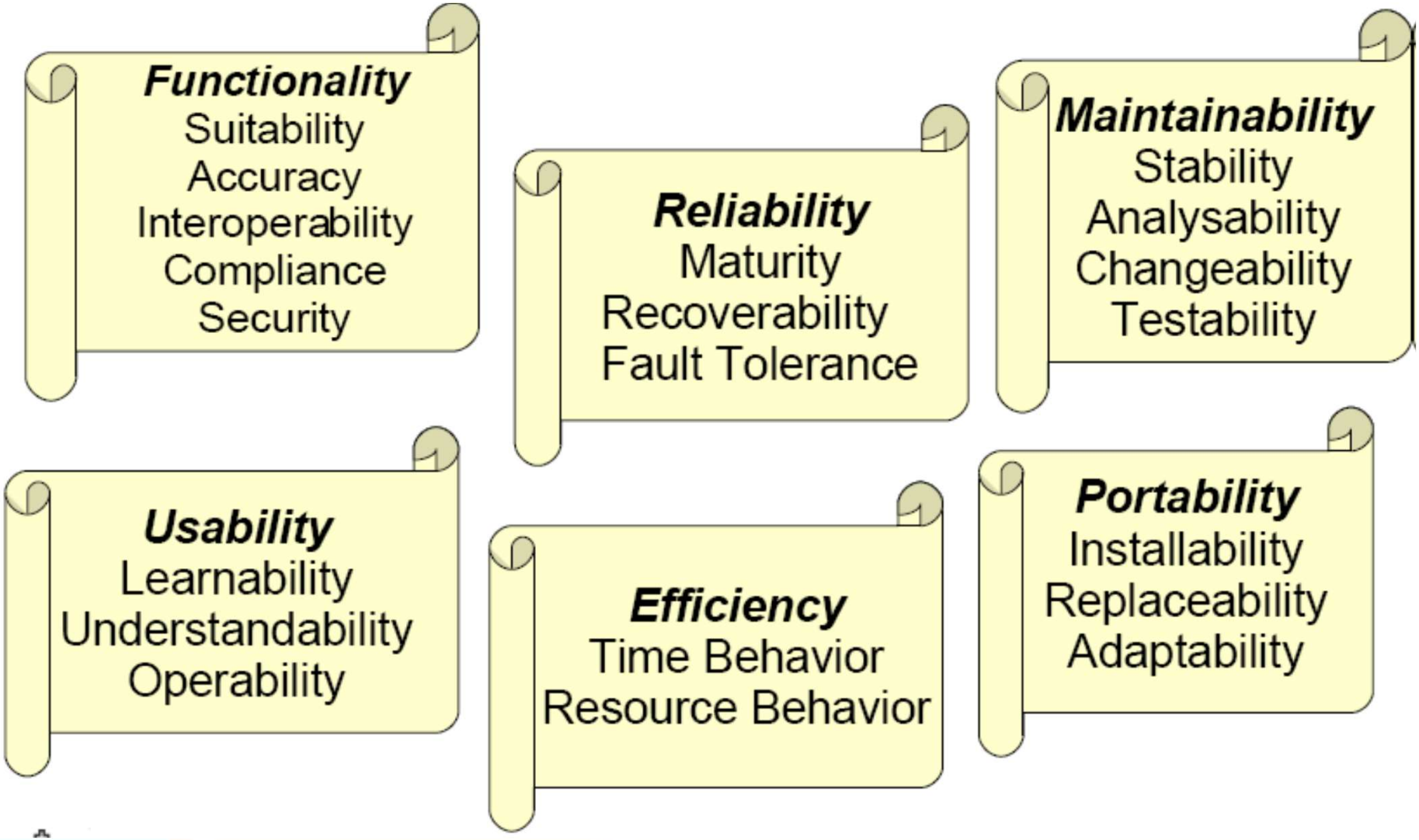


“Formelle”, i.e.??

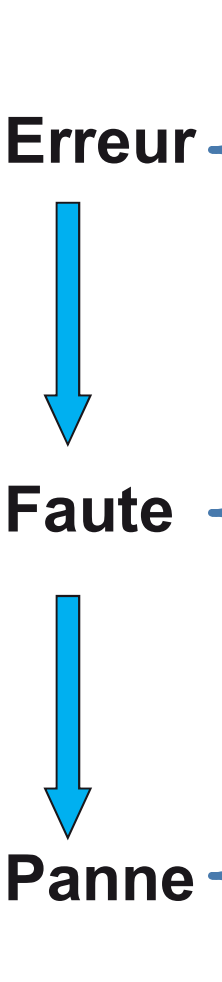
- Les idées d’un système sont nommées *Requirements* ou besoins.
- Pour établir les besoins d’un système, les métriques ne sont pas suffisantes, d’autres documents sont nécessaires.



Qualité des sytèmes - ISO 9126 - ISO/IEC 250xy



Pour améliorer la qualité

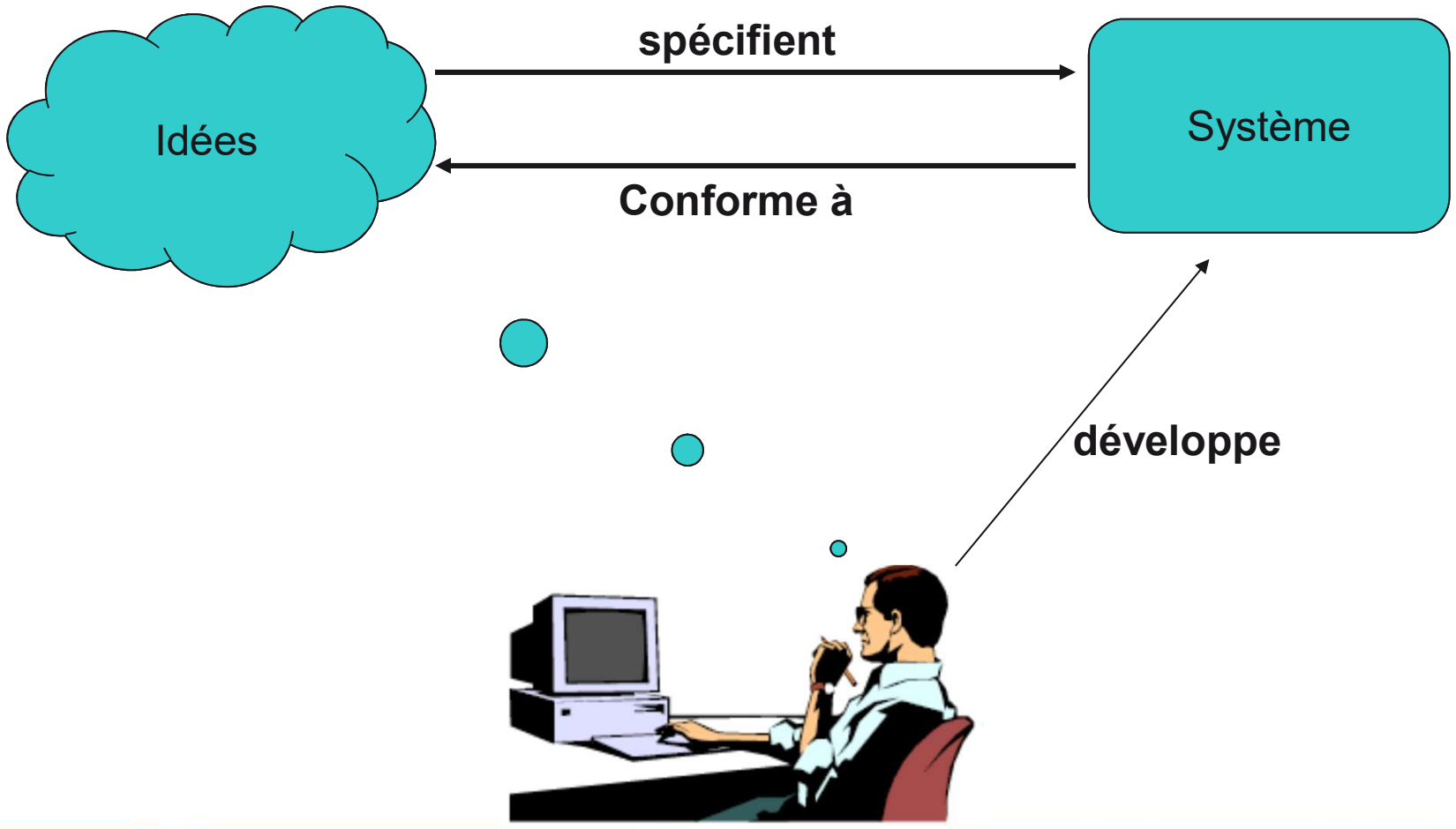


Organize, assess and improve the development process:
V-Model, XP, Agile ...
TQM, Six Sigma, ...
CMM, SPICE, ...

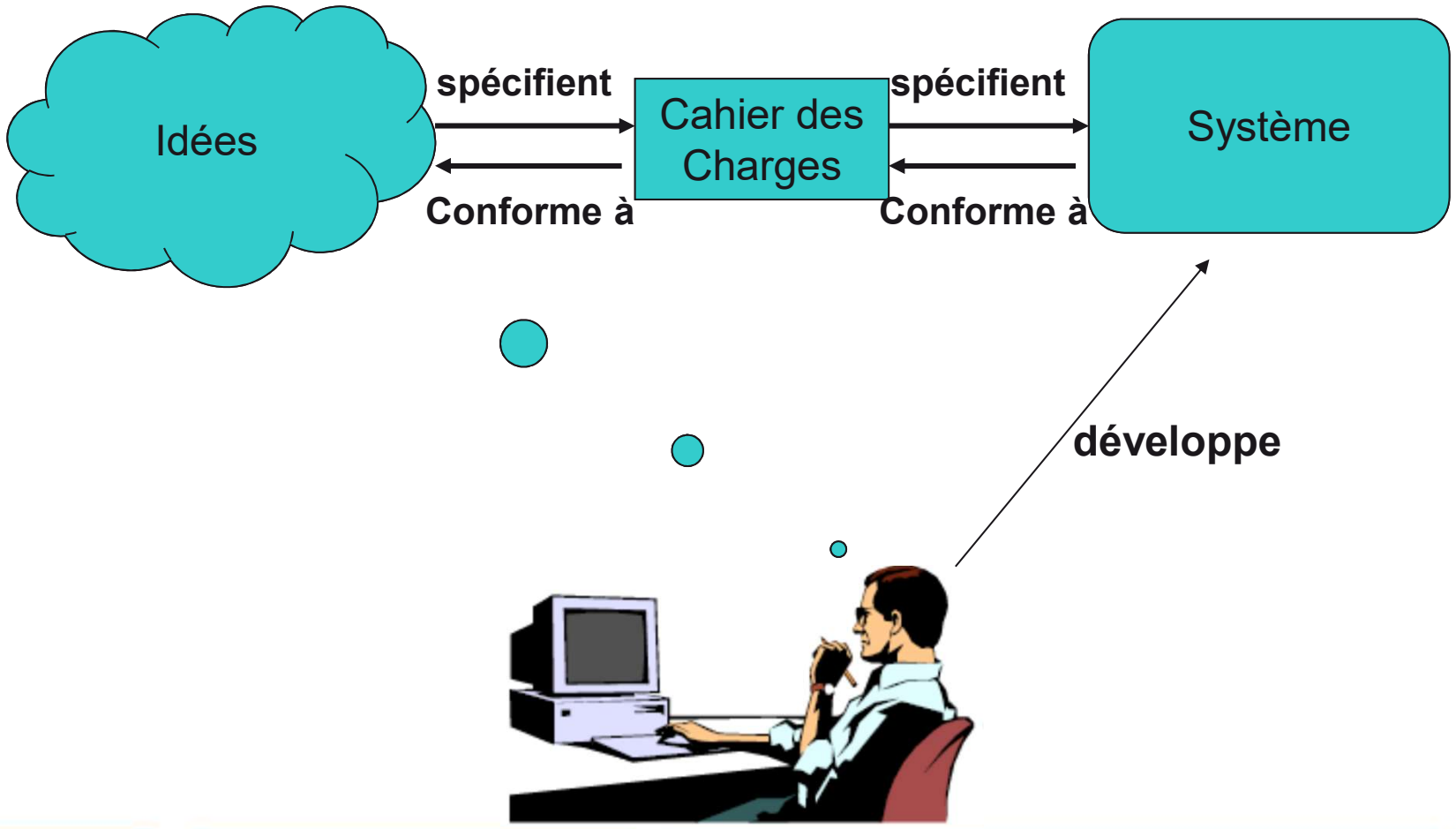
Static/Dynamic analysis
SW testing (xUnit, etc.)
Reviews, reporting,…

Organize, assess and improve the testing process:
Certified Tester, TMM, TPI, CTP, STEP,…

