Quick Introduction to Software Engineering

Gerson Sunyé
gerson.sunye@univ-nantes.fr
http://www.univ-nantes.fr/sunye-g

Agenda

- Software Engineering Definition
- Modern Software Complexity 复杂性
- The Software Industry Today
- Software Development Process
- Conclusion

Remerciements

- Jean-Marc Jézéquel, Benoît Combemale, Olivier Barais.
  - Université de Rennes 1
- Yves Le Traon
  - Université du Luxembourg
- Jean-Marie Mottu, Gilles Ardourel
  - Université de Nantes

Objectifs

- Appréhender la complexité des systèmes modernes
- Comprendre les enjeux du Génie Logiciel
- Avoir un aperçu des éléments de solution :
  - la séparation des préoccupations
  - la continuité (technologique) de la modélisation à la programmation
  - la continuité (des exigences, ... à la livraison, ... à l'évolution) des activités d'un processus de développement

Definition

A Little History

- In 1843, Ada Lovelace translates Frederico Luigi de Menabrea's paper "Sketch of the Analytical Engine Invented by Charles Babbage", adding several notes.
- Note G describes an detailed algorithm for computing Bernoulli's numbers with the analytical engine.

Number of Operation	Nature of Operation	Variables acted upon	Variables receiving results	Indication of change in the value on any Variable	Statement of Results	Data			Working Variables										Result Variables			
						$^1V_1$	$^1V_2$	$^1V_3$	$^0V_4$	$^0V_5$	$^0V_6$	$^0V_7$	$^0V_8$	$^0V_9$	$^0V_{10}$	$^0V_{11}$	$^0V_{12}$	$^0V_{13}\cdots$	$^1V_{21}$	$^1V_{22}$	$^1V_{23}$	$^0V_{24}$
						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
						0	0	0	0	0	0	0	0	0	0	0	0	0	B in a dec. fract.	B in a dec. fract.	B in a dec. fract.	0
						0	0	0	0	0	0	0	0	0	0	0	0	0				0
						1	2	4	0	0	0	0	0	0	0	0	0	0				0
						$\boxed{1}$	$\boxed{2}$	$\boxed{n}$	$\boxed{}$	$\boxed{}$	$\boxed{}$	$\boxed{}$	$\boxed{}$	$\boxed{}$	$\boxed{}$	$\boxed{}$	$\boxed{}$	$\boxed{}$	$\boxed{B_1}$	$\boxed{B_3}$	$\boxed{B_5}$	$\boxed{B_7}$
	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$\begin{Bmatrix} {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \end{Bmatrix}$	$= 2n$	......	2	n	$2n$	$2n$	$2n$											
	−	$^1V_4 - {}^1V_1$	$^2V_4$	$\begin{Bmatrix} {}^1V_4 = {}^2V_4 \\ {}^1V_1 = {}^1V_1 \end{Bmatrix}$	$= 2n - 1$	1	......	......	$2n - 1$													
	+	$^1V_5 + {}^1V_1$	$^2V_5$	$\begin{Bmatrix} {}^1V_5 = {}^2V_5 \\ {}^1V_1 = {}^1V_1 \end{Bmatrix}$	$= 2n + 1$	1	......	......	......	$2n + 1$												
	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$\begin{Bmatrix} {}^2V_5 = {}^0V_5 \\ {}^2V_4 = {}^0V_4 \end{Bmatrix}$	$= \frac{2n-1}{2n+1}$	......	......	......	0	0	......	......	......	......	......	$\frac{2n-1}{2n+1}$						
	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$\begin{Bmatrix} {}^1V_{11} = {}^2V_{11} \\ {}^1V_2 = {}^1V_2 \end{Bmatrix}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	......	2	......	......	......	......	......	......	......	......	$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
	−	$^0V_{13} - {}^2V_{11}$	$^1V_{13}$	$\begin{Bmatrix} {}^2V_{11} = {}^0V_{11} \\ {}^0V_{13} = {}^1V_{13} \end{Bmatrix}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	......	......	......	......	......	......	......	......	......	......	0	......	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
	−	$^1V_3 - {}^1V_1$	$^1V_{10}$	$\begin{Bmatrix} {}^1V_3 = {}^1V_3 \\ {}^1V_1 = {}^1V_1 \end{Bmatrix}$	$= n - 1 (= 3)$	1	......	n	......	......	......	......	......	......	$n - 1$							
	+	$^1V_2 + {}^0V_7$	$^1V_7$	$\begin{Bmatrix} {}^1V_2 = {}^1V_2 \\ {}^0V_7 = {}^1V_7 \end{Bmatrix}$	$= 2 + 0 = 2$	......	2	......	......	......	......	2										
	÷	$^1V_6 \div {}^1V_7$	$^3V_{11}$	$\begin{Bmatrix} {}^1V_6 = {}^1V_6 \\ {}^0V_{11} = {}^3V_{11} \end{Bmatrix}$	$= \frac{2n}{2} = A_1$	......	......	......	......	......	$2n$	2	......	......	......	$\frac{2n}{2} = A_1$						
	×	$^1V_{21} \times {}^3V_{11}$	$^1V_{12}$	$\begin{Bmatrix} {}^1V_{21} = {}^1V_{21} \\ {}^3V_{11} = {}^3V_{11} \end{Bmatrix}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$	......	......	......	......	......	......	......	......	......	......	$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$	......	B_1			
	+	$^1V_{12} + {}^1V_{13}$	$^2V_{13}$	$\begin{Bmatrix} {}^1V_{12} = {}^0V_{12} \\ {}^1V_{13} = {}^2V_{13} \end{Bmatrix}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$	......	......	......	......	......	......	......	......	......	......	......	0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$				
	−	$^1V_{10} - {}^1V_1$	$^2V_{10}$	$\begin{Bmatrix} {}^1V_{10} = {}^2V_{10} \\ {}^1V_1 = {}^1V_1 \end{Bmatrix}$	$= n - 2 (= 2)$	1	......	......	......	......	......	......	......	......	$n - 2$							
	−	$^1V_6 - {}^1V_1$	$^2V_6$	$\begin{Bmatrix} {}^1V_6 = {}^2V_6 \\ {}^1V_1 = {}^1V_1 \end{Bmatrix}$	$= 2n - 1$	1	......	......	......	......	$2n - 1$											
	+	$^1V_1 + {}^1V_7$	$^2V_7$	$\begin{Bmatrix} {}^1V_1 = {}^1V_1 \\ {}^1V_7 = {}^2V_7 \end{Bmatrix}$	$= 2 + 1 = 3$	1	......	......	......	......	......	3										
	÷	$^2V_6 \div {}^2V_7$	$^1V_8$	$\begin{Bmatrix} {}^2V_6 = {}^2V_6 \\ {}^2V_7 = {}^2V_7 \end{Bmatrix}$	$= \frac{2n-1}{3}$	......	......	......	......	......	$2n - 1$	3	$\frac{2n-1}{3}$									
	×	$^1V_8 \times {}^3V_{11}$	$^4V_{11}$	$\begin{Bmatrix} {}^1V_8 = {}^0V_8 \\ {}^3V_{11} = {}^4V_{11} \end{Bmatrix}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$	......	......	......	......	......	......	......	0	......	......	$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
	−	$^2V_6 - {}^1V_1$	$^3V_6$	$\begin{Bmatrix} {}^2V_6 = {}^3V_6 \\ {}^1V_1 = {}^1V_1 \end{Bmatrix}$	$= 2n - 2$	1	......	......	......	......	$2n - 2$											
	+	$^1V_1 + {}^2V_7$	$^3V_7$	$\begin{Bmatrix} {}^2V_7 = {}^3V_7 \\ {}^1V_1 = {}^1V_1 \end{Bmatrix}$	$= 3 + 1 = 4$	1	......	......	......	......	......	4										
	÷	$^3V_6 \div {}^3V_7$	$^1V_9$	$\begin{Bmatrix} {}^3V_6 = {}^3V_6 \\ {}^3V_7 = {}^3V_7 \end{Bmatrix}$	$= \frac{2n-2}{4}$	......	......	......	......	......	$2n - 2$	4	......	$\frac{2n-2}{4}$	......							
	×	$^1V_9 \times {}^4V_{11}$	$^5V_{11}$	$\begin{Bmatrix} {}^1V_9 = {}^0V_9 \\ {}^4V_{11} = {}^5V_{11} \end{Bmatrix}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$	......	......	......	......	......	......	......	......	0		$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4}\right\} = A_3$						
	×	$^1V_{22} \times {}^5V_{11}$	$^0V_{12}$	$\begin{Bmatrix} {}^1V_{22} = {}^1V_{22} \\ {}^0V_{12} = {}^2V_{12} \end{Bmatrix}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = B_3 A_3$	......	......	......	......	......	......	......	......	......	......	0	$B_3 A_3$	......	......	B_3		
	+	$^2V_{12} + {}^2V_{13}$	$^3V_{13}$	$\begin{Bmatrix} {}^2V_{12} = {}^0V_{12} \\ {}^2V_{13} = {}^3V_{13} \end{Bmatrix}$	$= A_0 + B_1 A_1 + B_3 A_3$	......	......	......	......	......	......	......	......	......	......	......	0	$\{A_0 + B_1 A_1 + B_3 A_3\}$				
	−	$^2V_{10} - {}^1V_1$	$^3V_{10}$	$\begin{Bmatrix} {}^2V_{10} = {}^3V_{10} \\ {}^1V_1 = {}^1V_1 \end{Bmatrix}$	$= n - 3 (= 1)$	1	......	......	......	......	......	......	......	......	$n - 3$							
Here follows a repetition of Operations thirteen to twenty-three																						
	+	$^4V_{13} + {}^0V_{24}$	$^1V_{24}$	$\begin{Bmatrix} {}^4V_{13} = {}^0V_{13} \\ {}^0V_{24} = {}^1V_{24} \end{Bmatrix}$	$= B_7$	......	......	......	......	......	......	......	......	......	......	......	......	......	......	......	......	B_7
	+	$^1V_1 + {}^1V_3$	$^1V_3$	$\begin{Bmatrix} {}^1V_1 = {}^1V_1 \\ {}^1V_3 = {}^1V_3 \end{Bmatrix}$ $^5V_6 = {}^0V_6$ $^5V_7 = {}^0V_7$	$= n + 1 = 4 + 1 = 5$ by a Variable-card. by a Variable-card.	1	......	$n + 1$	......	......	0	0										