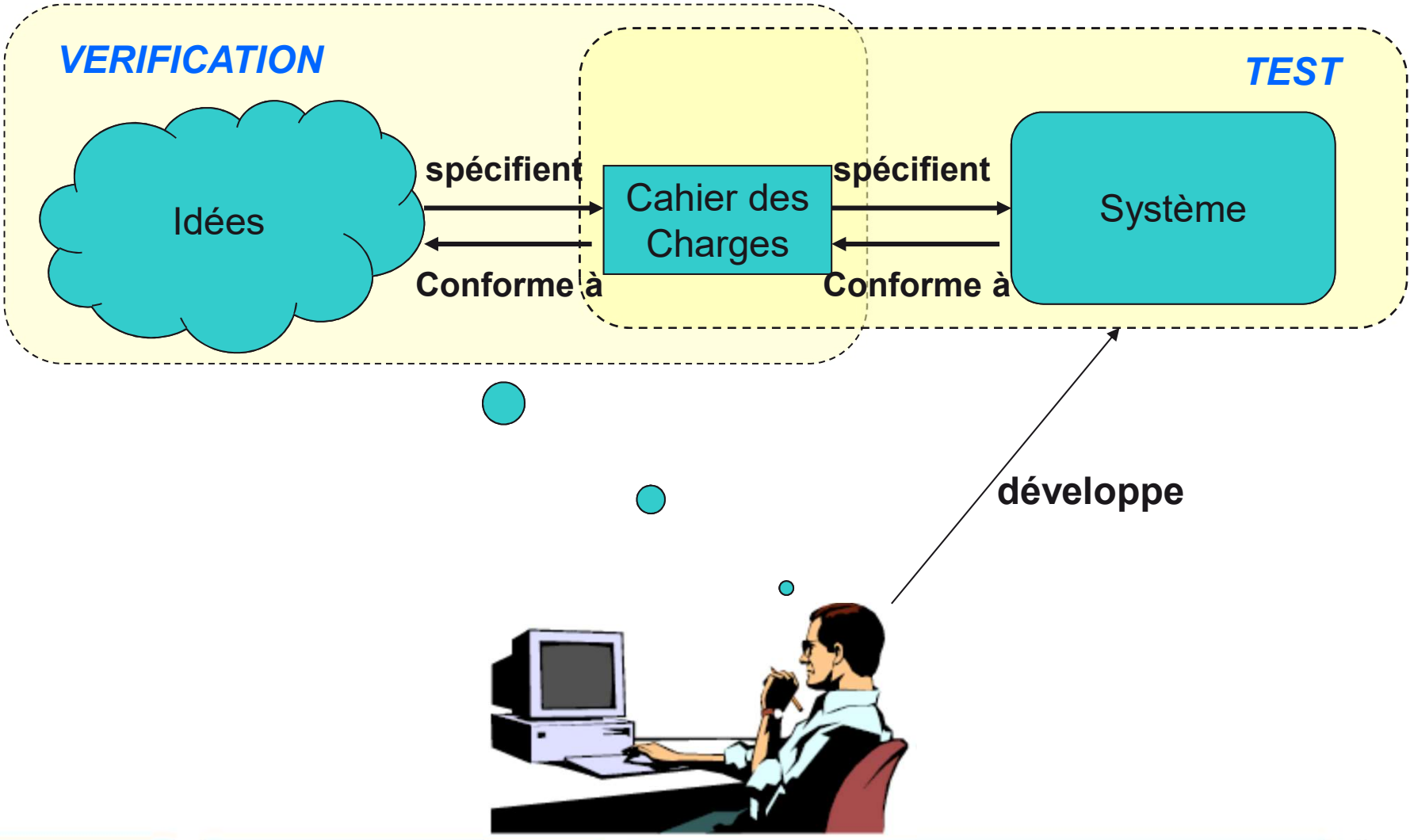
Idées et Systèmes - Revisités



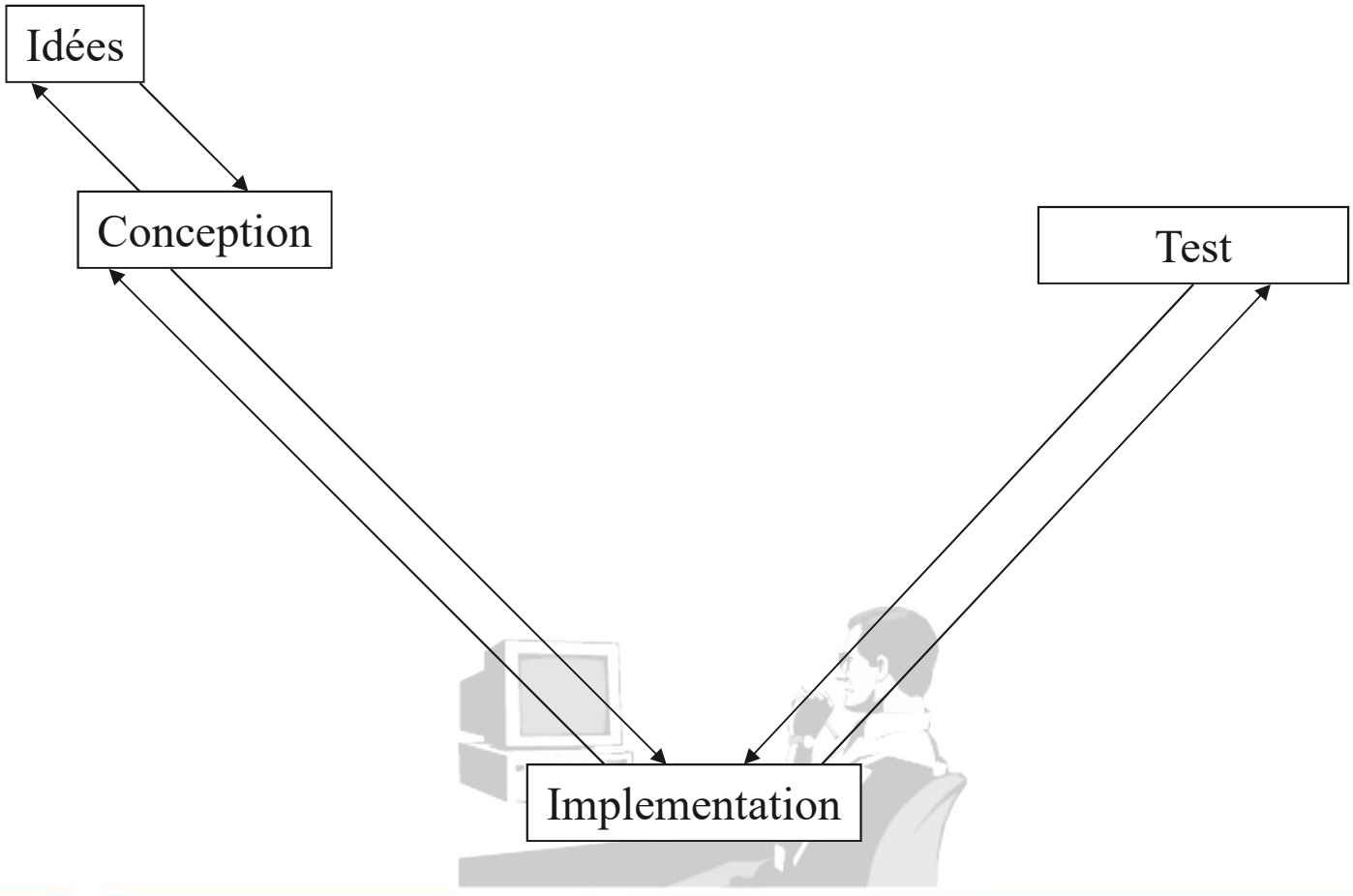
Idées et Systèmes - Revisités



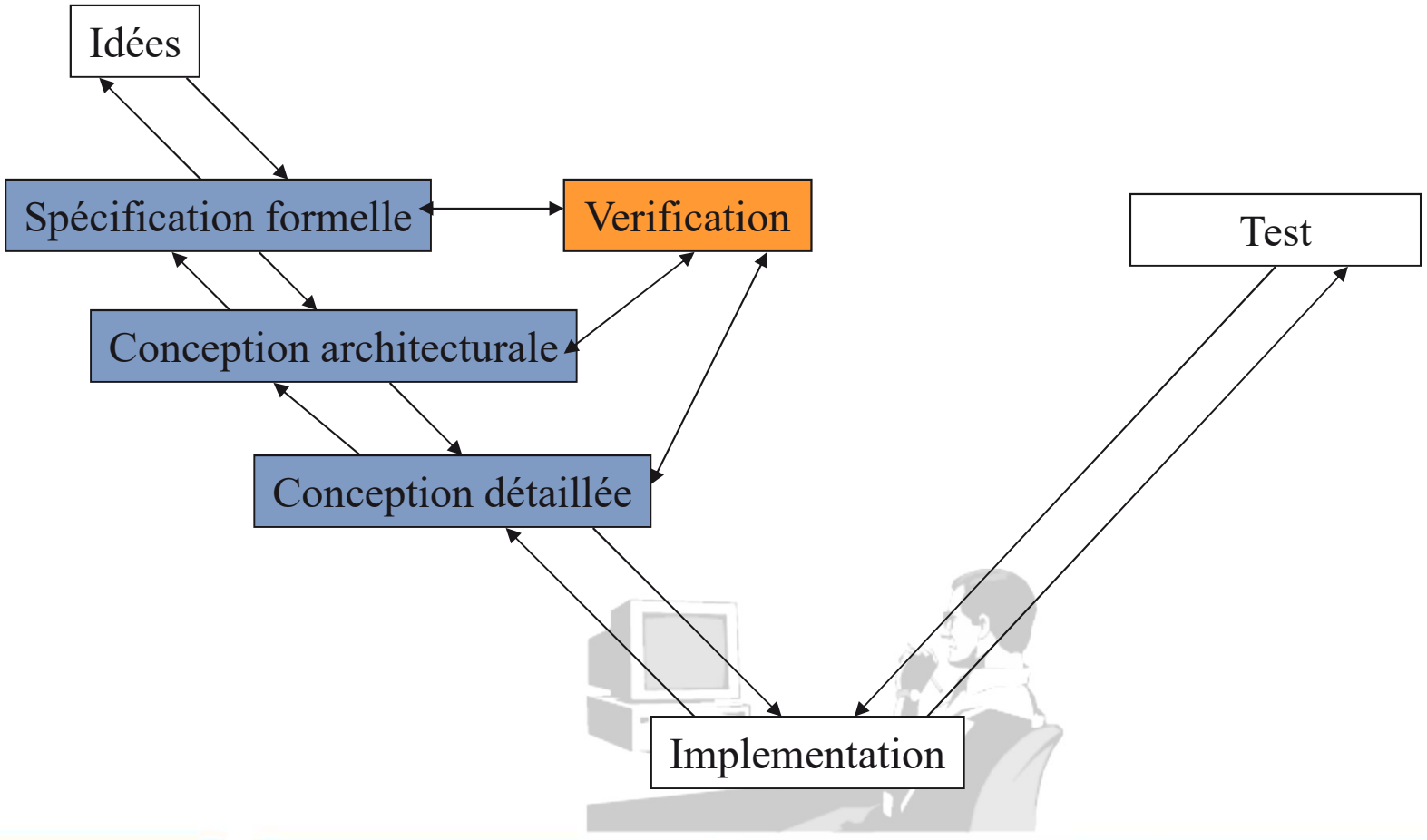
Idées et Systèmes - Revisités



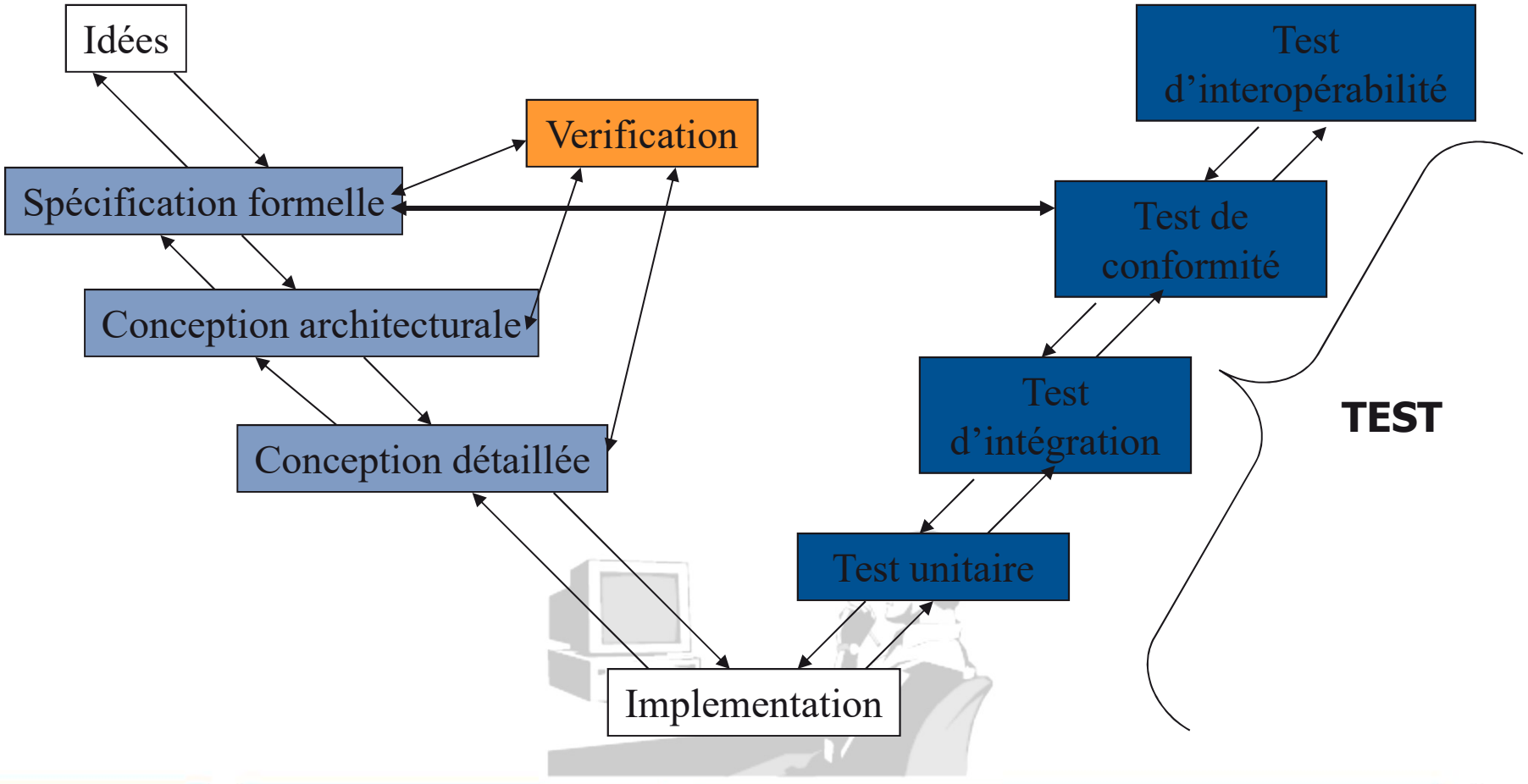
Cycle de vie en V



Cycle de vie en V



Cycle de vie en V



- Abstract State Machines (ASMs)
- A Computational Logic for Applicative Common Lisp (ACL2)
- Actor model
- Alloy
- ANSI/ISO C Specification Language (ACSL)
- Autonomic System Specification Language (ASSL)
- B-Method
- CADP
- Common Algebraic Specification Language (CASL)
- Esterel
- Java Modeling Language (JML)
- Knowledge Based Software Assistant (KBSA)
- Lustre
- mCRL2
- Perfect Developer
- Petri nets
- Predicative programming
- Process calculi
  - CSP
  - LOTOS
  - π-calculus
- RAISE
- Rebeca Modeling Language
- SPARK Ada
- Spec sharp (Spec#)
- Specification and Description Language
- TLA+
- USL
- VDM
  - VDM-SL
  - VDM++
- Z notation

Modèles formels

La plupart sont basés sur la sémantique des « state machines »

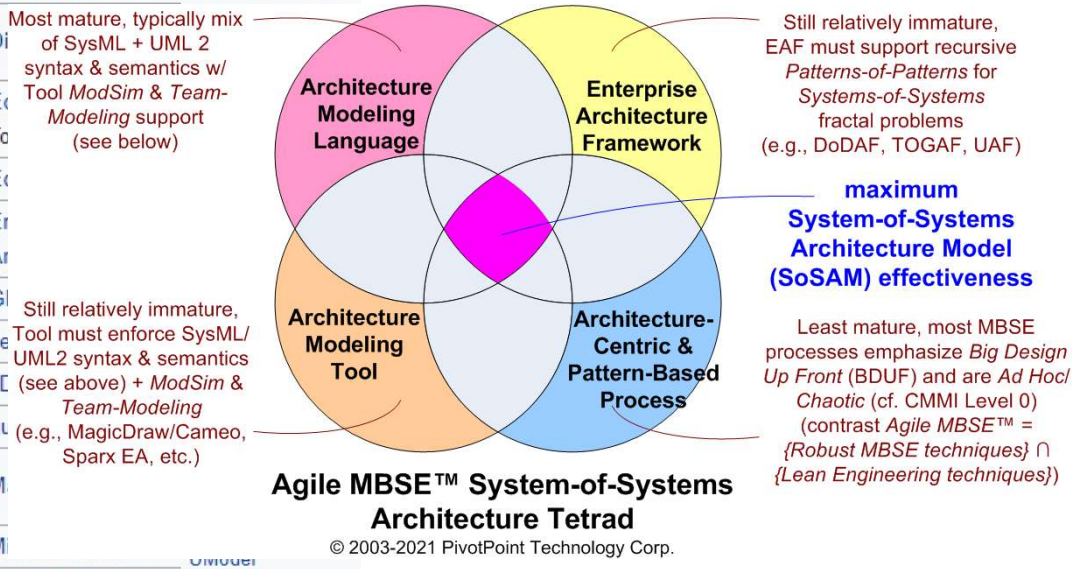
- FSM: X-Finite State Machines
- 2 types majeurs de FSM (en particulier pour le test):
  - Mealy machines
  - Moore machines
- Domaines pratiques:
  - Software Systems
  - Information Systems
  - Sequential Circuits
  - Communication protocols
  - Virtual networks / SDN
  - Etc.

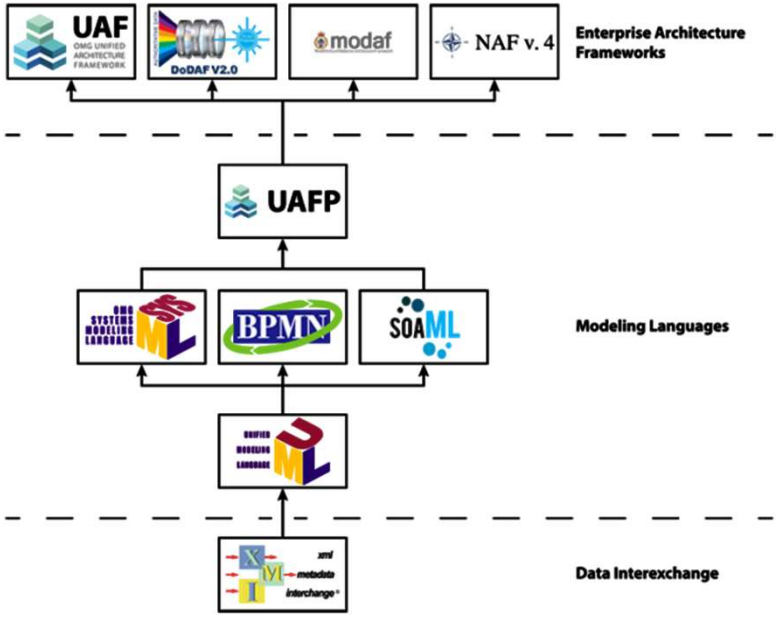
REF: Wikipedia

Modèles formes – de nombreux outils

Name	Name
ArgoUML	NetBeans[27]
Astah	Open ModelSphere
ATL	Papyrus
Micro Focus Together	PlantUML
	PowerDesigner
BOUML	PragmaDev Studio
	Prosa UML Modeller
Cacoo	
CaseComplete	Rational Rhapsody
ConceptDraw PRO	
Creately	Rational Rose XDE
Microsoft Visual Studio	Umple
Modelio	Visual Paradigm for UML
MyEclipse	WhiteStarUML
NClass	yEd

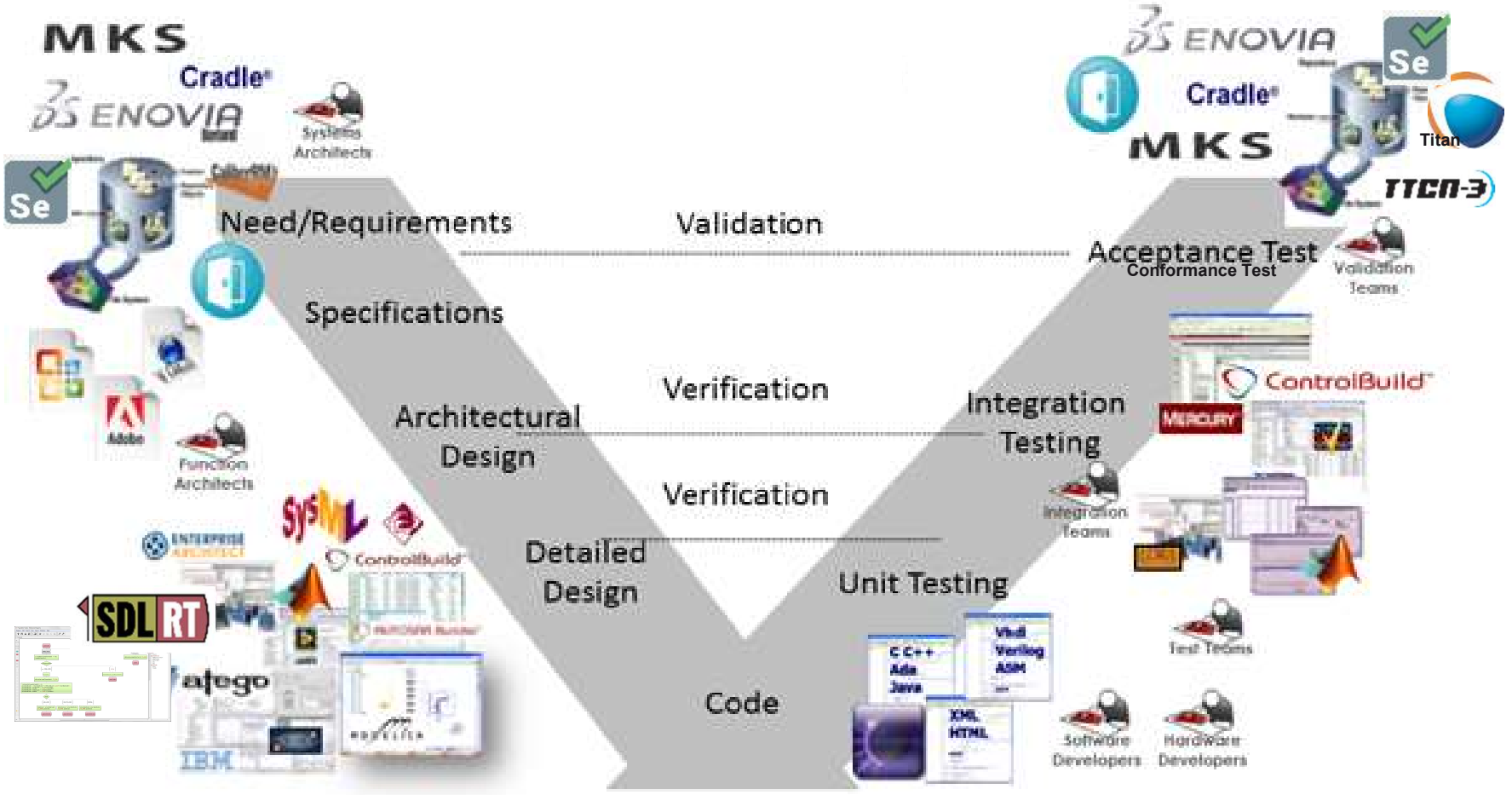
- **Ces outils sont fortement liés.**





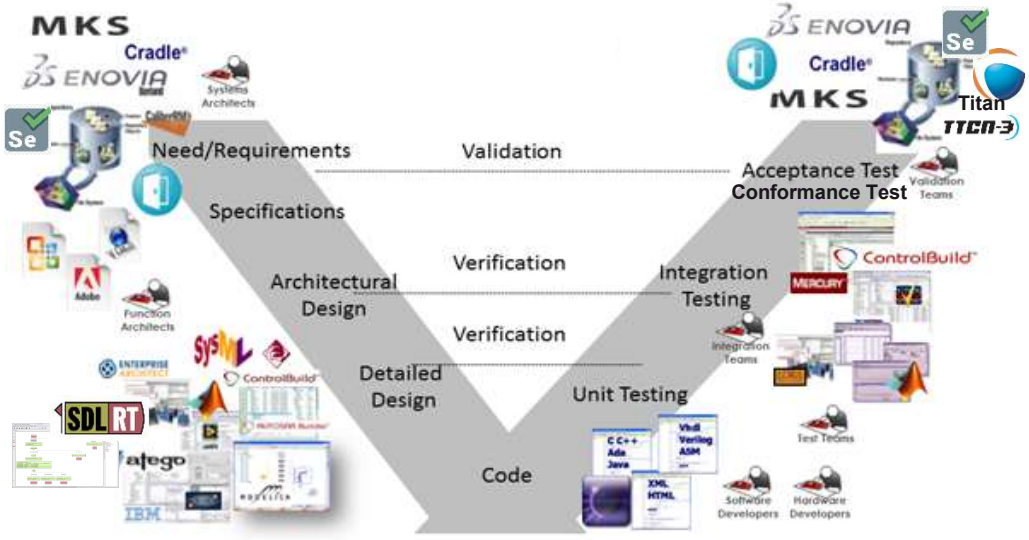
L'ingénierie des Protocoles et le test dans le cycle en V … ou en cycle agile …

Il existe des dizaines de modèles, méthodes et outils.



L'ingénierie des Protocoles et le test dans le cycle en V … ou en cycle agile …

Plusieurs types de test et définitions. Nous considèrerons:



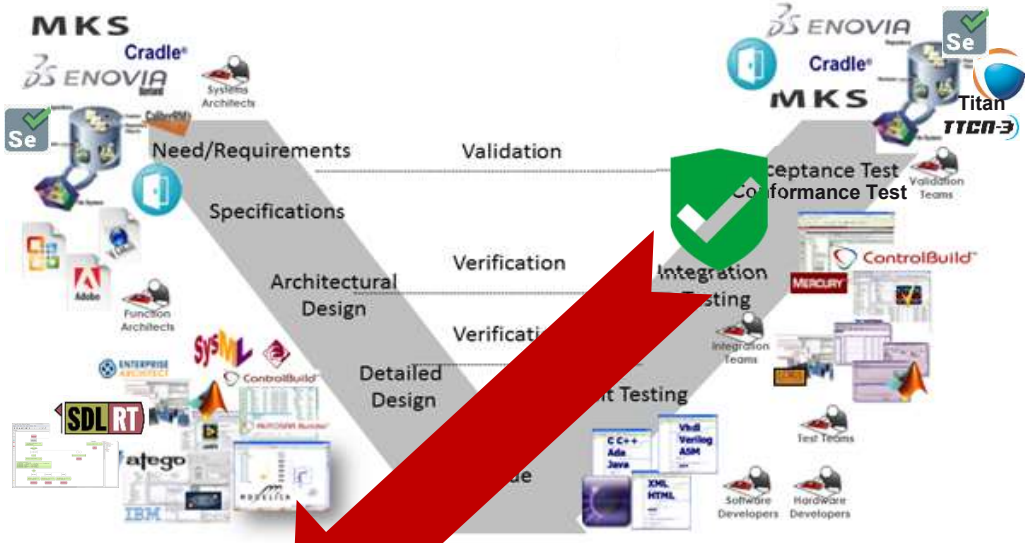
- **Unit testing**: *"tests written and run to ensure that a section of an application (known as the "unit" – e.g., a function, procedure, or even module or interface, class, etc. in sw. eng.) meets its design and behaves as intended."*

- **Integration testing**: "*phase in which individual modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified **functional** requirements*".
- **Acceptance/Conformance testing** (often named "conformity" in the standards): "*test to evaluate the adherence or non-adherence of a candidate implementation to a standard." May be defined as the fulfillment by a product, process or service of all relevant specified conformance **requirements**".*

Hamill, Paul (2004). Unit Test Frameworks: Tools for High-Quality Software Development. O'Reilly Media
ISO/IEC/IEEE International Standard - Systems and software engineering. ISO/IEC/IEEE 24765:2010
ISO/IEC DIS 10641
ISO/IEC TR 13233

Software System testing
in the V-cycle … or in an agile cycle …

Plusieurs types de test et définitions. Nous considèrerons:



- **Unit testing**: *"tests written and run to ensure that a section of an application (known as the "unit" – e.g., a function, procedure, or even module or interface, class, etc. in sw. eng.) meets its design and behaves as intended."*

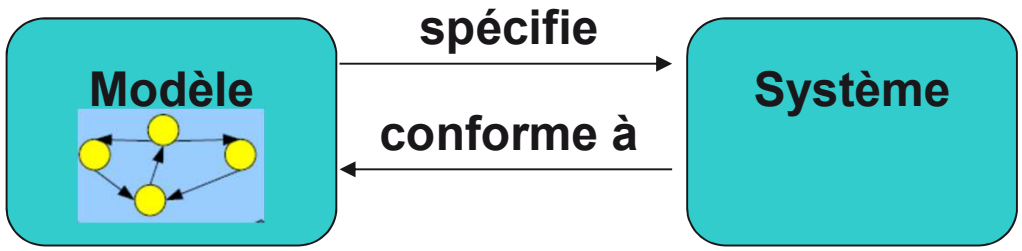
- **In**[illegible] *"phase in which individual [illegible] are combined and tested as a[illegible] n or c[illegible]*
- **A**[illegible] *o e[illegible] standard." May be defined as the fulfillment by a product, process or service of all relevant specified conformance* ***requirements****".*

SYSTEM TESTING: "*tester les comportements d'un système complet et intégré permettant d'évaluer la conformité de ce système selon ses exigences.*".

Hamill, Paul (2004). Unit Test Frameworks: Tools for High-Quality Software Development. O'Reilly Media
ISO/IEC/IEEE International Standard - Systems and software engineering. ISO/IEC/IEEE 24765:2010
ISO/IEC DIS 10641
ISO/IEC TR 13233
GB/T 11457 - 2006, Information technology - Software engineering terminology. 2006.

Techniques de spécification

- **Informelle (basée sur les langages naturels)**
- **Formelle**
  - Notations mathématiques



Techniques de Descriptions Formelles (FDT)

Pourquoi les FDT ?

- **Pour normaliser la communication**
  - Entre systèmes hétérogènes
  - Entre réseaux de télécommunication communicants
  - ITU et ISO: architectures OSI et TCP
  - Les normes (standards) contiennent:
    - Description des architectures (couches, services, protocoles) utilisant le langage naturel

    ⊕

    - Diagrammes de transitions avec tables d'états

    ⇒les spécifications contiennent des ambiguïtés

    ⇒FDT nécessaires

Langages de spécification

- **Ces techniques (aussi appelées *langage de spécification*):**
  - Spécifient les **propriétés fonctionnelles** (qualitatives) d'un système selon son environnement
  - Sont conçues pour décrire des systèmes distribués composés de processus exécutés en parallèle, se synchronisant et communicants par messages.

- **D'autres techniques: algèbres de processus (CCS), machines à états finis, logique temporelle, réseaux de Petri, ...**

ISO, ITU et IEEE normalisent 4 FDTs: VHDL, Estelle, Lotos, … et SDL

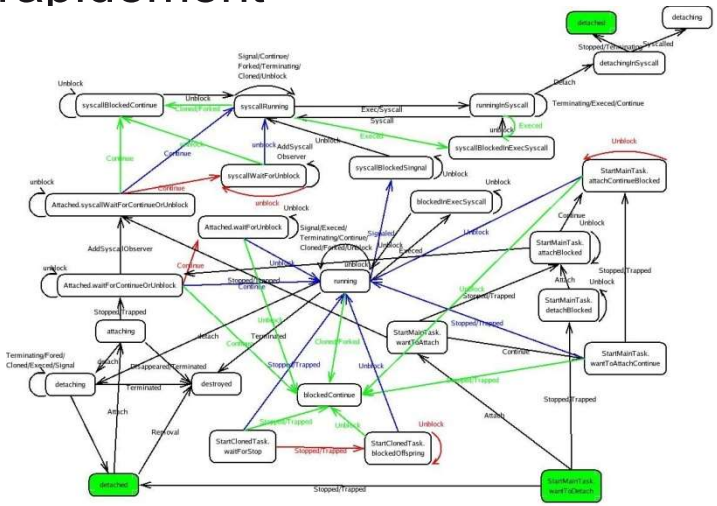
- **VHDL: normalisé par IEEE (1987, 1992, 2002, 2006, 2008 v4)**
  - Basé sur les Machines à états finis, synchrone, pour le Hardware
- **Estelle: normalisé par ISO (1989)**
  - Basé sur les Machines à états finis, synchrone
  - Type de données: PASCAL
- **Lotos: normalisé par ISO (1989)**
  - basé sur les algèbres de processus (CCS, CSP), asynchrone
  - Types de données abstrait
  - 2 visions: graphic et textuelle
- **SysML – UML2 … Mais semi-formel! (INCOSE-OMG, 2001)**
- **SDL: vous connaissez!**

SDL

- **SDL (Specification Description Language):**
  - Z.100, défini et normalisé par ITU(-T) (1988, 1992, 1996, 2000/2004/2009)
  - Basé sur les machines à états finis étendues (EFSM), asynchrone
  - 2 visions: SDL-GR (graphique) et SDL-PR (textuel)
  - Types de données abstraits, ASN.1
  - Profile UML – extension ITU-T Z.109

Avantages des FDT

- **Définitions stables et standardisées (consensus international)**
- **Évolution contrôlée**
- **Applications à des systèmes réels grandes échelles**
- **Communauté importante d'utilisateurs**
- **Réduit le coût du développement logiciel:**
  - Correction rapide des erreurs – agir rapidement



Avantages des FDT

- **Pas d'ambiguïtés**
- **Précis**
- **Cohérent, pas de contradictions**
- **Abstraction:**
  - Implémentations indépendantes
  - FDT ≠ langage de programmation

$$s_0 = \bot$$
$$s_1 = \beta \vee (\alpha \wedge \mathbf{EX}\overbrace{\bot}^{s_0}) = \beta$$
$$s_2 = \beta \vee (\alpha \wedge \mathbf{EX}\overbrace{\beta}^{s_1})$$
$$s_3 = \beta \vee (\alpha \wedge \mathbf{EX}\overbrace{(\beta \vee (\alpha \wedge \mathbf{EX}\beta))}^{s_2})$$
$$\ldots$$

Qui est intéressé par les FDTs ??!

- **Industriels:**
  - L'influence des systèmes OSI parmi les industriels (majors/PME) augmente
    - Airbus, Matra-Nortel, Orange, Bouygues, SFR, CISCO, SIEMENS, EADS, Renault, Daimler, Google, IBM, Microsoft, …
- **Les universités et centre de recherches**
  - Intérêt pour les systèmes distribués
  - Étude de la QoS (prop. Quantitatives), QoE, QoBiz
  - Pour définir des langages et modèles dédiés (Manet, Vanet, IoT, Web systems, …)

Les outils de et pour l'industrie

- **SDL et TTCN3**
  - Sema Group
- **ObjectGEODE**
  - Verilog-Telelogic
- **Tau2**
  - Telelogic / IBM
- **PRISM**
  - Oxford
- **SPIN**
  - Bell-Labs
- **etc.**

Que font ces outils ?

- **Éditeur graphique et syntaxique**
- **Parser/lexer**
- **Vérificateur**
  - Trouver les “bons comportements”
- **Simulateur**
  - de la spécification
- **Génération de code (C, C++)**

- **Générateur de tests**
- **Exécution de TTCN3**

Validation : Verification *plus* Test

- Validation : **un processus incluant la** Vérification **des spécifications ⊕ procédures de** Test**.**

- **Nombreuses techniques formelles/informelles de validation.**

Verification **de la specification**

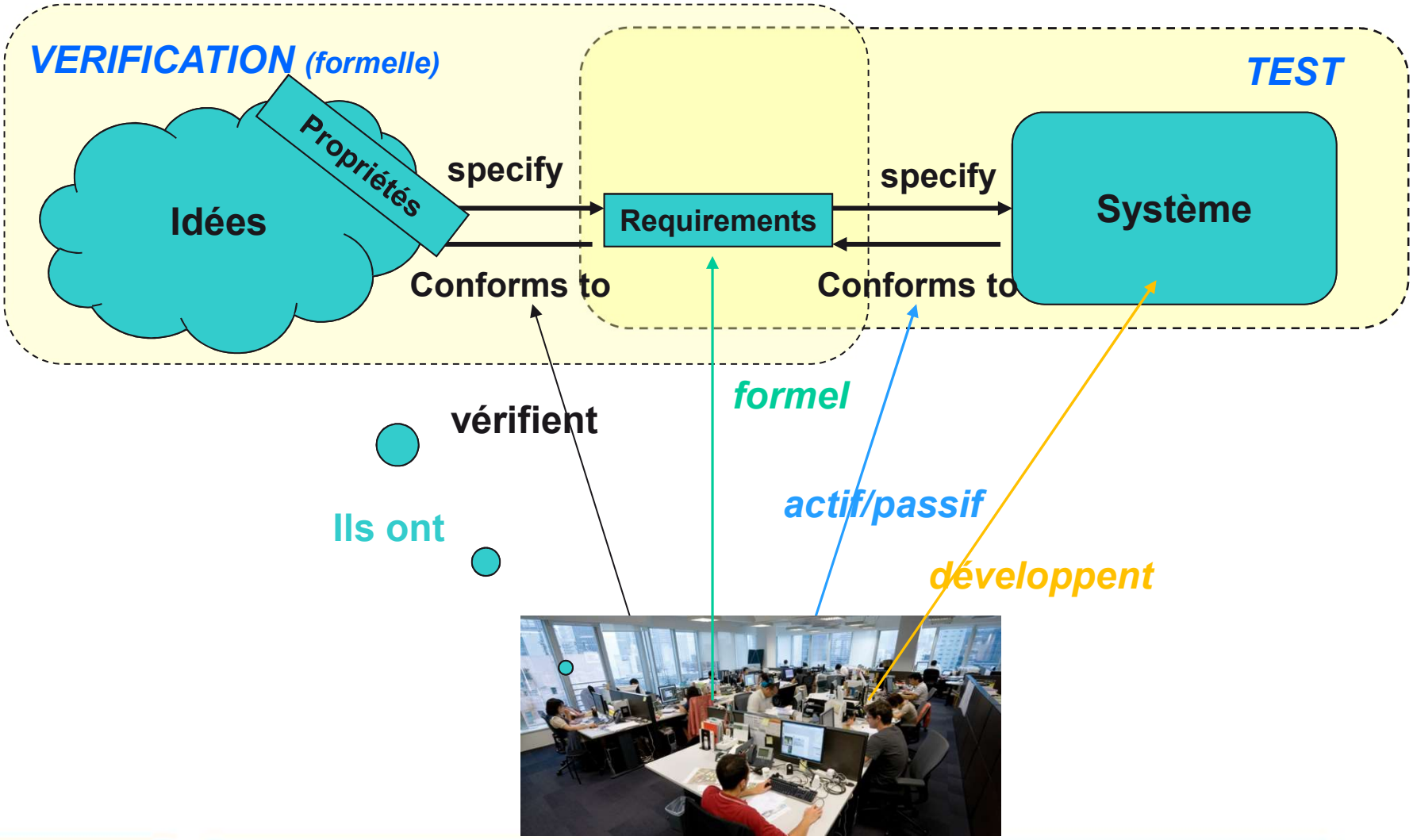
- **Vous avez plusieurs cours à ce sujet ☺**

Test des implémentations

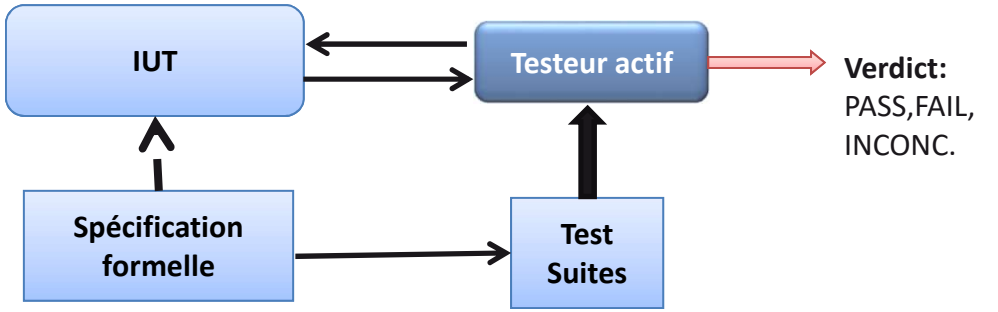
- **Nombreux travaux basés sur des simulations/émulations – approches informelles – essentiellement pour les prop. Quantitatives / QoS**
- **Approches formelles: relations d'équivalences entre**
  - Spécification formelle / IUT
- **Idée:** *générer des séquences de test à partir de spec puis les exécuter sur l'IUT*
- **Spécification souvent basée sur des modèles d'automates**
- **Obtention d'une couverture de test maximale recherchée – rarement atteinte – concessions nécessaires**
- **Essentiellement pour prop. Qualitatives**
- **Nécessite une architecture de test pour l'exécution des séquences de tests**
- **Pour prop. Quantitatives: modèles (stochastiques) labellisés par des intervalles de valeurs + Prédicats – e.g. réseaux de Petri**

- Passive / Active testing.