http://www.fourmilab.ch/babbage/figures/menat6_2k.png

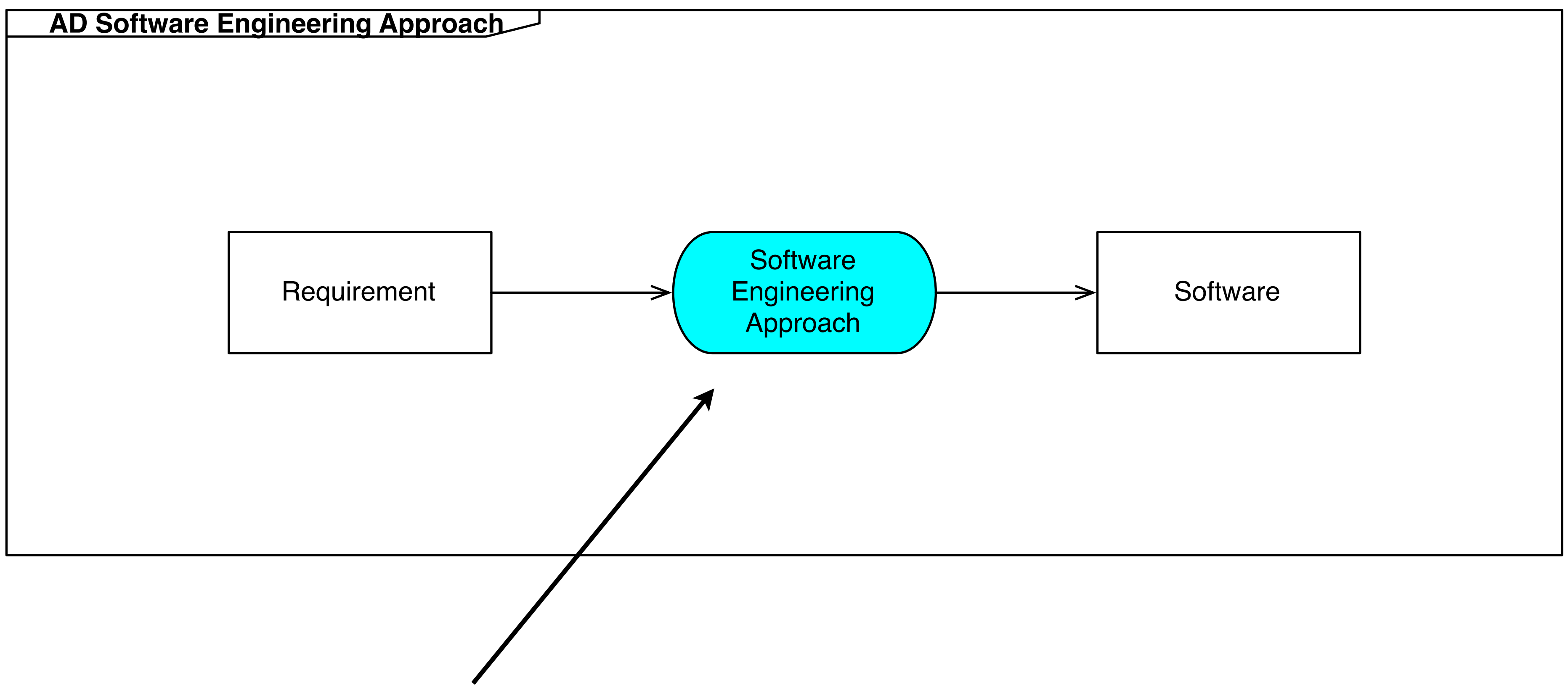
First Computer Programmer

- Ada translated well known formulae into an implementable algorithm.
- This activity is the core of the software engineering process.