FDT pour le Test

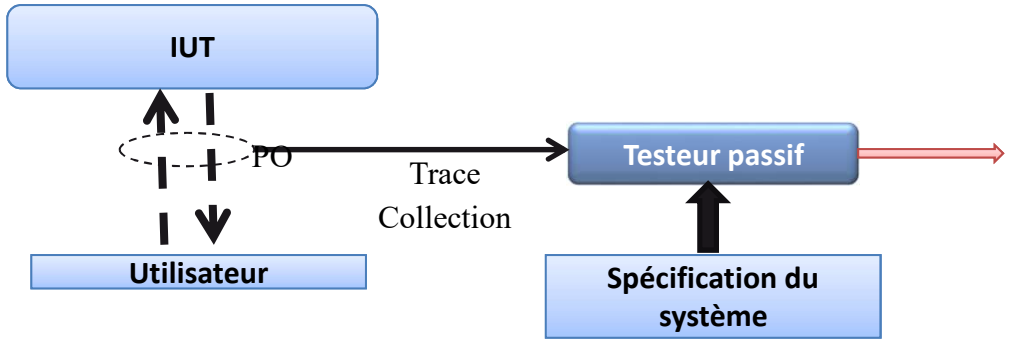


Test Passif vs Test Actif



pros et cons :

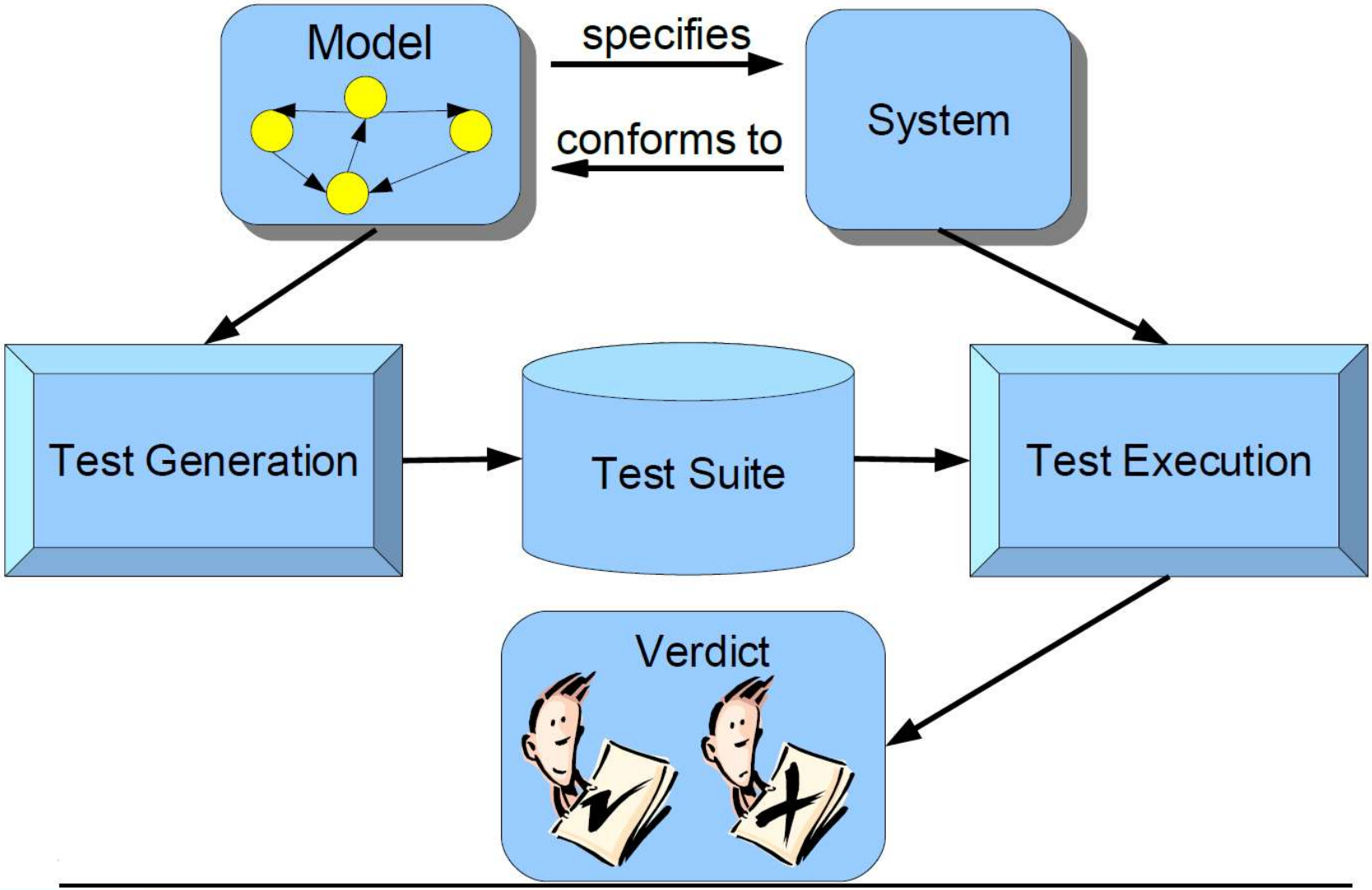
- ☺ Possibilité de s'intéresser à une partie précise de la spécification
- ☹ Génération automatique des tests
- ☹ Peut altérer (crash) le fonctionnement de l'IUT



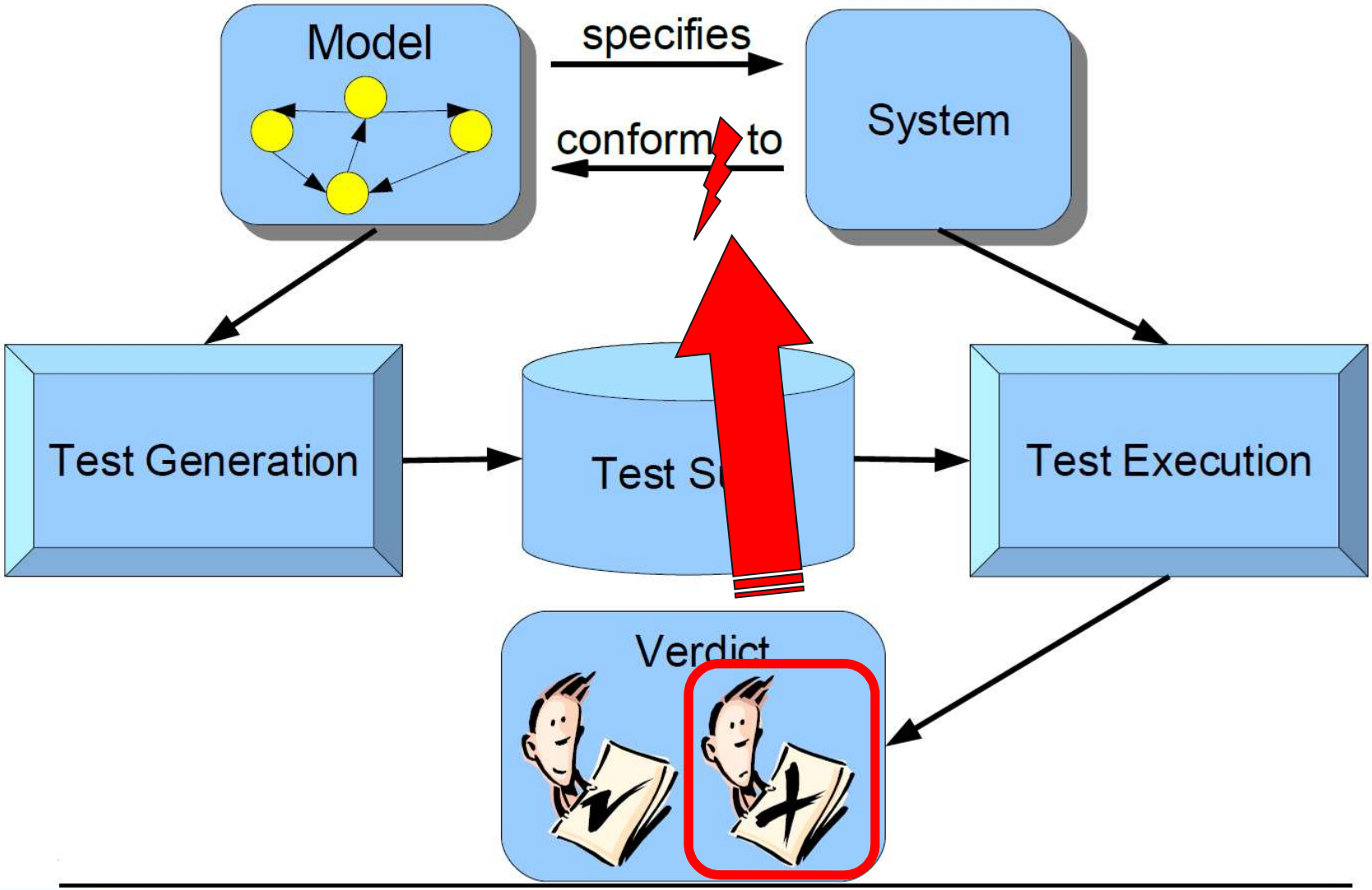
pros et cons :

- ☺ pas d'interférences avec l'IUT
- ☺ pas de génération des tests
- ☹ efficacité des algorithmes

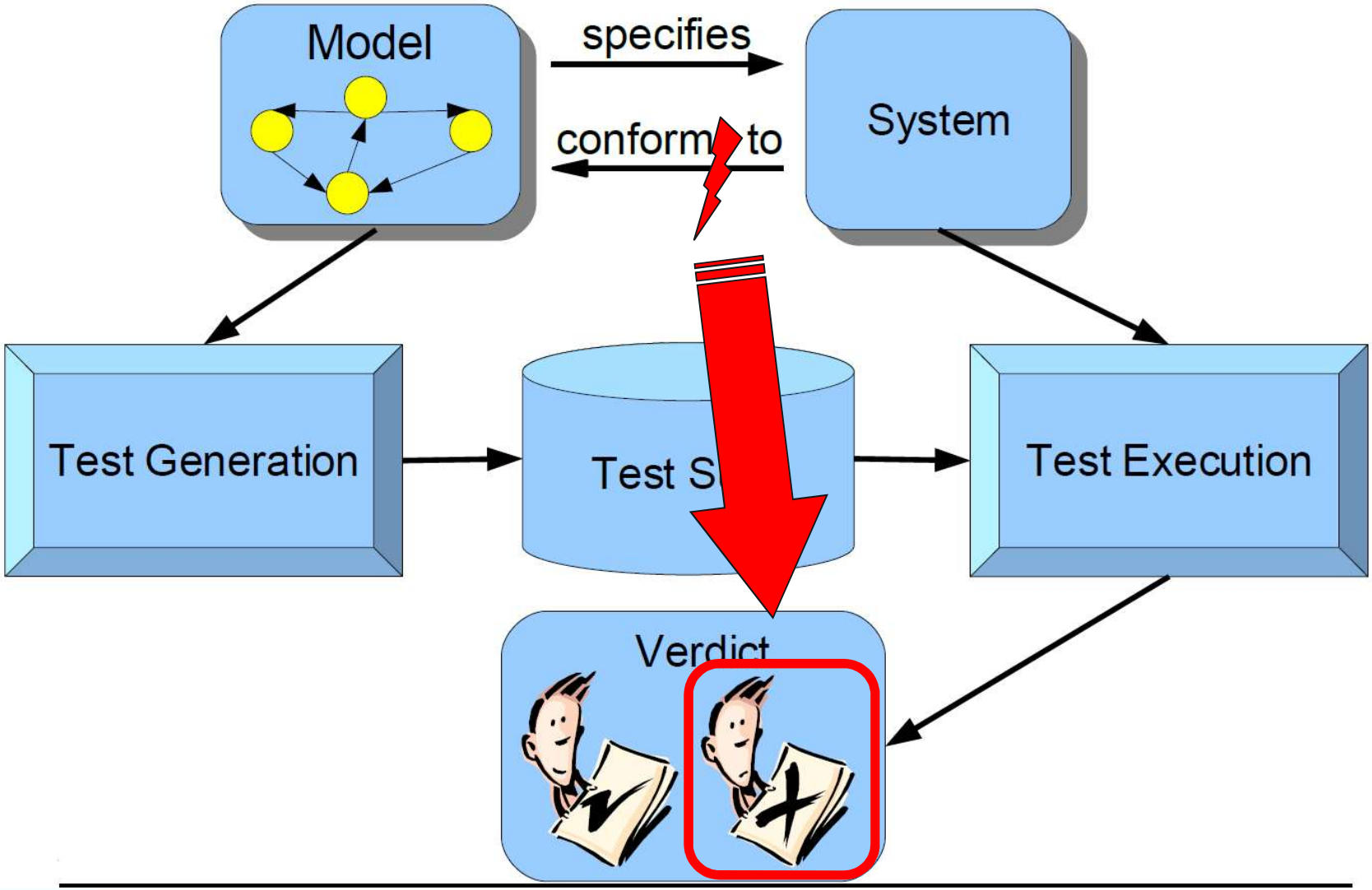
Active Testing



Soundness of Conformance Testing



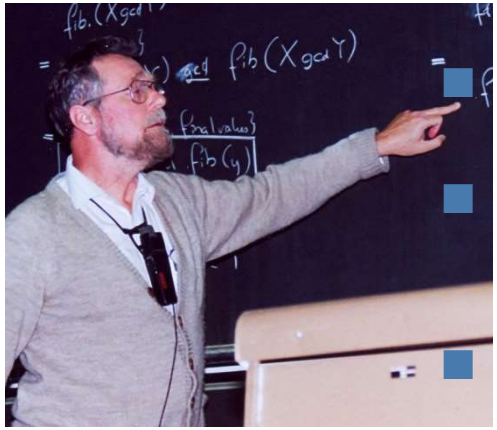
Completness of Conformance Testing



Soundness and Completness

- A test system, which always says  is sound.
- A test system, which always says  is complete.
- We want test systems that are **sound and complete**!
- But … someone told …

Soundness and Completness



- "*Testing can never be sound and complete*", dixit Dijkstra … of course, he is right (of course!).
- He refers to the fact, that the number of test cases in a sound and complete test suite is usually infinite (or at least too big).
- If that would not be the case, testing could prove the conformity of the system to the model (given some assumptions on the system).
- In practice, conformity is usually only semi-decidable by finding a failure with a (sound) test system.

- But still: completeness is a crucial property of a sound test system stating that it can *potentially* find all failures!
  ➔ **theoretically possible, but most of the time impossible in practice!**

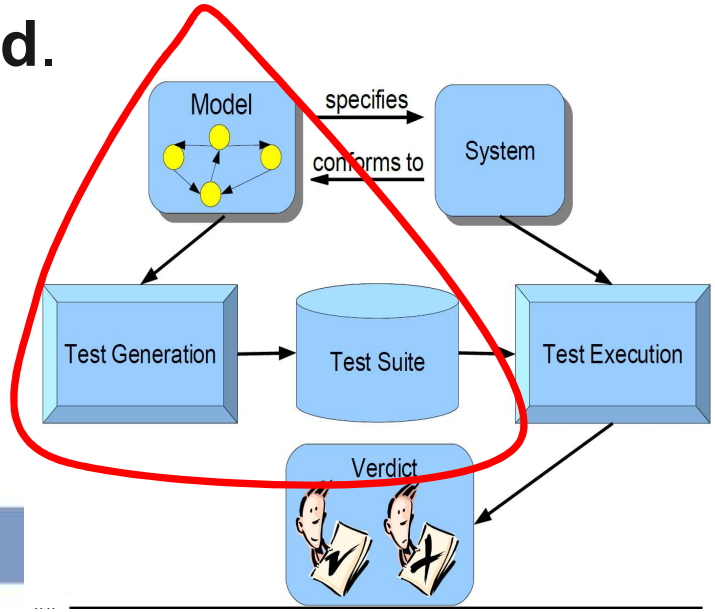
An *all-inclusive* definition of Active Testing

- **Protocol testing consists in:**
  - Dynamic verification of its behavior…
  - … According to a finite set of test cases ...
  - … Aptly selected from an input domain (in practice infinite) ...
  - … This compared to the specified expected behavior.

Automatic test cases generation

- For many years, several generation techniques! [Maag2010]
  - W, Wp, HSI, UIOv, DS, H, …
  - Many tools: TGV, Conformiq, JST, SmartTester, …
  - What about the **coverage test criteria**?
  - Outputs are test cases that are most of the time abstract ➔ need to be **concretized**.
  - One common notation: TTCN3

(+ETSI TDL)

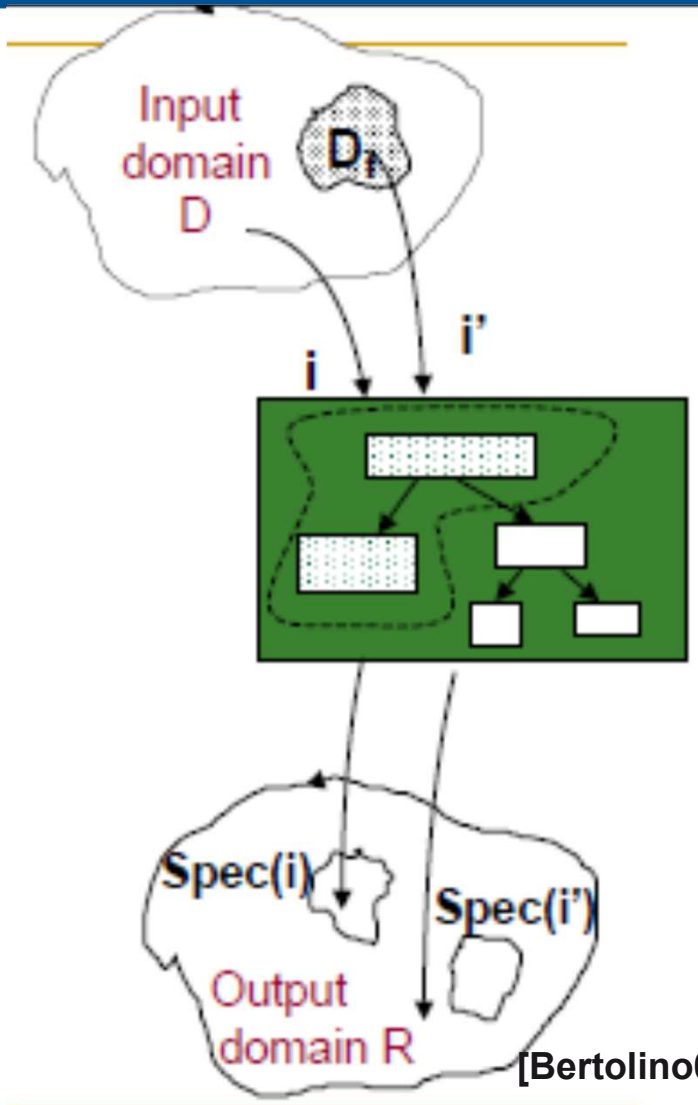


Exécution des Tests

et couverture

Comment appliquer le Test ?