Software Engineering Definition

«Software engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software» [SWEBOK]

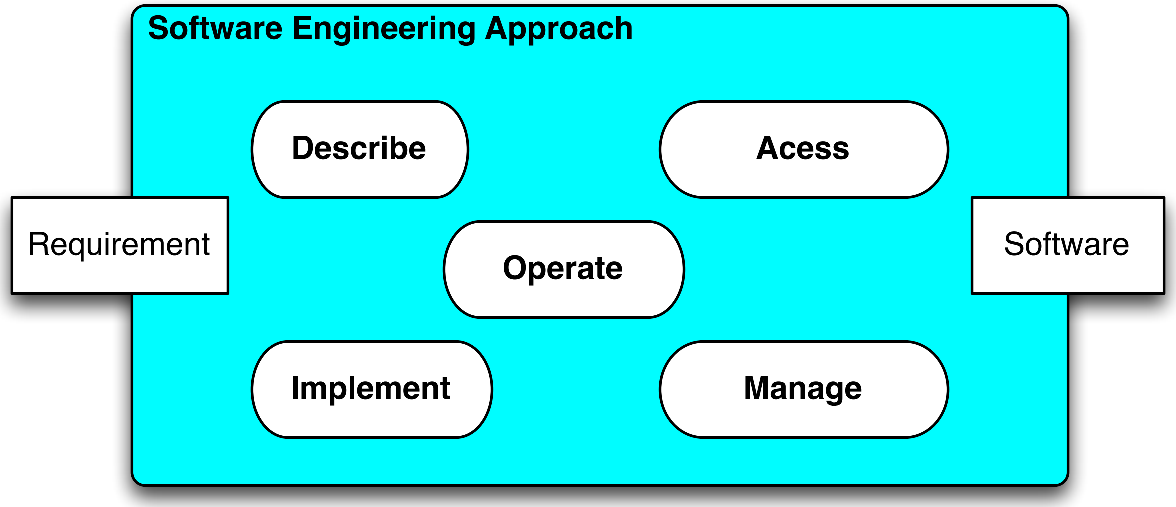
Software Engineering Approach 方法，途径



UML Activity: systematic, disciplined, and quantifiable approach

Software Engineering Activities [Meyer]

- **Describe**: requirements, design, specification, documentation...
- **Implement**: modeling, programming
- **Assess**: testing and other V&V techniques
- **Manage**: plans, schedules, communication, reviews
- **Operate**: deployment, installation...



A Systematic Approach

An approach that follows a method,
with rigor and precision

A Disciplined Approach

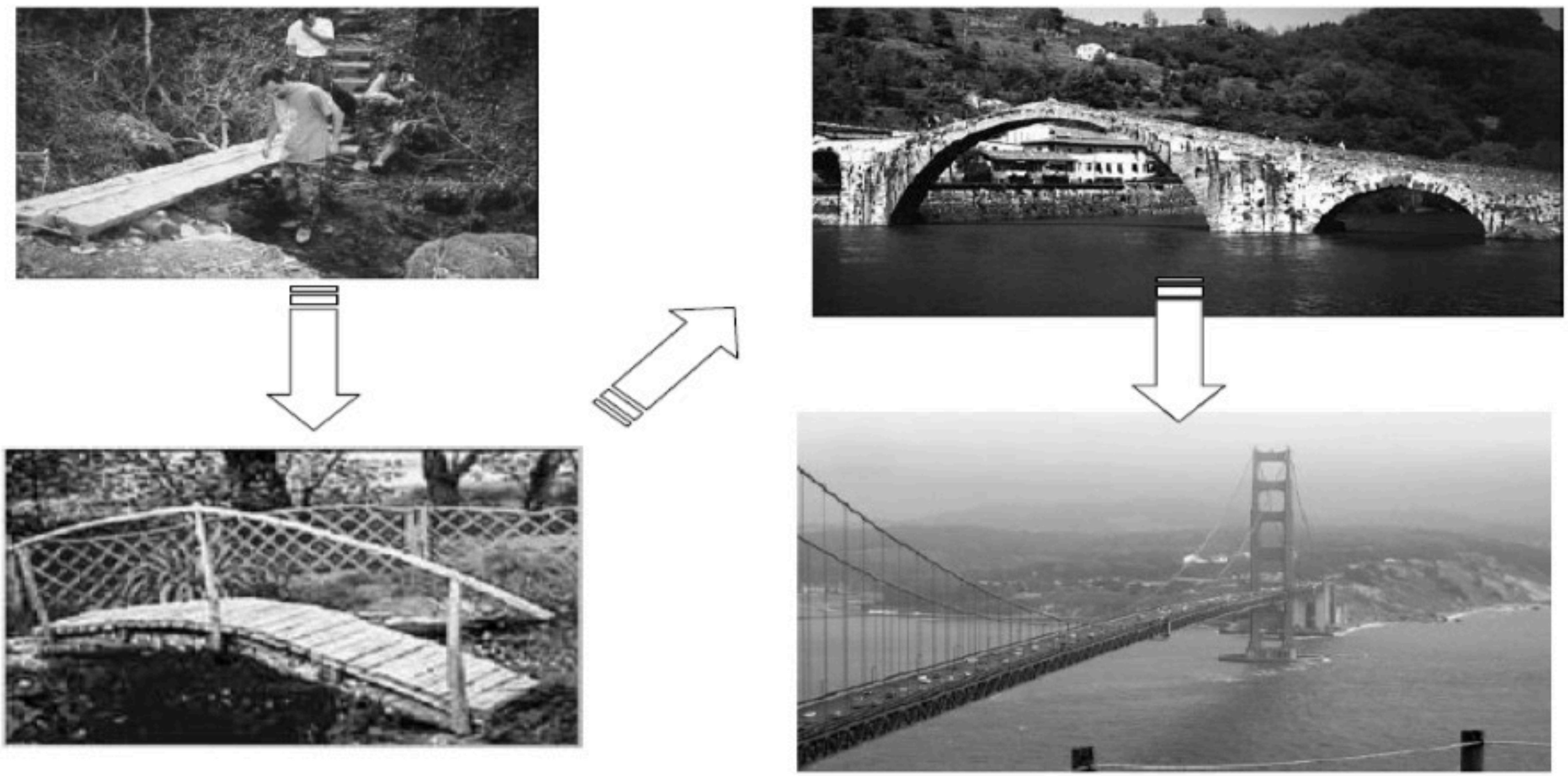
An approach that shows
a controlled form of behavior

A Quantifiable Approach

An approach that can be measured:
from the input until the output

Modern Software Complexity

Modern Software Complexity

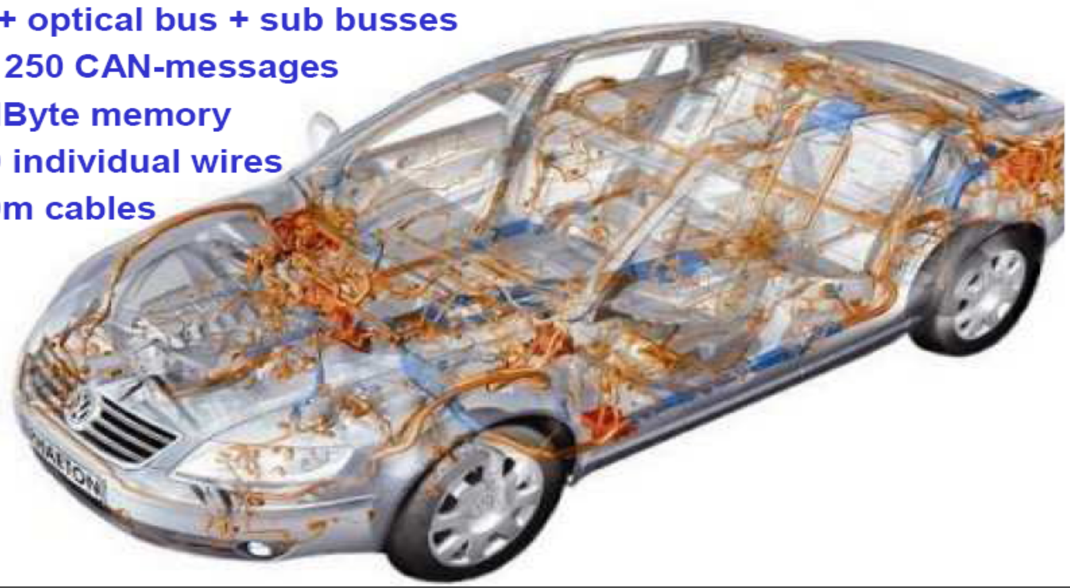


Modern Software Complexity

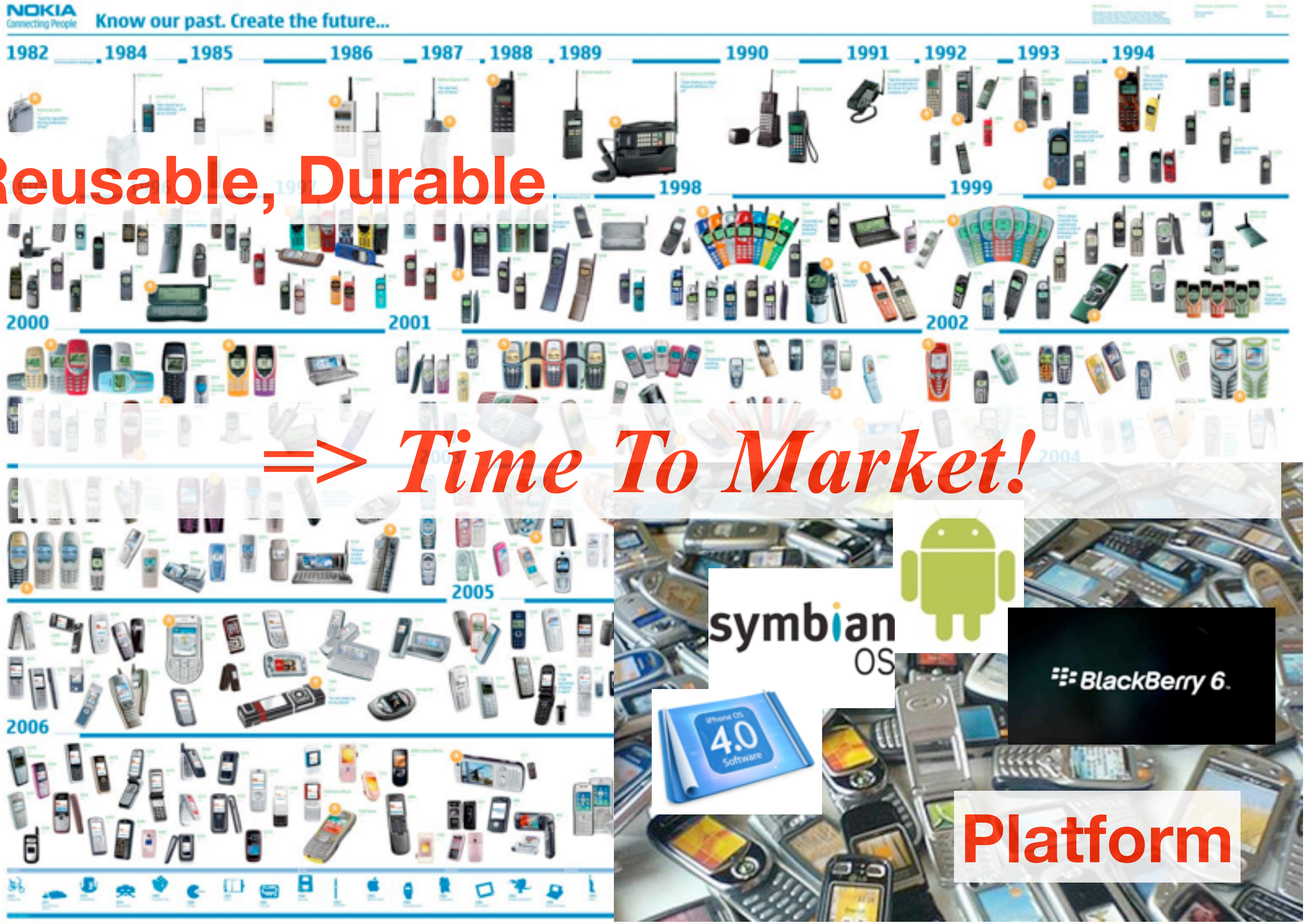
Critical,
Real-Time,
Embedded

Phaeton

- 61 networked ECUs
- 3 bus systems + optical bus + sub busses
- 2500 signals in 250 CAN-messages
- more than 50 MByte memory
- more than 2000 individual wires
- more than 3800m cables