- **Les Systèmes deviennent de plus en plus larges, complexes, distribués, hétérogènes, …**
  - Comment partitionner le système en parties *testables?*
  - Comment organiser le processus *(who, what, how)?*
  - Quel test pertinent pour une étape spécifique?
  - *Evergreen:* et le test de non regression dans tout ça?!



[Bertolino07]

Un débat épineux !

- **Idéalement, un critère de test:**

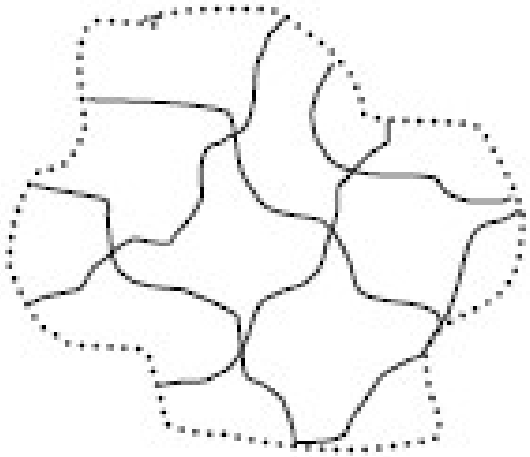
  C (Protocol, Modèle, Test set)

  Induit une subdivision du domaine d'inputs en classes disjointes regroupant les comportements équivalents:
  - Dans ce cas, un test dans chaque partition est suffisant pour contrôler le comportement du tout. (hypothèse de test)
- **En pratique:**
  - Sous-domaines superposés
  - Comportements non homogènes.

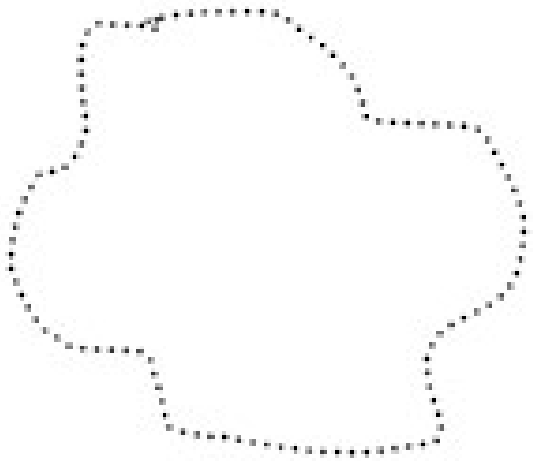
⇒ **Test Partitionné (i.e. systématique) vs. Test aléatoire (i.e.uniforme)**

Test Partitionné vs. Aléatoire

Test Partitionné vs. Aléatoire



$P = 1 - \prod (1 - \theta_j)$

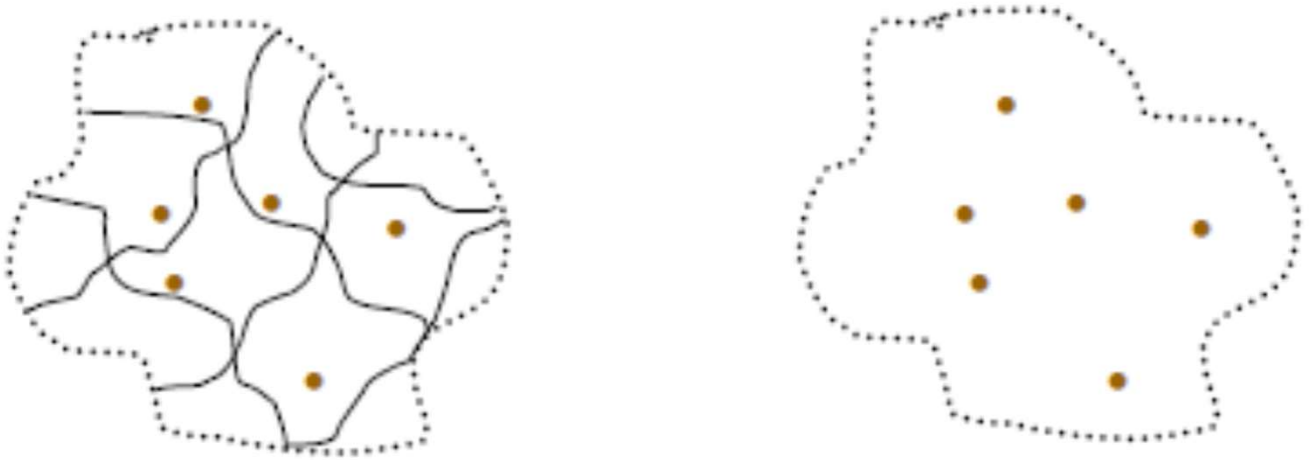


$P = 1 - \prod (1 - \theta_j)^k$

Où, pour une comparaison équitable, le nombre de cas de test est le même dans les 2 cas.

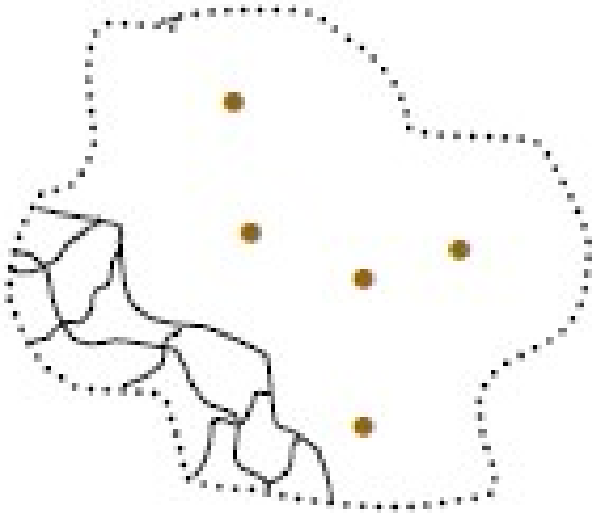
P = probabilité de la détection d'une faute
θ = taux d'erreurs obtenus par les inputs dans une partition j
K = le nombre d'inputs ayant été utilisés pour les tests.

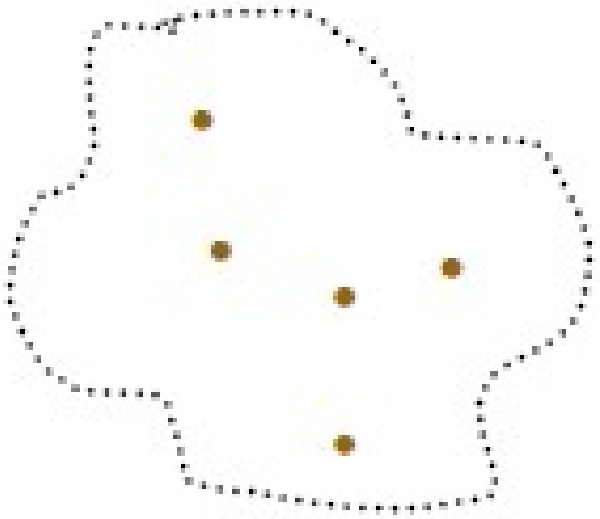
Test Partitionné vs. Aléatoire



Les deux cas sont équivalents.

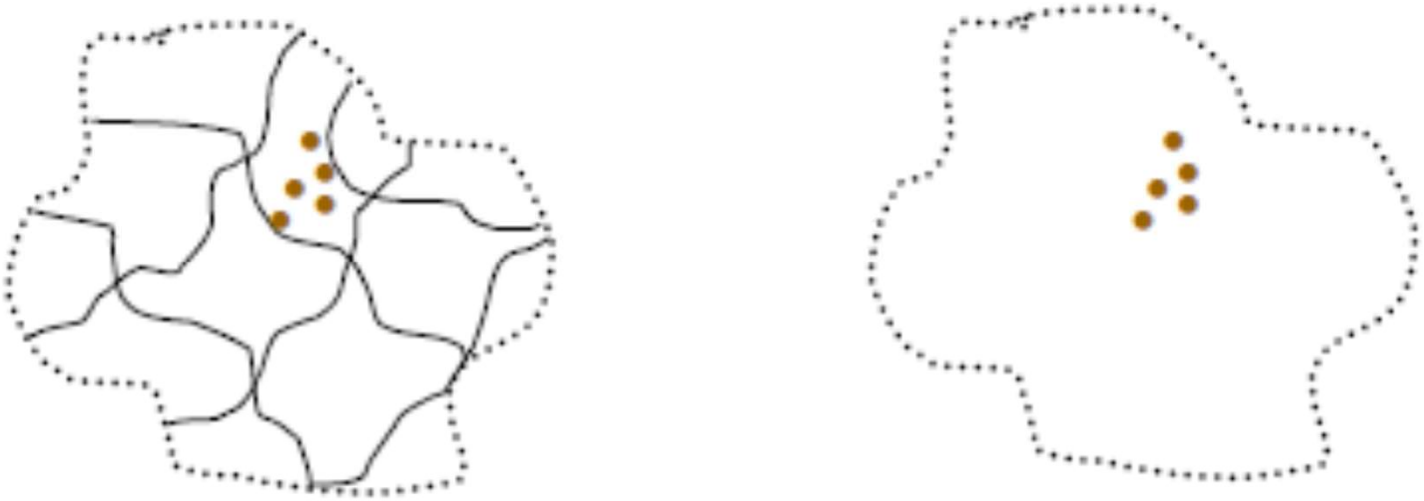
Test Partitionné vs. Aléatoire





L'Aléatoire est meilleur

Test Partitionné vs. Aléatoire



Le partitionné est meilleur

Critère de Test – *pour quoi faire ?*

C (Protocol, Modèle, Test set)

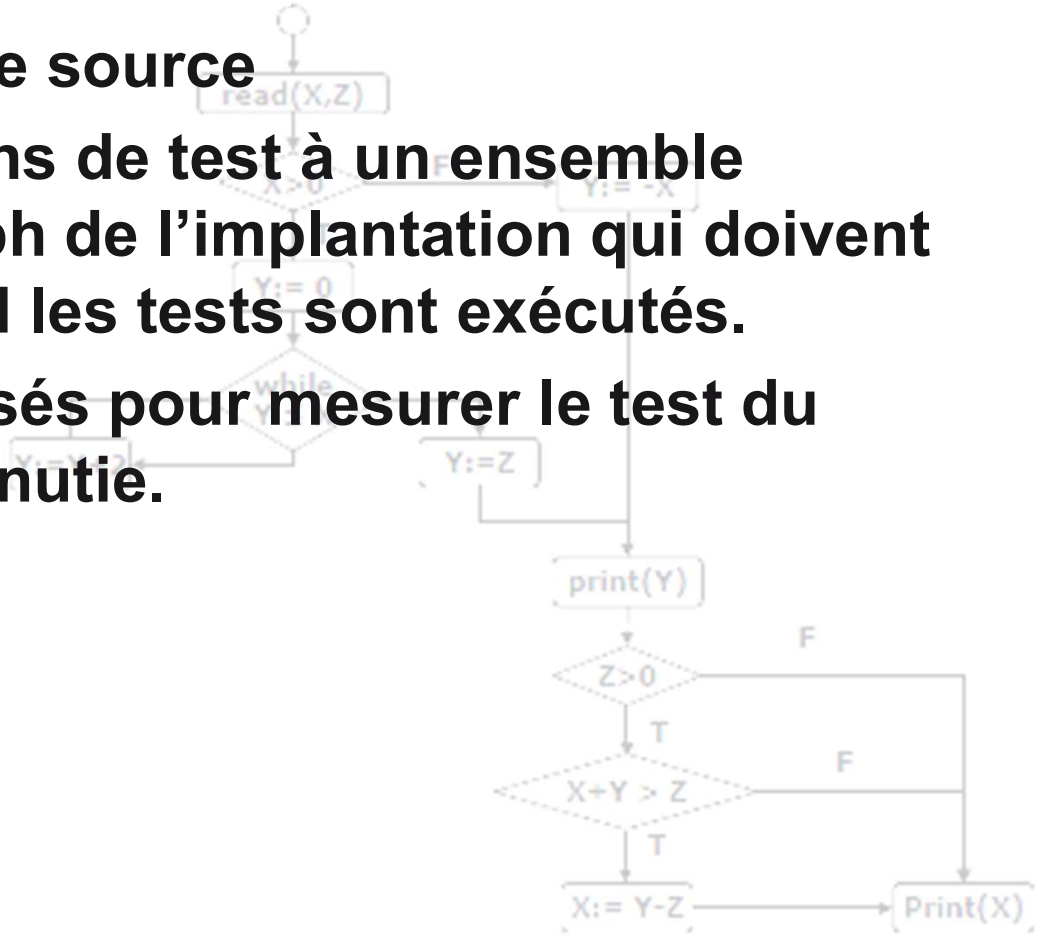
- **Pour dériver le Test set**
  - Génération/sélection des tests (automatisé)
- **Pour décider si un Test set est suffisant**
  - Règle de stoppage / critère d'adéquation

White Box et Black Box

- **Deux grandes familles de critères de sélection des tests:**
  - Test Boite blanche: analyse par mesures du code source (notion de couverture)
  - Test Boite noire: analyse d'un protocole à partir de son modèle (formel) ou du cahier des charges
- **Deux familles souvent utilisées de paire**
- **Recherche ouverte: « Real White-box Testing » visant à utiliser le code source pour guider automatiquement les procédures de test.**

Critères de Couverture de Test

- Appliqués sur le code source
- Associent des besoins de test à un ensemble d'entités du flowgraph de l'implantation qui doivent être couvertes quand les tests sont exécutés.
- Essentiellement utilisés pour mesurer le test du code source avec minutie.



Critères de Couverture de Test

- La Couverture est une mesure de complétude
- Une couverture de 100% ne signifie jamais « test complet » mais seulement la complétude en ce qui concerne la stratégie sélectionnée.
- ∃ de nombreuses stratégies et métriques de couverture
- Pas de « meilleure » mais certaines meilleures que d'autres selon les cas.

Ex.: Critère de Couverture de Branches

- Nécessite que chaque branche de l'implantation soit exécutée par au moins un cas de test
- Une suite de test T satisfait le critère de Branche pour une implantation I ssi pour chaque branche B de I, ∃ un cas de test dans T qui cause l'exécution de B.

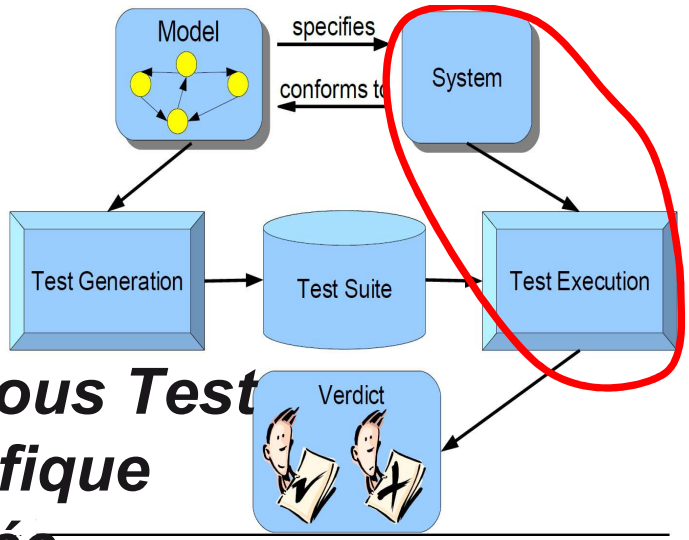
- NB: la couverture de branches n'est pas garantie par la couverture des états.
- NB: couverture de branches obligatoire dans le test unitaire

Concept de *Traçabilité - Concrétisation*

- **Objectif:** ***Lier les valeurs abstraites de la spécification aux valeurs concrètes de l'implantation.***

- **Les cas de test synthétisés décrivent des séquences d'actions qui ont une interprétation au niveau abstrait de la spécification.**
- **Pour pouvoir exécuter ces test sur l'implantation, on doit *concrétiser* ces tests en termes d'exécution à travers l'interface I/O du système.**

L'Exécution



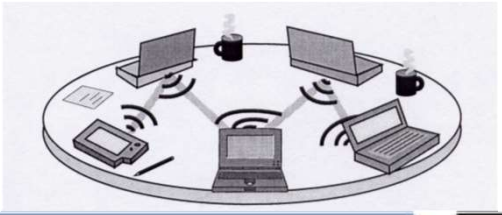
- **Objectif:** ***forcer l'Implémentation Sous Test (IUT) d'exécuter la séquence spécifique d'événements qui a été sélectionnée.***
- **Deux exigences:**
  - Mettre le système dans un état à partir duquel les tests spécifiés peuvent être lancés (pré-condition)
  - Reproduire la séquence désirée (connu sous le nom de Replay problem): problème très difficile, notamment en présence de processus concurrents et de non-déterminisme (i.e. même input, différents outputs!).

Quelques Exemples

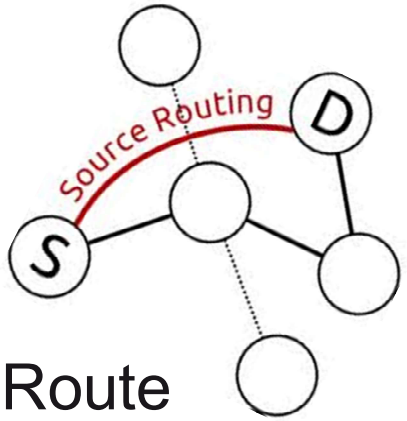
MANET - What's that ?