Modern Software Complexity

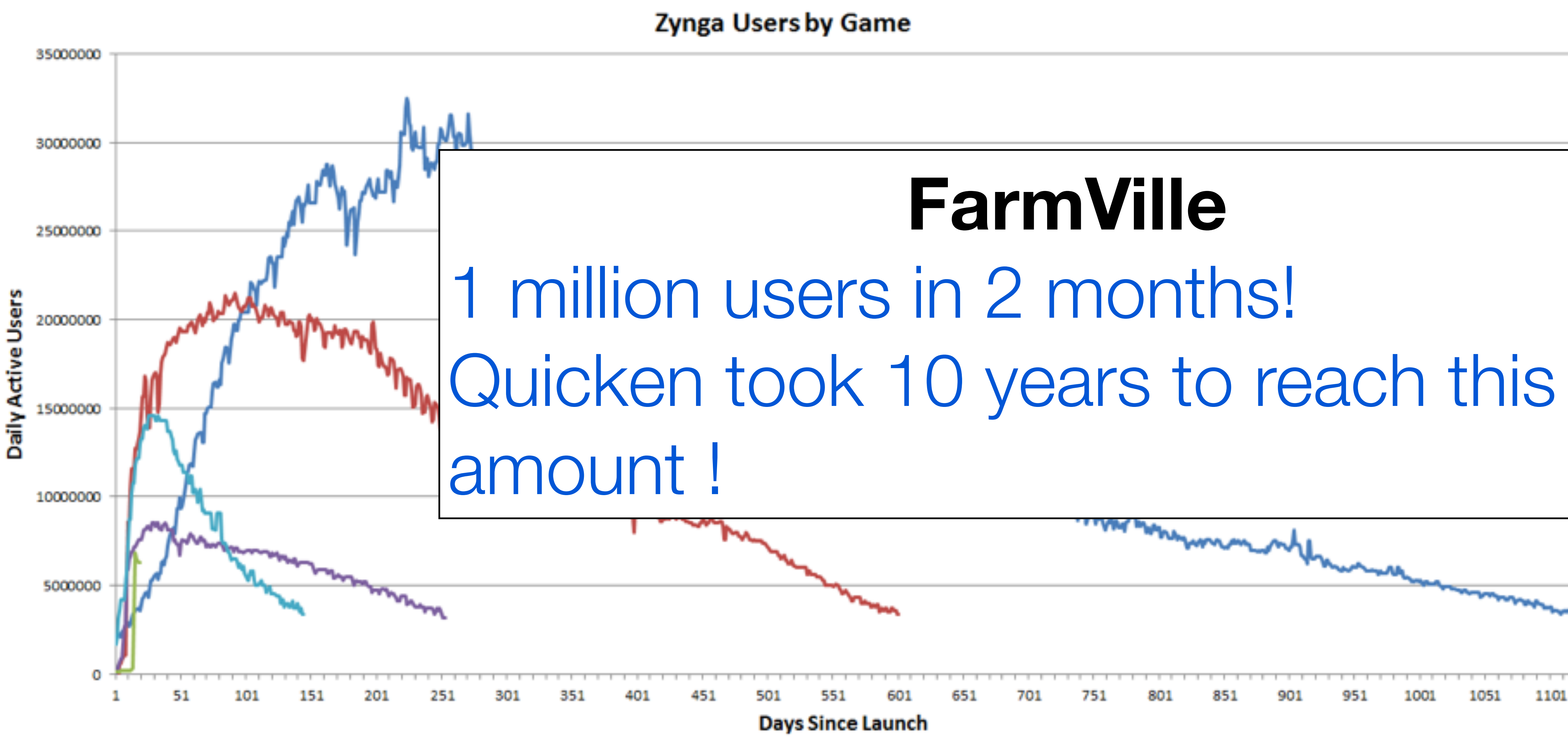


Reusable, Durable

=> *Time To Market!*

Platform

Modern Software Complexity

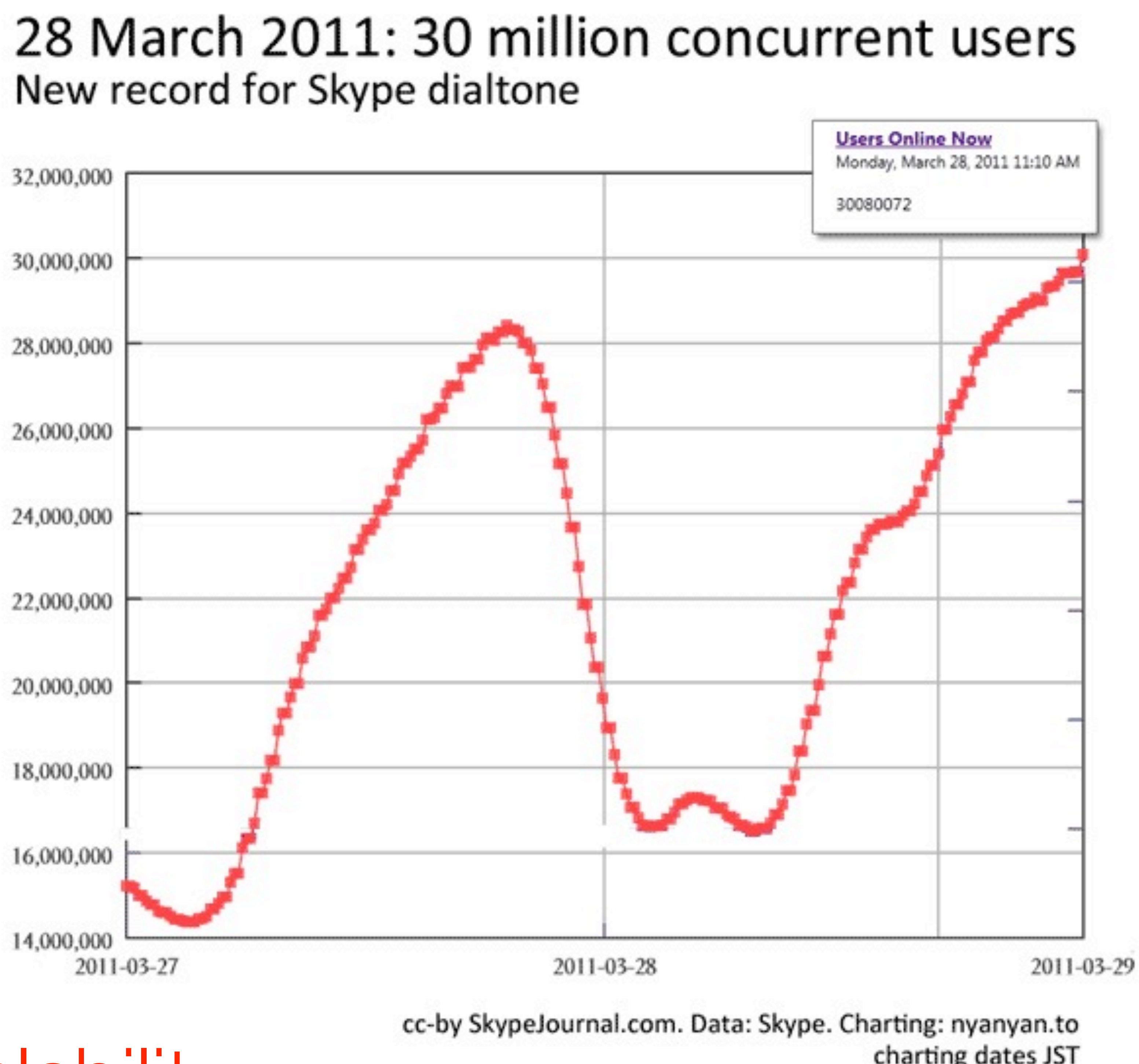


FarmVille

1 million users in 2 months!
Quicken took 10 years to reach this amount !

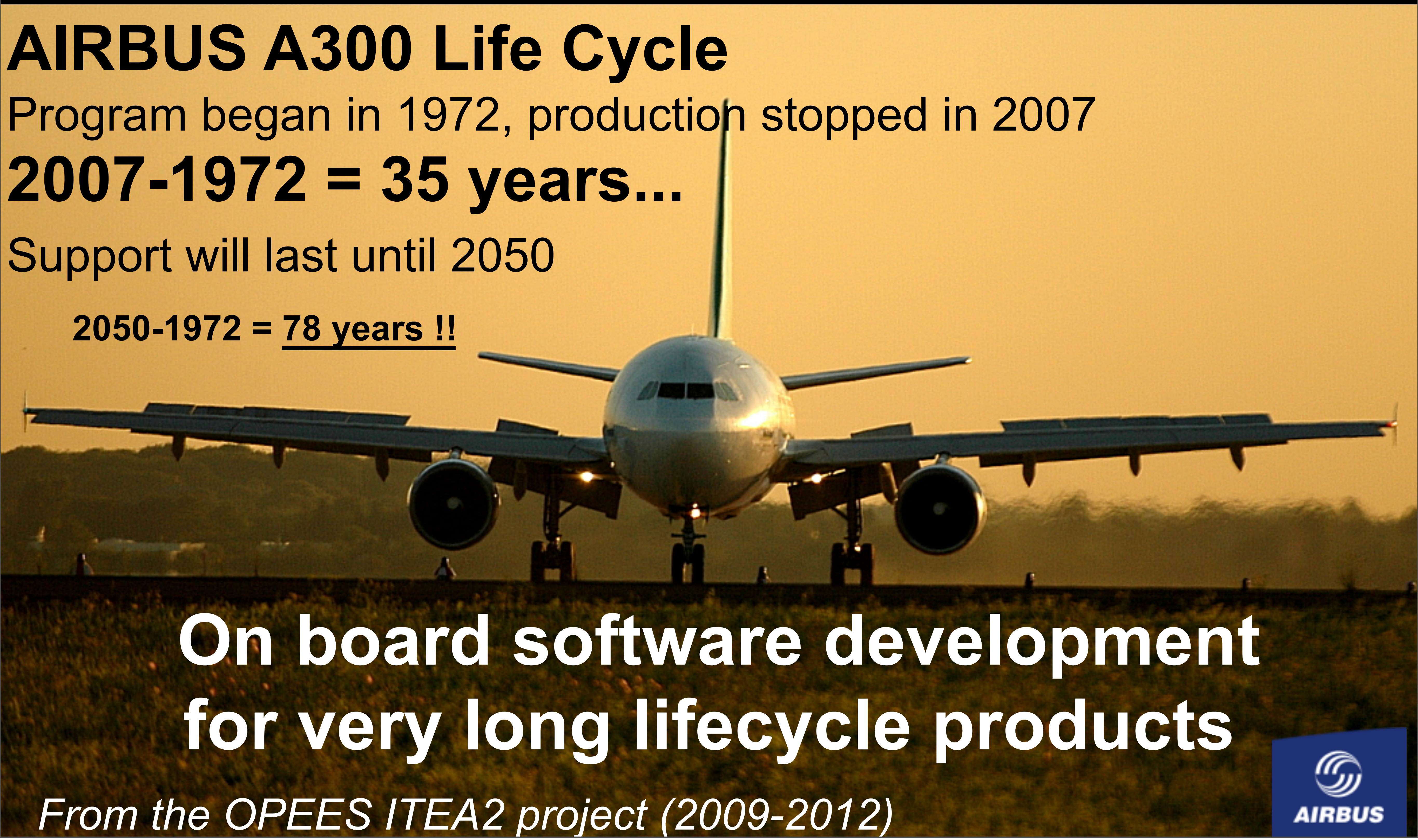
Elasticity: Workload Adaptation

Modern Software Complexity



Scalability

Long term availability…



AIRBUS A300 Life Cycle

Program began in 1972, production stopped in 2007

2007-1972 = 35 years...

Support will last until 2050

2050-1972 = 78 years !!

On board software development for very long lifecycle products

From the OPEES ITEA2 project (2009-2012)

Modern Software Complexity

- Google :
  - 300 000 serveurs
    - répartis dans une vingtaine de datacenters.
    - répondre à plus d'1 milliard de requêtes par jour,
      - *chacune interrogeant 8 milliards de pages Web*
      - *en moins d'un cinquième de seconde*
  - Building for Scale:
    - 6,000 developer / 1,500+ projects
    - Each product has custom release cycles
      - *few days to few weeks*
    - 1(!!) code repository
    - No binary releases
      - *everything builds from HEAD*
    - 20+ code changes per minute
      - *50% of the code changes every month*

- Distribué

- Large-echelle

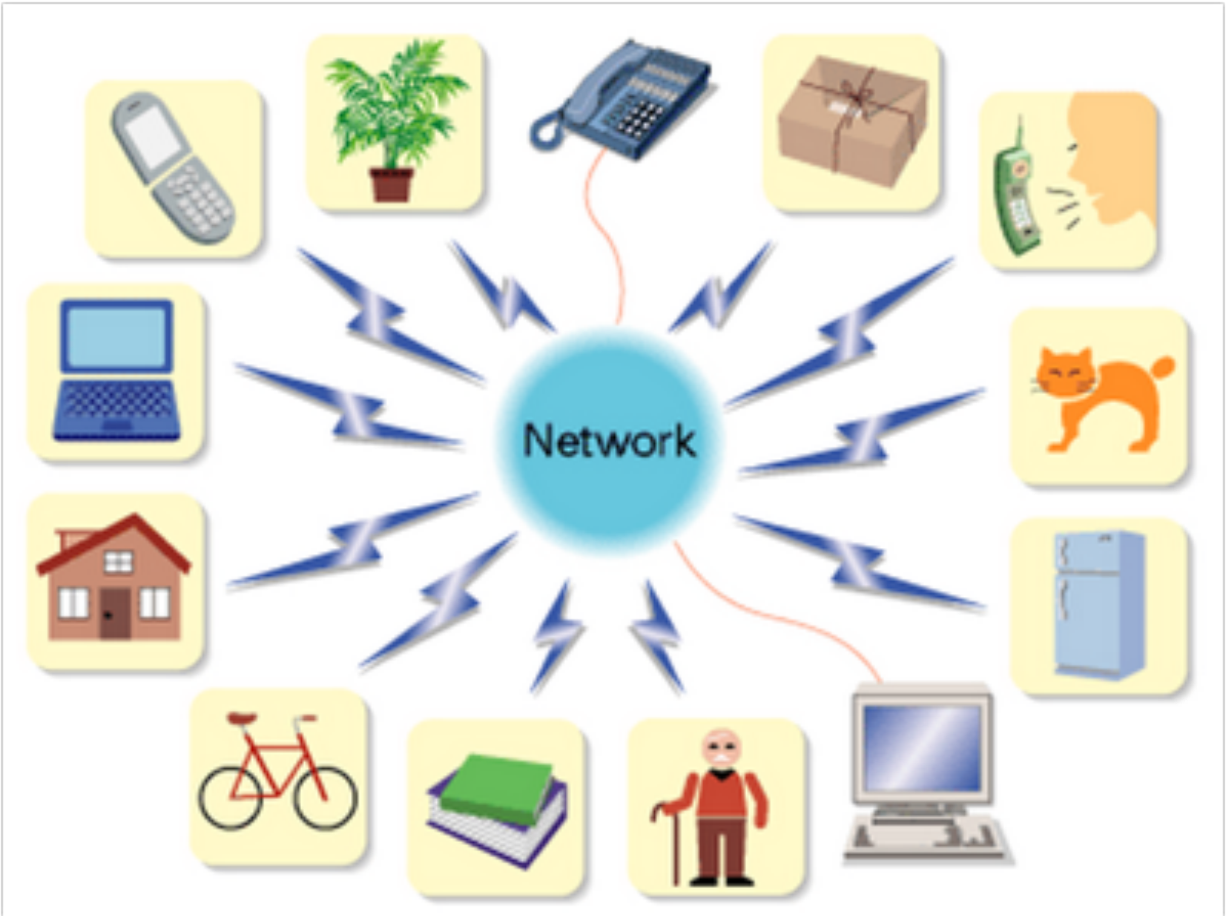
Innovation Factory:
Testing, Culture, &
Infrastructure

Patrick Copeland, Google
ICST 2010

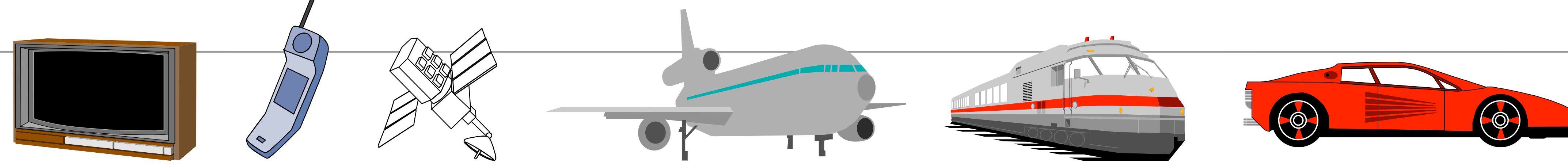
Source: http://googletesting.blogspot.com/search/label/Copeland