- **"*An mobile ad hoc network is a collection of wireless mobile hosts forming a temporary network without the aid of any established infrastructure or centralized administration*", Johnson et al., 1994**

- Infrastructure-less,
- Autonomous,
- The nodes behaves like routers,
- Multi-hops paths
- Dynamic topology (due to mobility or sleep mode),
- Energy constraints due to batteries,
- Heterogeneous radio communications (uni/bi-directional links, different interfaces),



DSR (Dynamic Source Routing) – RFC4728



- Reactive protocol
  - Unicast reactive routing protocol,
  - No routing table but Source Routing,
  - Two mechanisms: Route Discovery and Route Maintenance.

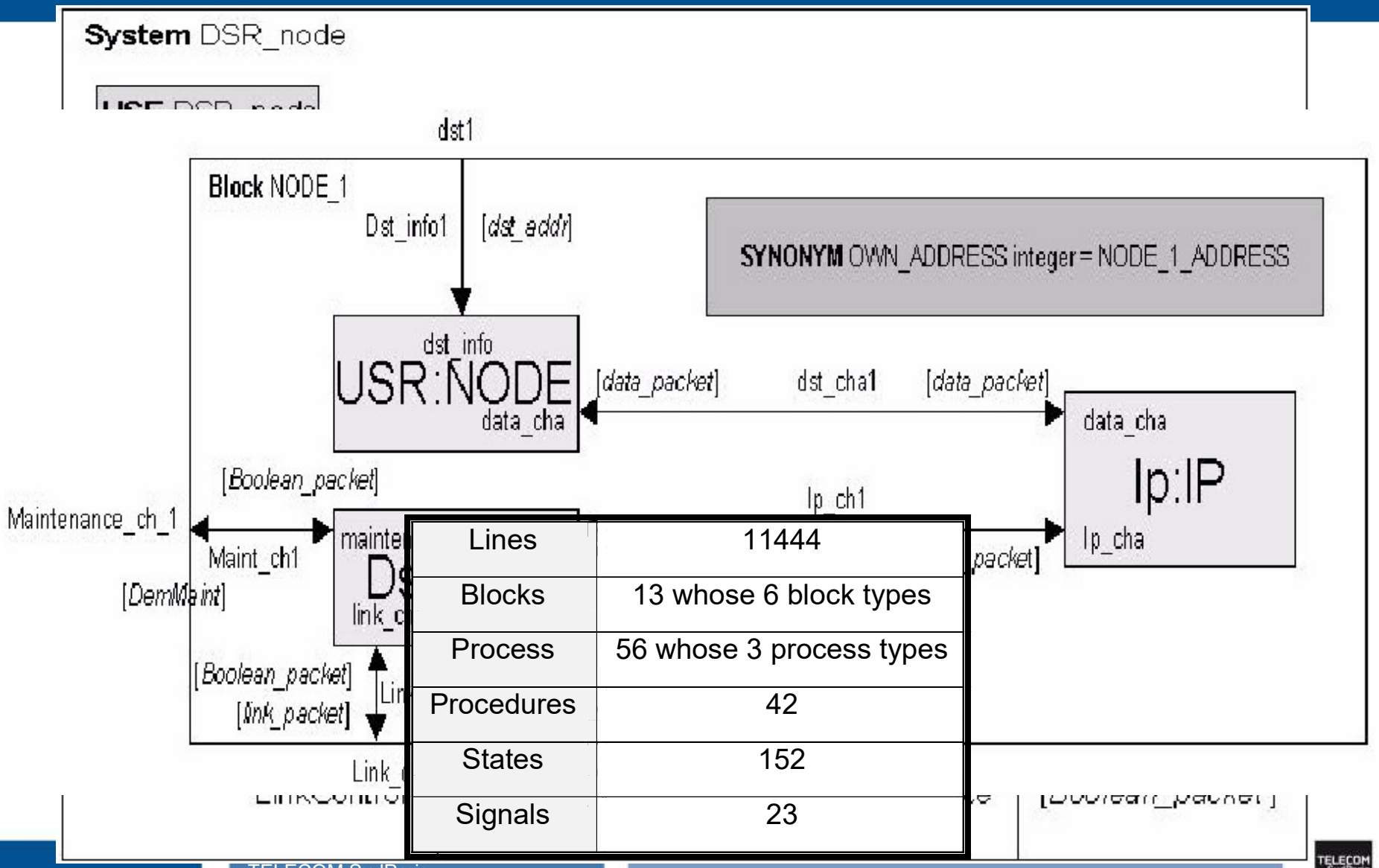
- **Our DSR implementation:**
  - DSR-UU-0.2 runs in the Linux kernel originally created at Uppsala University

DSR formal specification

- DSR formal model designed in SDL (Specification and Description Language – ITU-T Z.100).
  - EFSM based, allows to specify the system architecture, the functional behaviors.

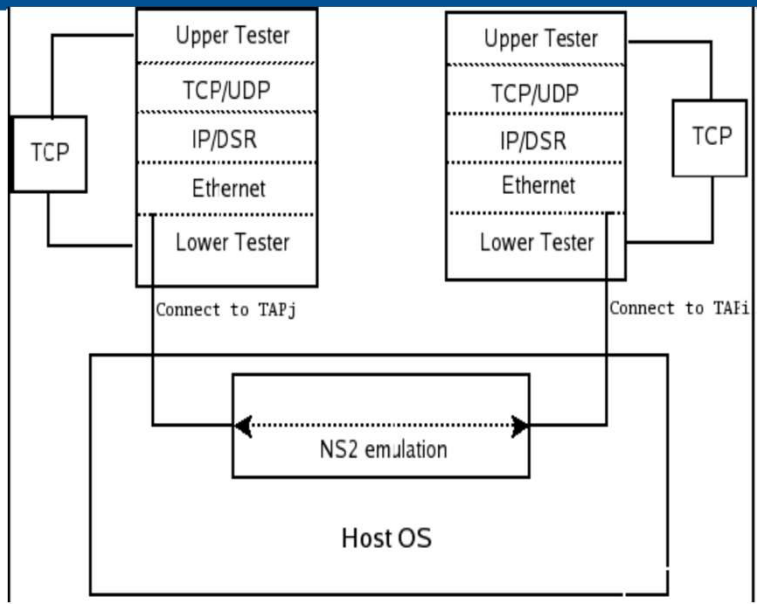
- Selection of the test purposes: from the DSR standard,
- Test sequences generation: from the specification and testing tools (TestGen-SDL).

DSR formal specification



Lines	11444
Blocks	13 whose 6 block types
Process	56 whose 3 process types
Procedures	42
States	152
Signals	23

DSR-UU Testing



- Testing assumptions:
  - The system could be tested,
  - Destination node exists,
  - Source-destination path connected,
  - Stable routes (to execute the tests),
  - Test scenarios may be replayed.

- TCP = Tester Coordination Procedure
  - as an Oracle developed in C.

- UML = User Mode Linux for NS-2 Emulator with its own DSR implementation
- Fedora Core 4.2.6.16 with virtual wireless network interfaces
- DSR-UU = IUT (Implementation Under Test)

- 22 GOAL test purposes → test seq. generation TESTGEN-SDL

➢ **RESULTS** with ≅50 test objectives
- No FAIL verdicts – ≈ 5% of PASS verdicts
- **≈ 95% INCONCLUSIVE verdicts**
- Too many packets loss, interferences, uncontrollability of the emulator (≠ specification), so many topological changes from the emulator !

DSR-UU Testing

- **Main reason:** unpredictable topological changes.
  - The formal model did not plan such changes, then not expected in the test sequences.

- **Our solution : the Nodes' Self-Similarity approach.**

Nodes' self-similarity

- **Nodes Self-Similarity (NSS) definition**
  - **Upon the formal model, composition of nodes that are functionally similar from the IUT view point.**

$N = \otimes_{i \in E} N_i$ $\{N_i\}_{i \in E}$ IOEFSM.

$O(N) = \cup_{i \in E} O(N_i)$

$I(N) = \cup_{i \in E} I(N_i) - \cup_{i \in E} O(N_i)$

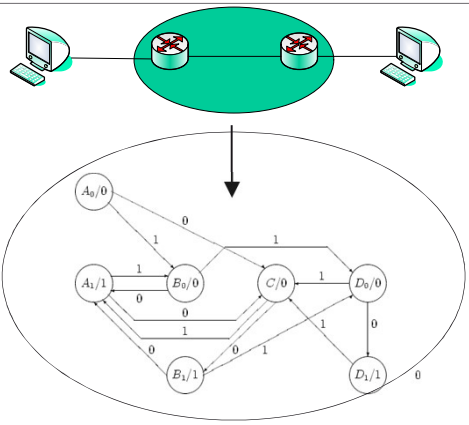
$S(N) = \prod_{i \in E} S(N_i)$

$x(N) = \prod_{i \in E} x(N_i)$

$Tr(ActHide_{\varphi} (N_1 \otimes N_2)) \subseteq Tr(N_3)$ φ = {1-hop exchanged messages between the N_i}

$\Leftrightarrow$

$N_1 \otimes N_2$ and N_3 are self-similar

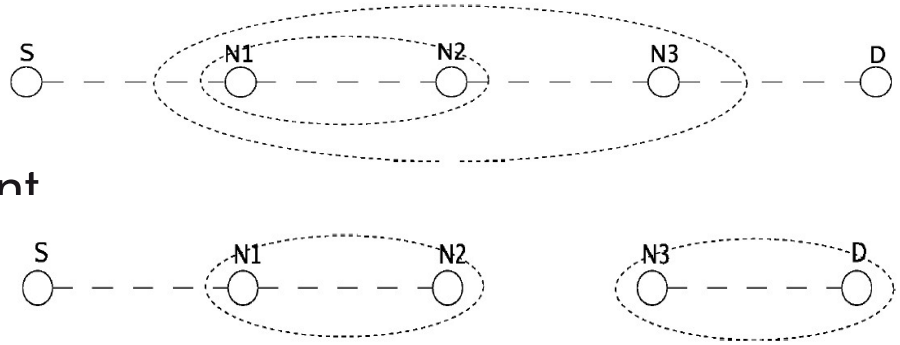


Objectives: To represent ***p*** real interconnected mobile nodes by ***q*** nodes formally modeled with ***q*** < ***p***.

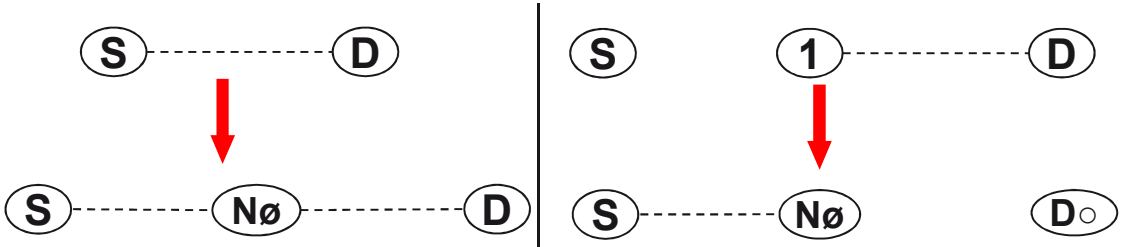
- Reduction of the combinatory explosion
- Reduction of inconclusive verdicts (minimal topology)

Nodes' self-similarity

- **Circumstantial NSS:**
  - Applied on the model
  - From the IUT view point
  - For a specific test obje



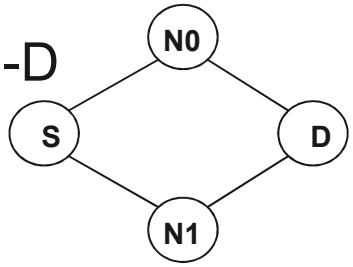
- **Exceptions**



$$N_\circ = \begin{cases} N_\varnothing & \text{where } N_\varnothing \text{ is the neutral element.} \\ N_x \circ N_{x+1} & \text{where } N_x \in \{N_\varnothing\} \cup \bigcup_{i=1}^{n} \{N_i\} \end{cases}$$

Nodes' self-similarity

- **NSS applied to our DSR specification.**
  - Three distinguished DSR elements: Source, Destination, other nodes N (routers)
  - By self similarity and according to test objectives plus the RFC: reduction of the model.
    - Two hops paths needed S-N-D
    - Two routes needed



NSS for Conformance testing

- Test Execution
  - In *Spec* : 2 routes from *S* to *D* with *4* nodes
  - In *IUT*: *n* routes from *S* to *D* with *p* nodes

  ⇒ TCP manager algorithm to allow the TCP to match similar routes from *Spec* with *IUT* ($O(n^2)$)

- Experimental results on DSRUU and the same test suite
  - **No FAIL verdicts – ≈ 95% of PASS verdicts**
  - **≈ 5% INCONCLUSIVE verdicts**

Open issues in active testing?

- What to do when:
  - Access to the interface unavailable
  - Unreachable component – UT/LT not allowed to be integrated
  - SUT cannot be interrupted
  - SUT cannot be stimulated
- When stop testing?
- What test cases selected?

  ⇒ How to manage the incompleteness of the practice test?

1. To accept and find heuristics such as coverage criteria, time constraints, *randomness,* test objectives, etc.
2. Ask other assumptions leading to the completeness practice.

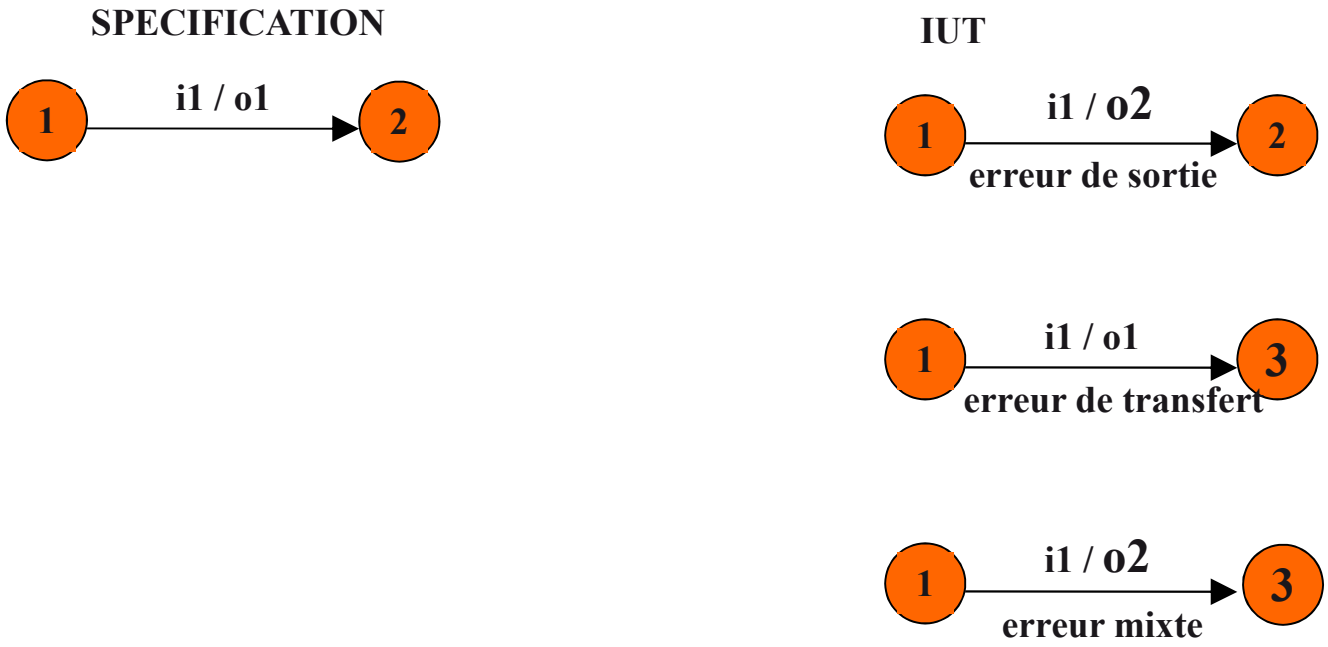
➔ Let's try **Passive Testing** …

Génération des suites de test

Test basé sur les « automates »

- **Hypothèses sur l'automate**
  - Complet (*input enable*)
  - Déterministe
  - Minimal??
  - *Strongly connected*
  - Pas d'états *transients*

Types d'erreurs dans une IUT
Faults model



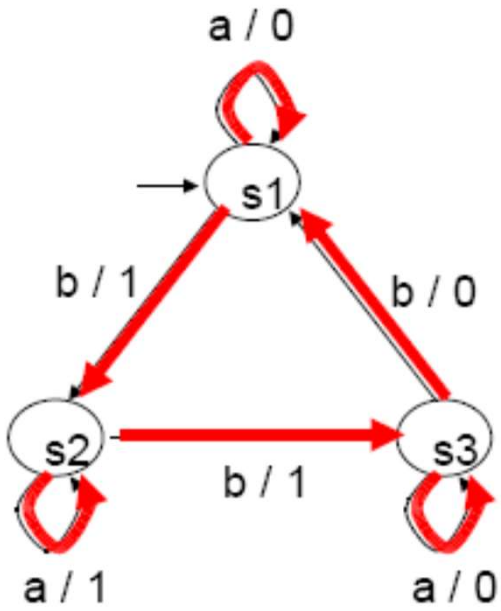
Test basé sur les « automates »

- **Erreurs de sortie**
  - Méthode TT (tour de transitions)

- **Erreurs de sortie + erreur de transfert**
  - DS (suites distinguantes)
  - W (ensembles discriminants)
  - UIO (suites uniques d'entrées/sorties)

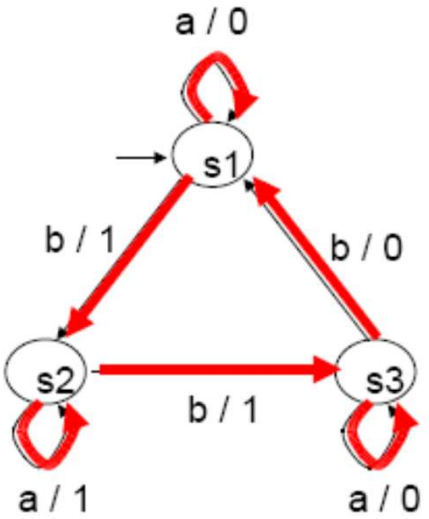
Méthode TT (1)

- Un tour de transition: chemin de l'état initial à l'état initial qui traverse toutes les transitions au moins une fois



Méthode TT (2)

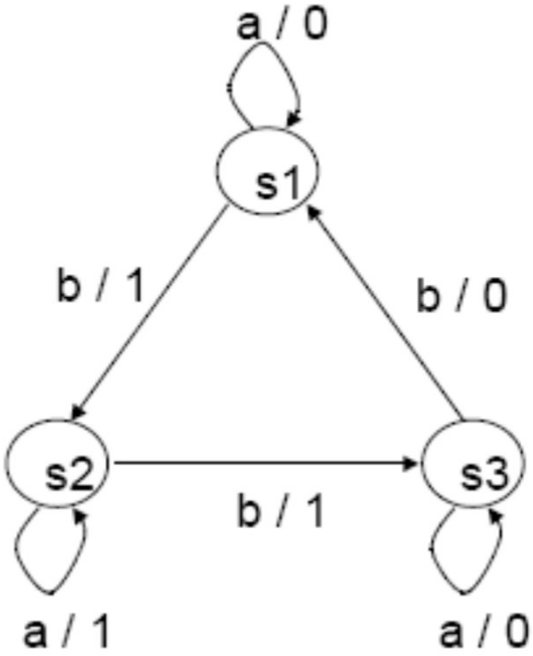
- D'un TT, on tire une séquence de test formée d'une séquence d'entrées et de la séquence de sorties correspondante
  - Suite d'inputs: *ababab*
  - Suite d'outputs attendues: *011100*



Méthode TT (3)

- **Détection erreur de transfert ??**

Spec



IUT ???