Modern Software Complexity

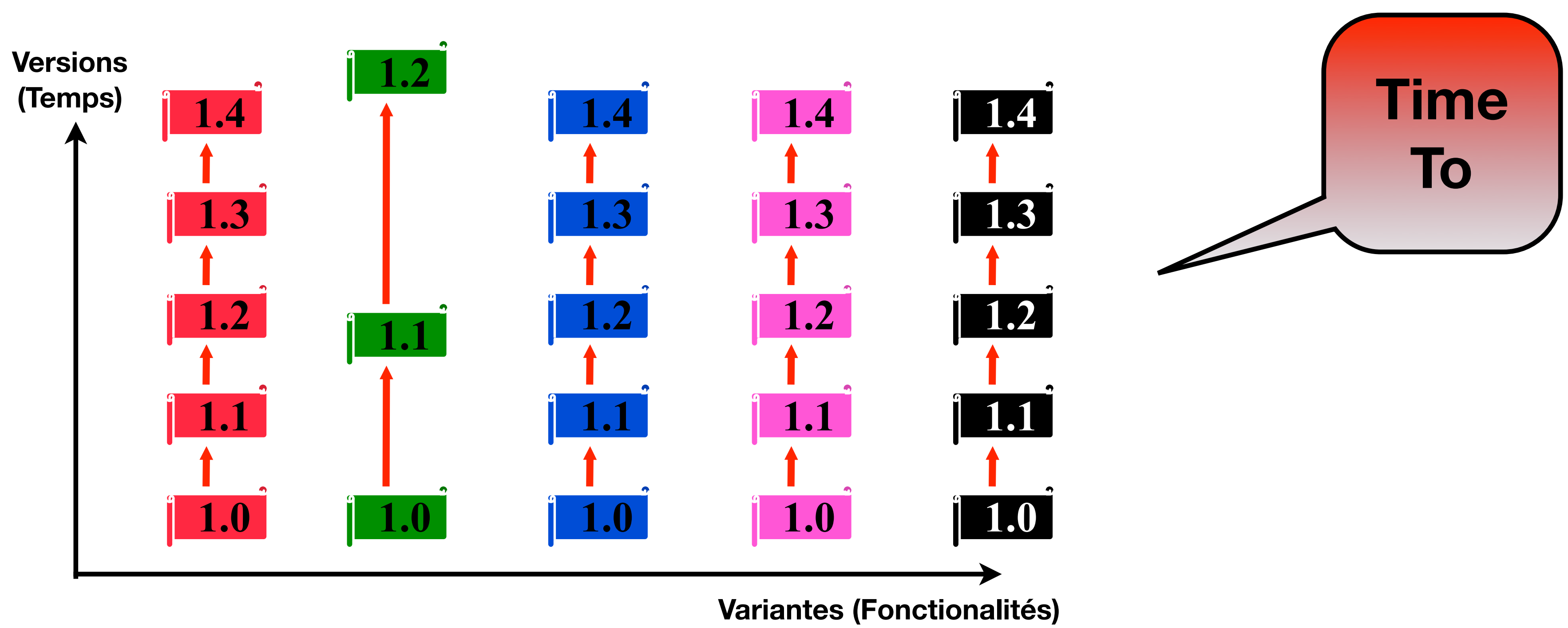
- Autonomic Computing
- Cloud Computing
- SaaS, IoS, IoT



Modern Software Complexity



- Importance des aspects non fonctionnels
  - systèmes répartis, parallèles et asynchrones
  - qualité de service : fiabilité, latence, performances...
- Flexibilité accrue des aspects fonctionnels
  - notion de lignes de produits (espace, temps)