À vous de trouver !!

Détection erreurs de sorties + transferts

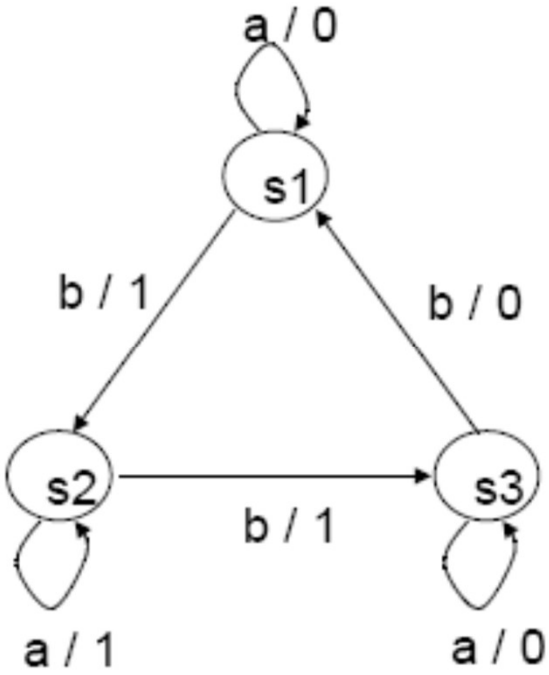
- **DS, W et UIO**
- **Idée:**
  - Générer une séquence de test tq: ∀ (*s1,i,o,s2*)
    - Mettre l'IUT dans l'état s1 (préambule)
    - Appliquer l'input *i* et vérifier si l'output observé est *o* (erreur de sortie)
    - Déterminer si l'état de l'IUT atteint est s2 (erreur de transfert)

DS (1)

- **Une séquence d'inputs est une DS si**
  - Après son exécution, on peut déterminer l'état initial par la suite de sorties observée

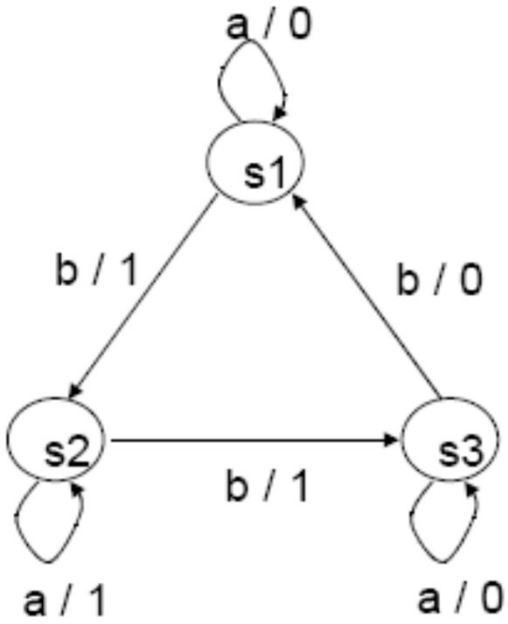
DS (2)

- **Montrer que *a* n'est pas une DS**



DS (3)

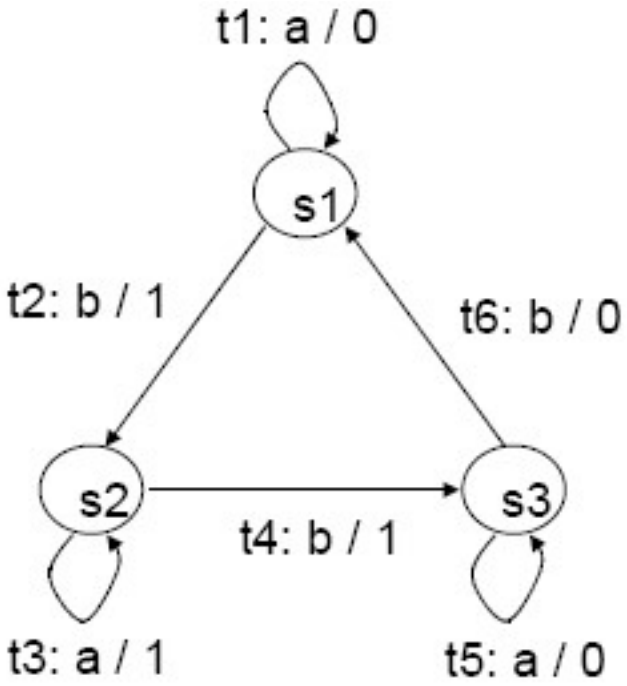
- **Trouver une DS – *à vous de jouer !***



Initial state	Input seq	Output seq	Final state
S1	ab	01	S2
S2	ab	11	S3
S3	ab	00	S1

DS (4)

- **Écrire la séquence de test en couvrant toute les transitions.** *À vous de jouer !*

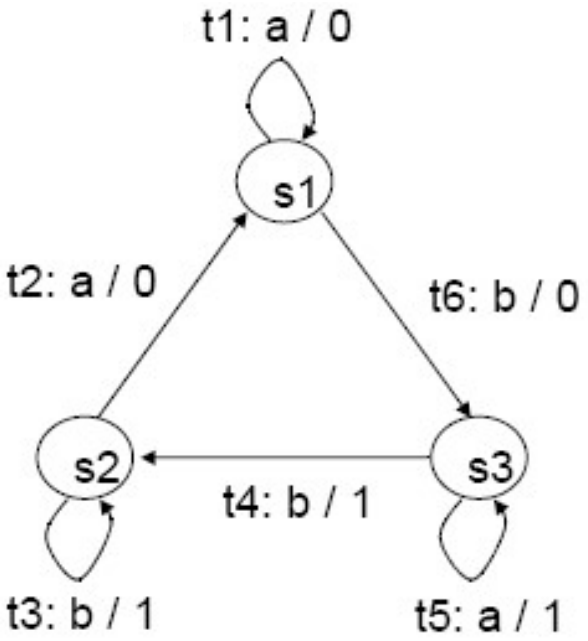


t1: reset/null a/0 a/0 b/1
t2: reset/null b/1 a/1 b/1
t3: reset/null b/1 a/1 a/1 b/1
t4: reset/null b/1 b/1 a/0 b/0
t5: reset/null b/1 b/1 a/0 a/0 b/0
t6: reset/null b/1 b/1 b/0 a/0 b/1

Préambule Sortie Transition

DS (5)

- *À vous de jouer !*



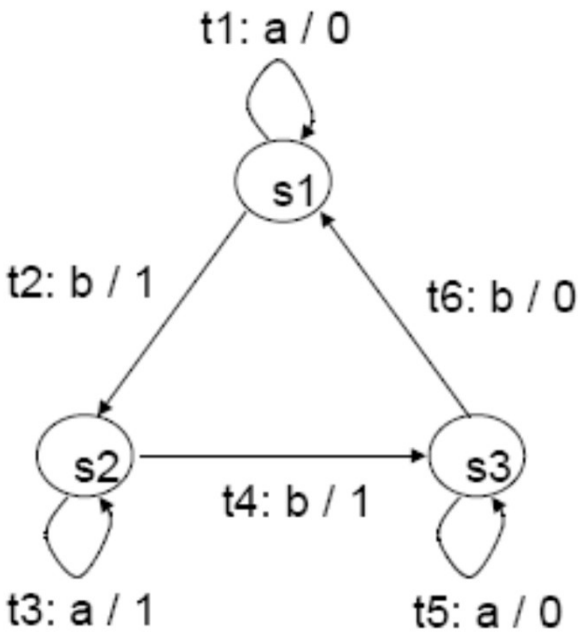
W (1)

- Un ensemble d'inputs est un ***ensemble discriminant***

  - Après application de toutes les suites de l'ensemble, on peut déterminer l'état source par observation de la suite de outputs.

W (2)

- ***{a,b}*** **est un W**

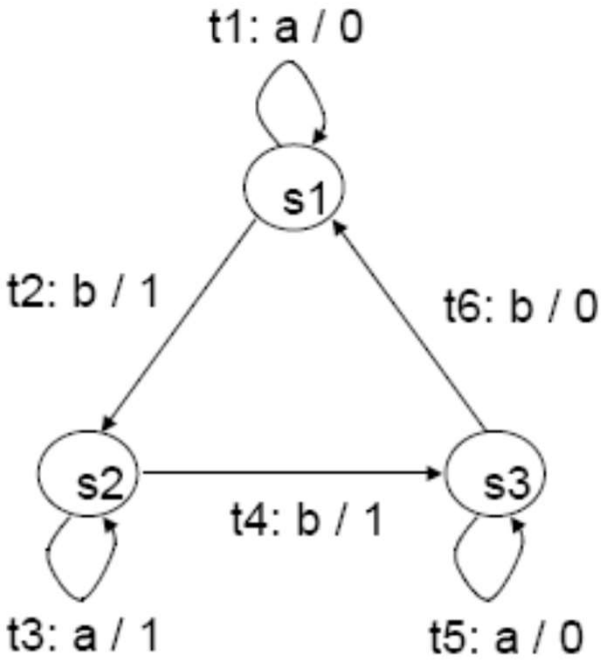


État initial	Suite entrée	Suite sortie	État final
S1	a	0	S1
S2	a	1	S2
S3	a	0	S3

État initial	Suite entrée	Suite sortie	État final
S1	b	1	S3
S2	b	1	S2
S3	b	0	S2

W (3)

- **Voici donc une séquence de test**



t1: reset/null a/0 a/0
t1: reset/null a/0 b/1
t2: reset/null b/1 a/1
t2: reset/null b/1 b/1
t3: reset/null b/1 a/1 a/1
t3: reset/null b/1 a/1 b/1
t4: reset/null b/1 b/1 a/0
t4: reset/null b/1 b/1 b/0
t5: reset/null b/1 b/1 a/0 a/0
t5: reset/null b/1 b/1 a/0 b/0
t6: reset/null b/1 b/1 b/0 a/0
t6: reset/null b/1 b/1 b/0 b/1

W (4)

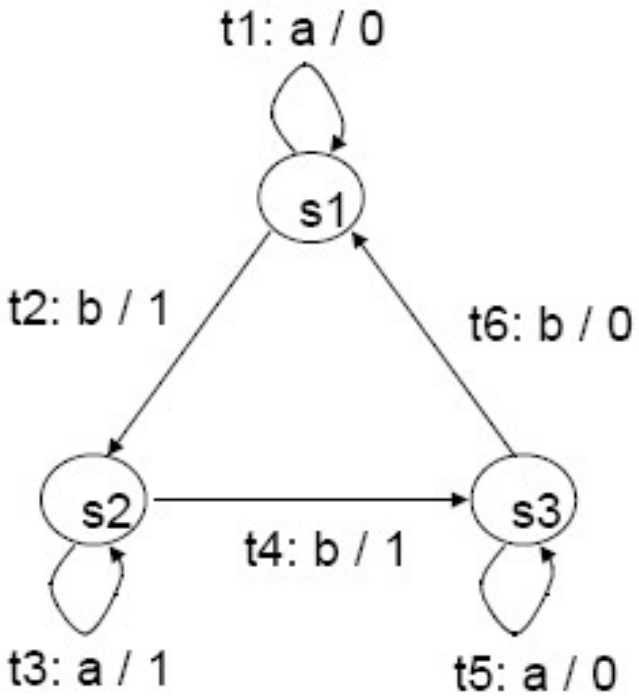
- **Tout système représenté sous forme « d'automates » possède un W**
  - Souvent très volumineux
  - Difficilement générable (temps/spacial)
  - Séquence très longue
  - Contraintes trop importantes !

UIO (1)

- Une suite **UIO** pour un état *s* si:
  - Après application de la suite d'inputs, on peut déterminer l'état initial par la suite d'outputs observée

UIO (2)

- **UIO pour s2 et s3 ??** *À vous de jouer !!*

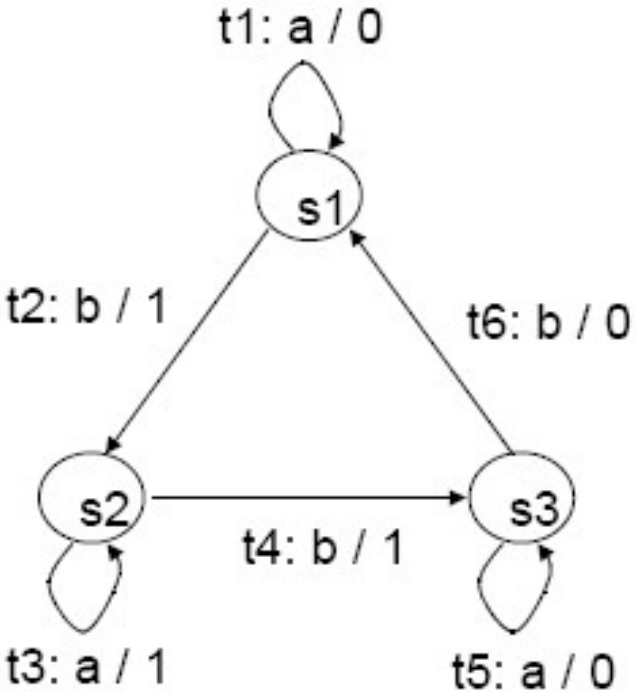


État initial	Suite entrée	Suite sortie	État final
S1	a	0	S1
S2	a	1	S2
S3	a	0	S3

État initial	Suite entrée	Suite sortie	État final
S1	b	1	S3
S2	b	1	S2
S3	b	0	S2

UIO (3)

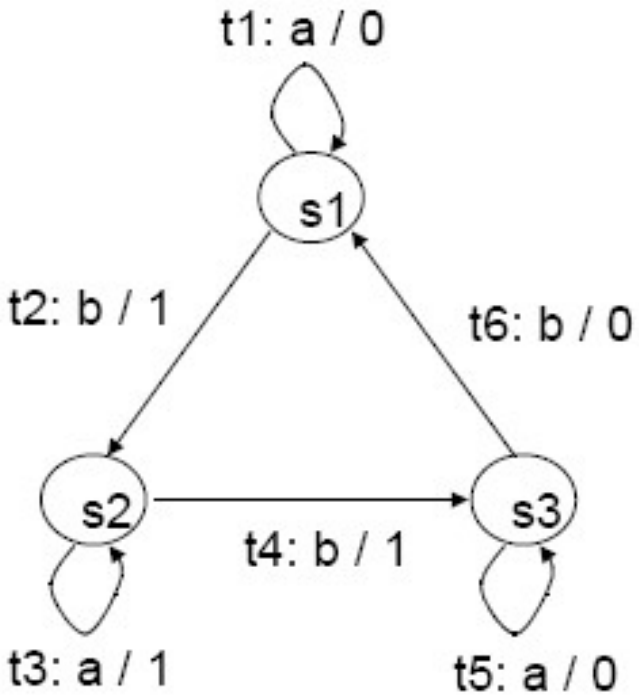
- Pour *s1* ?? *À vous de jouer !!*



État initial	Suite entrée	Suite sortie	État final
S1	ab	01	S2
S2	ab	11	S3
S3	ab	00	S1

UIO (4)

- **Séquence de test ??** *À vous de jouer !!*



t1: reset/null a/0 a/0 b/1
t2: reset/null b/1 a/1
t3: reset/null b/1 a/1 a/1
t4: reset/null b/1 b/1 b/0
t5: reset/null b/1 b/1 a/0 b/0
t6: reset/null b/1 b/1 b/0 a/0 b/1

Séquences de test de conformité

- **Si une réponse est observée durant l'exécution du préambule du cas de test, le verdict doit être déclaré comme *inconclusif*.**
  - D'autres tests devront s'attaquer à la transition mise en défaut
  - Attention à la localisation de l'erreur
- **Questions en vrac …**
  - Que dire si erreur dans le postambule (le reset) ?!
  - Si des préambules, des parties de séquences de test sont les mêmes, peut-on "superposer" les séquences de tests?

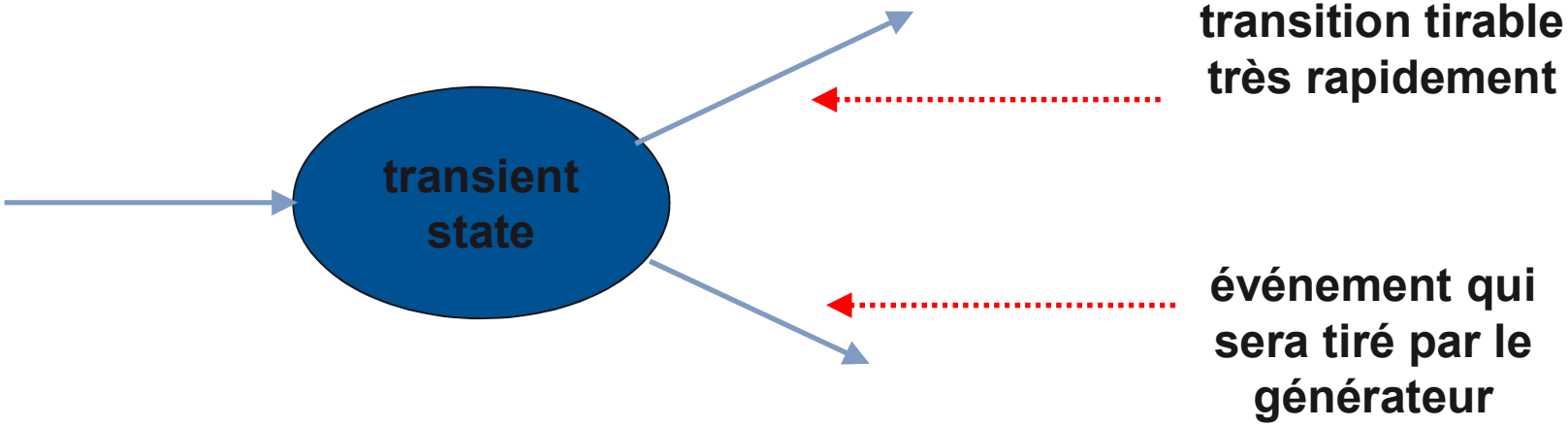
Génération des Séquences
De quoi avons nous besoin?

- **Un parser du modèle formel à partir duquel les séquences sont obtenues**
- **Des algos !**
  - DS et UIO sont généralement des problèmes NP-complets (mais sont facilement résolus dans de très nombreux cas)
  - Les séquences n'existent pas toujours!
  - Non-déterminismes: Spec ? IUT ?
  - Pour trouver les préambules (et éventuellement les postanbules)
    - Approche théorique: shortest path to/from
    - Approche pratique: l'utilisateur peut vouloir identifier des préambules (post-) préférés pour garder un contrôle modérer sur le SUT (ne pas utiliser un bouton "emergency stop" comme préambul … le test est actif !).
  - Et le temps dans tout ça ?!!

Génération des Séquences
De quoi avons nous besoin?

- **Etats *transient* :**
  - **Ces états existent de manière temporaire, elle s'exécute sans intervention extérieure.**
  - **Ces états peuvent avoir des transitions sans inputs !**
  - **Les outils de génération ne prennent pas souvent ces états en considération ! Le générateur passera par une autre transition.**

- **Tests difficiles à écrire et à générer !**



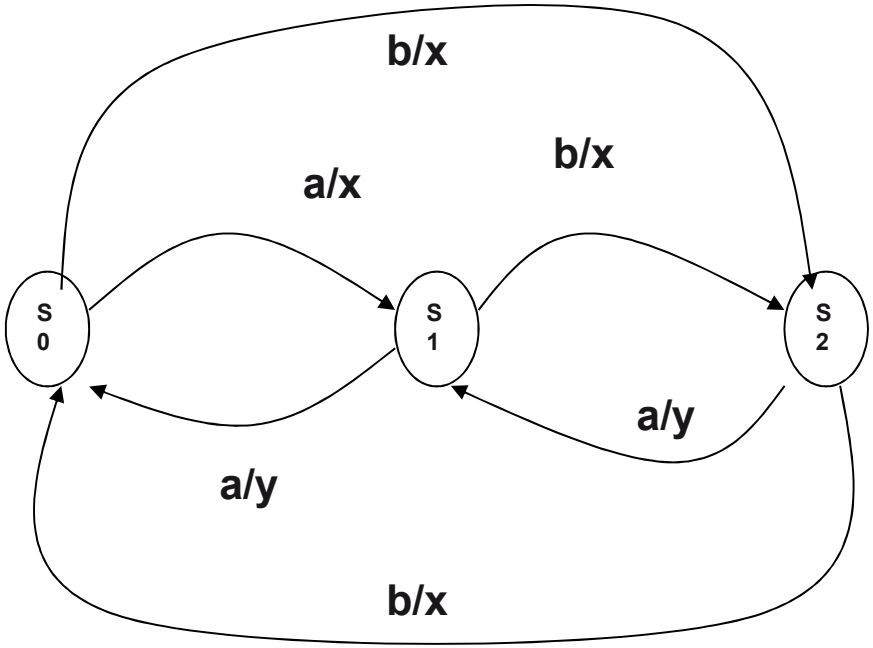
Problèmes Ouverts et Test !

- **FSM / conformance testing conference series:**
  - FORTE (formal description techniques)
  - ICTSS / Testcom (testing of communicating systems)
  - ISSTA / ISSRE (software testing and analysis / software reliability engineering)
  - ICST (International Conference on Software Testing)

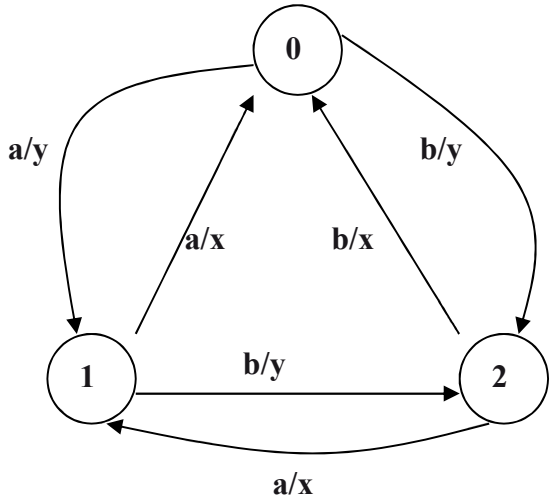
  - ASE, IFM, ……..

UIO (5)

- *À vous de jouer !!*



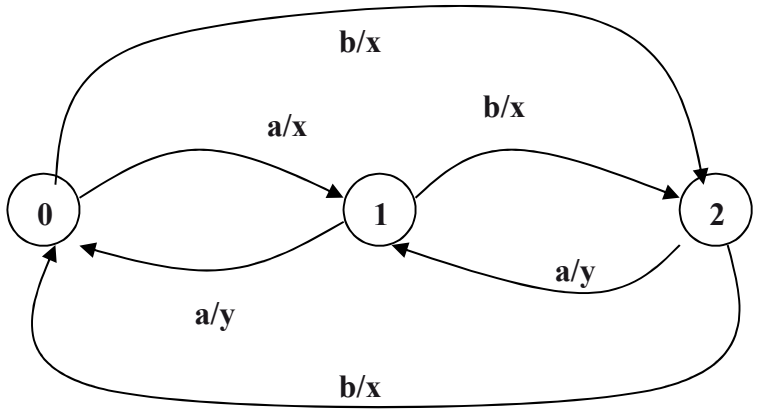
Exercices



1-Donner l'ensemble W pour ce modèle et une séquence UIO pour chaque état.

2- Donner ensuite, à partir des UIO, une séquence pour tester la transition 1 vers 0.

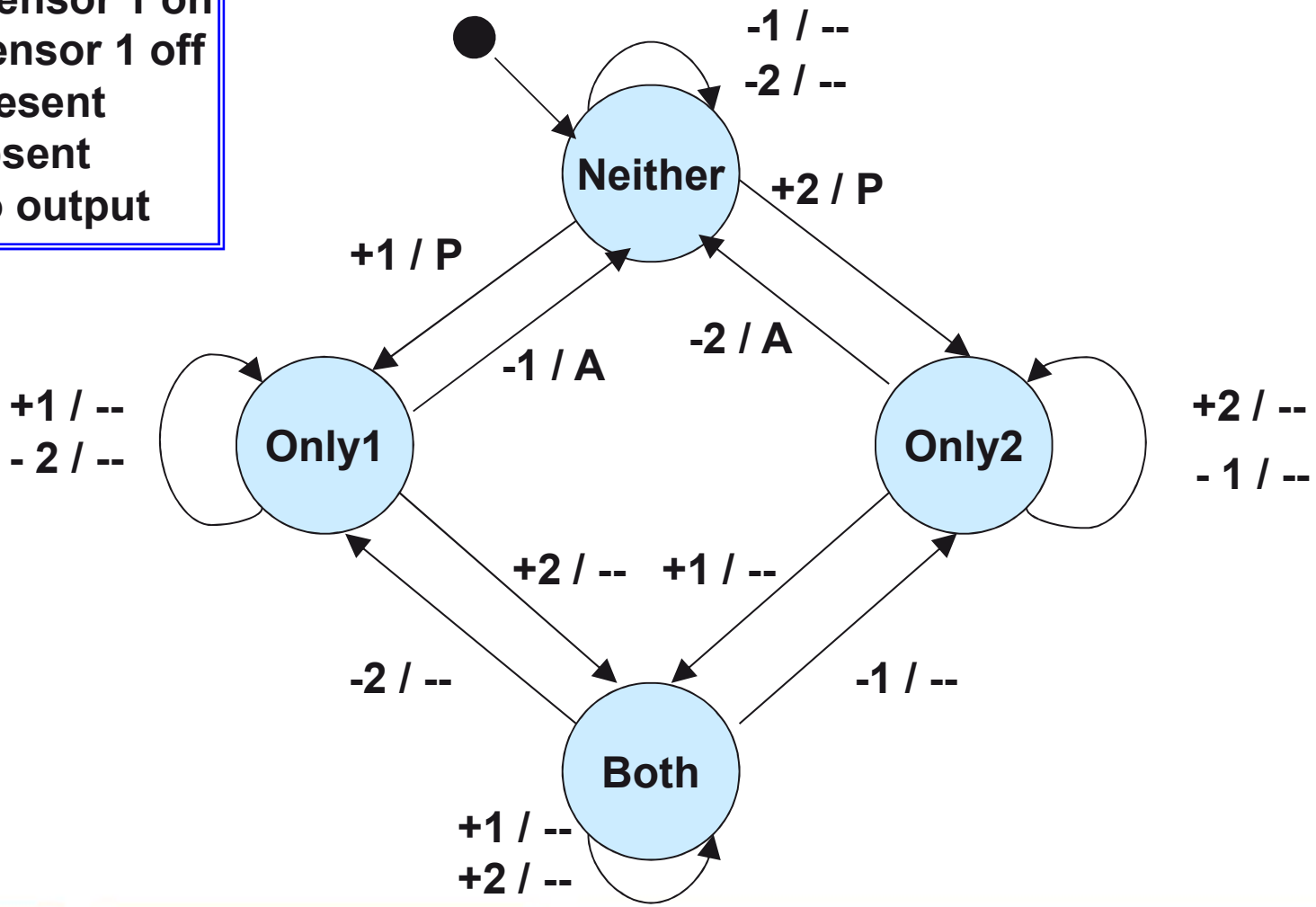
Initial state : 0
Input= {a,b}
Output= {x,y}



1- Déterminer une DS pour ce modèle et écrire une séquence de test pour la transition 1->0
2- utiliser la méthode UIO pour écrire une séquence de test pour la même transition.

Exemple d'un réseau de capteurs

+1 = sensor 1 on
-1 = sensor 1 off
P = present
A = absent
-- = no output



UIO - PUIO

Etat	Préambule	UIO		Postambule
		Sequence	End state	Reset à partir de Etat
Neither	[none]	+1 / P	Only1	[none]
Only1	+1 / P	-1 / A	Neither	-1 / A
Only2	+2 / P	-2 / A	Neither	-2 / A

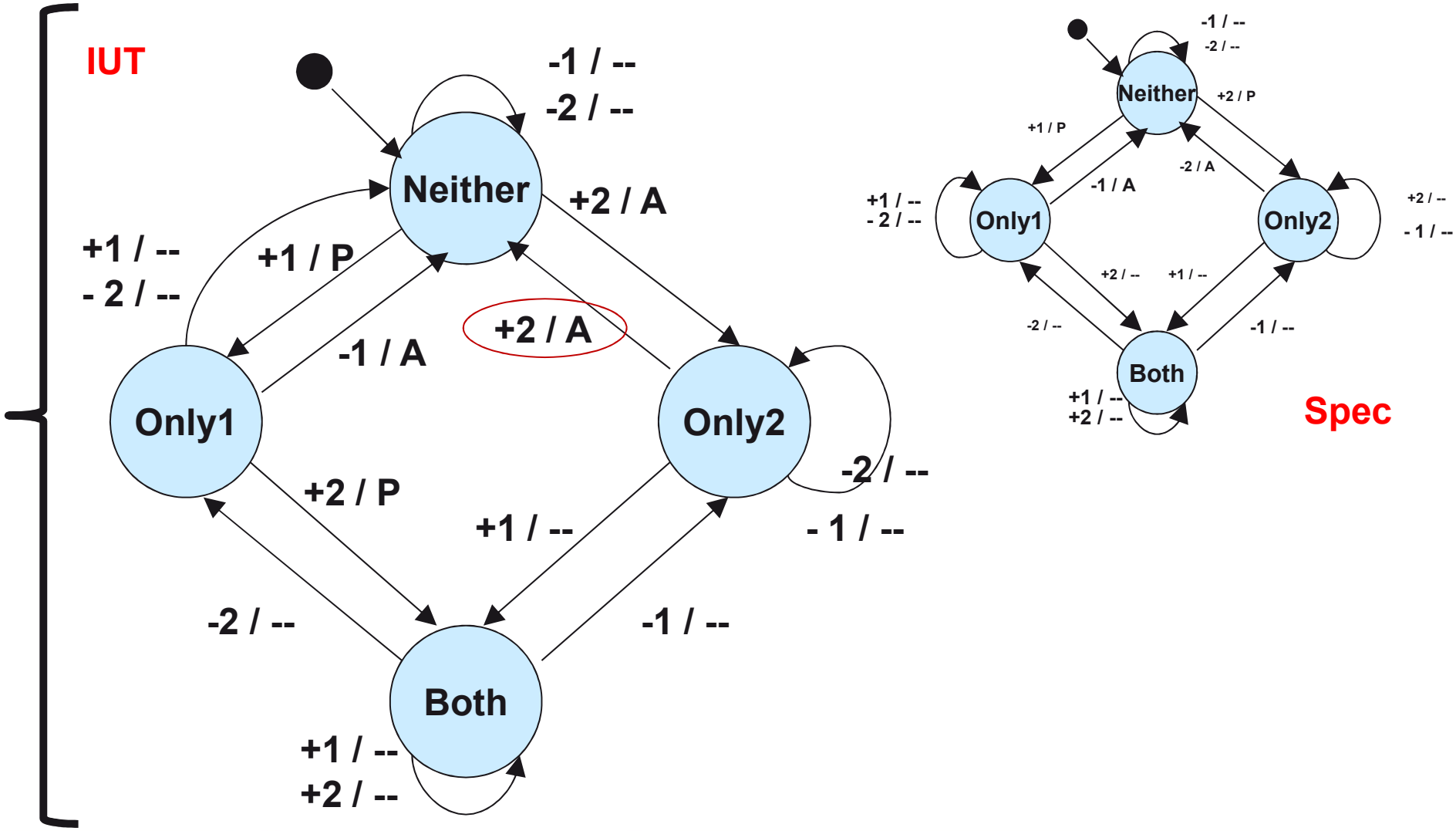
Both	+1 / P,+2 / --	-2 / --, -1 / A -1 / --, -2 / A	Neither	-2 / --, -1 / A

Partial UIO

PUIO

- **Quelles séquences de test pour tester**
  - Only2, -1/-- , Only2
  - Only2, +1/-- , Both

Quelles erreurs dans l'IUT ?



Quelles séquences de test ?

- **Testons “O2, -2/A, N” ???**

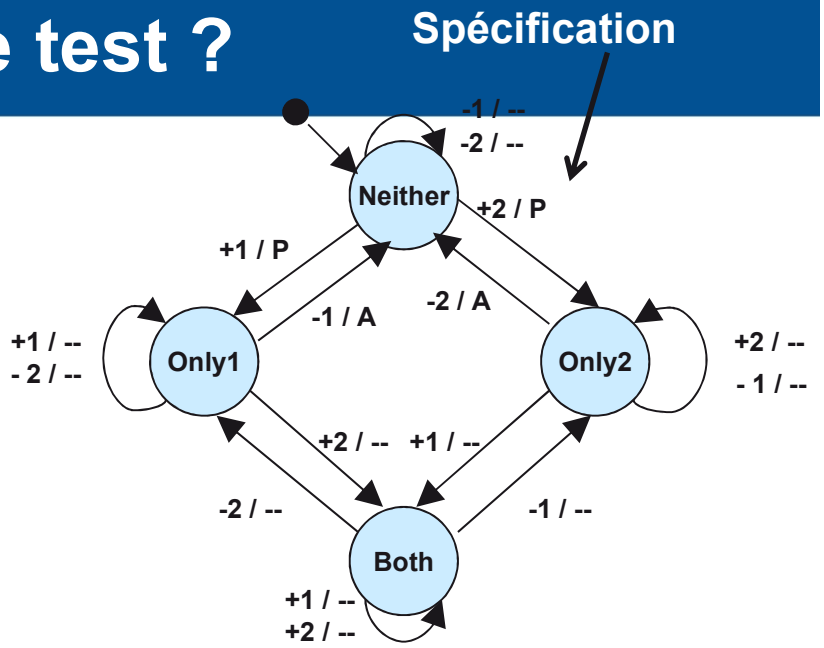
  +1/P , +2/-- , -1/-- , -2/A , +2/P

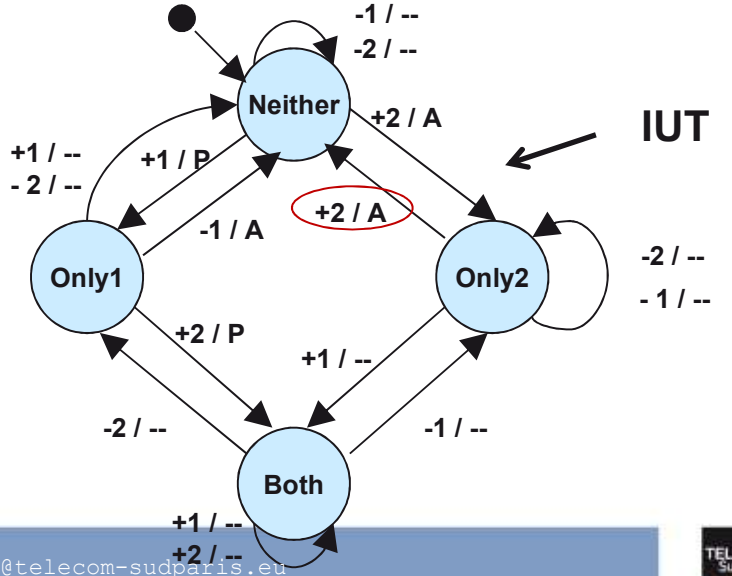
  On obtient:

  P, P, -- , -- , A

  Que conclure? Verdicts?

  Inconclusif !?





Trouvez toutes les erreurs.

Quelles erreurs dans l'IUT ?