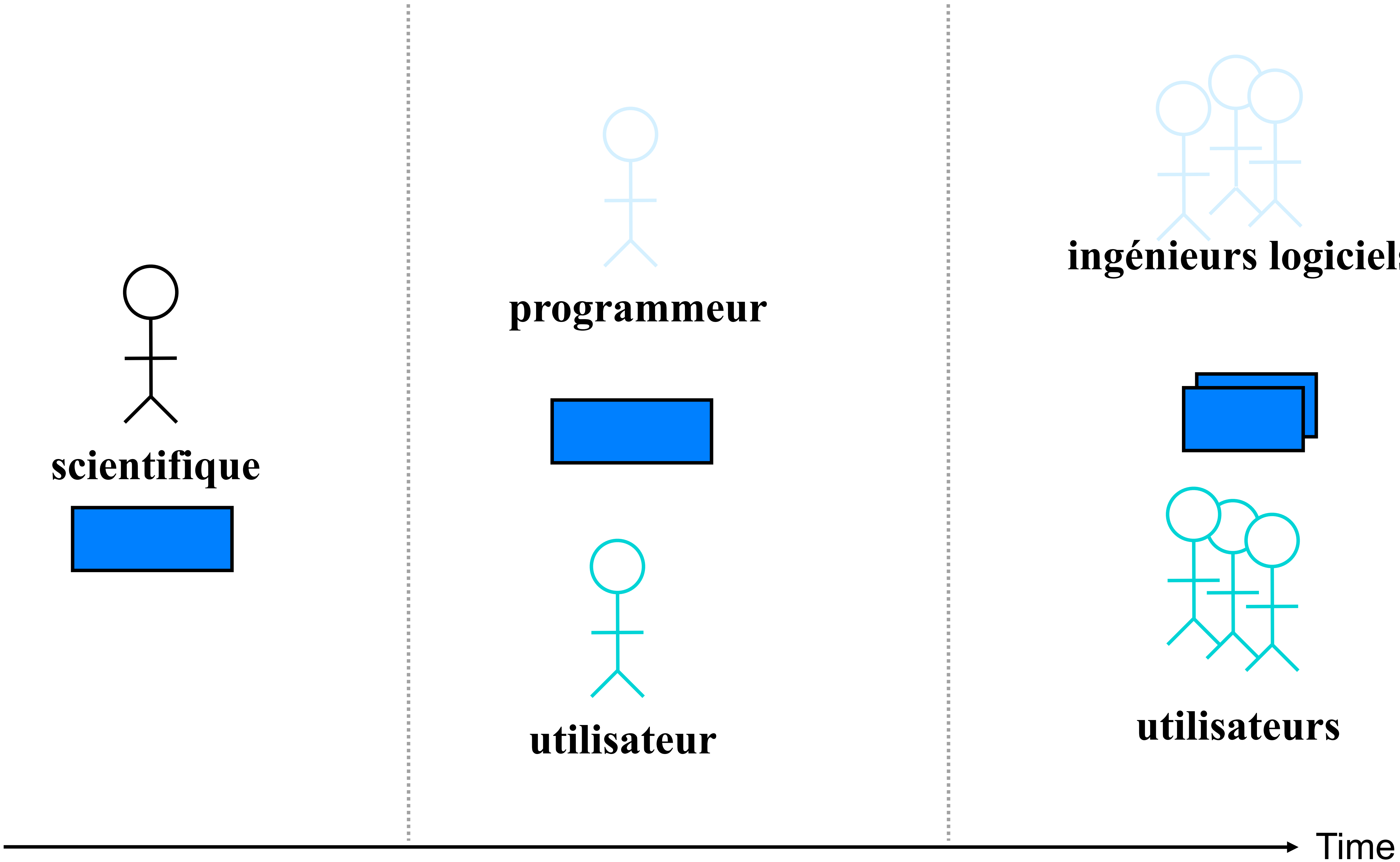


Some Areas of Complexity

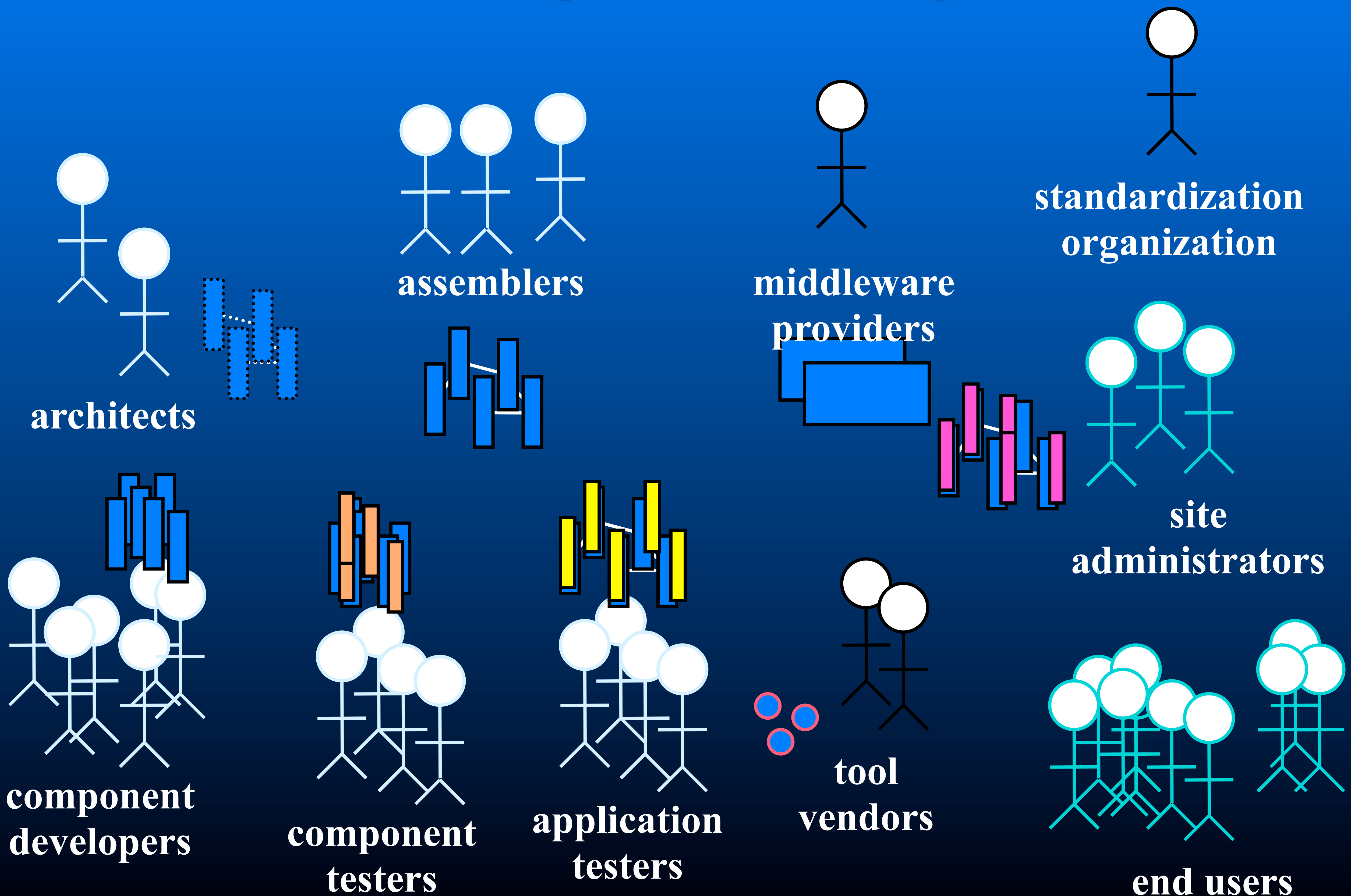
critical, real-time, embedded, distributed, parameterized, reusable, interoperable, durable, large-scale, pervasive, dynamic (self) adaptable, autonomous ...

The Software Industry Today

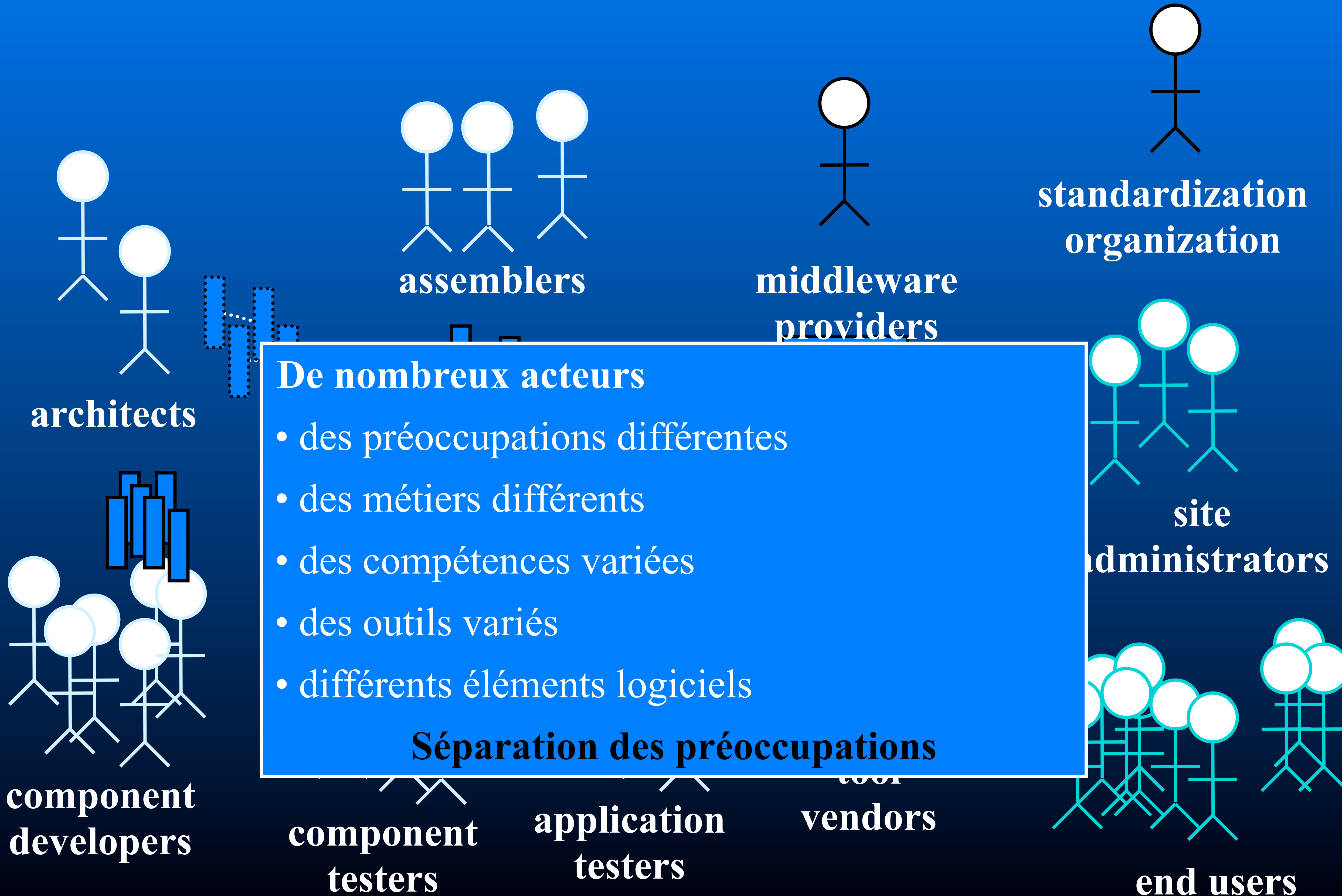
Évolution des acteurs



L'industrie logicielle aujourd'hui



L'industrie logicielle aujourd'hui

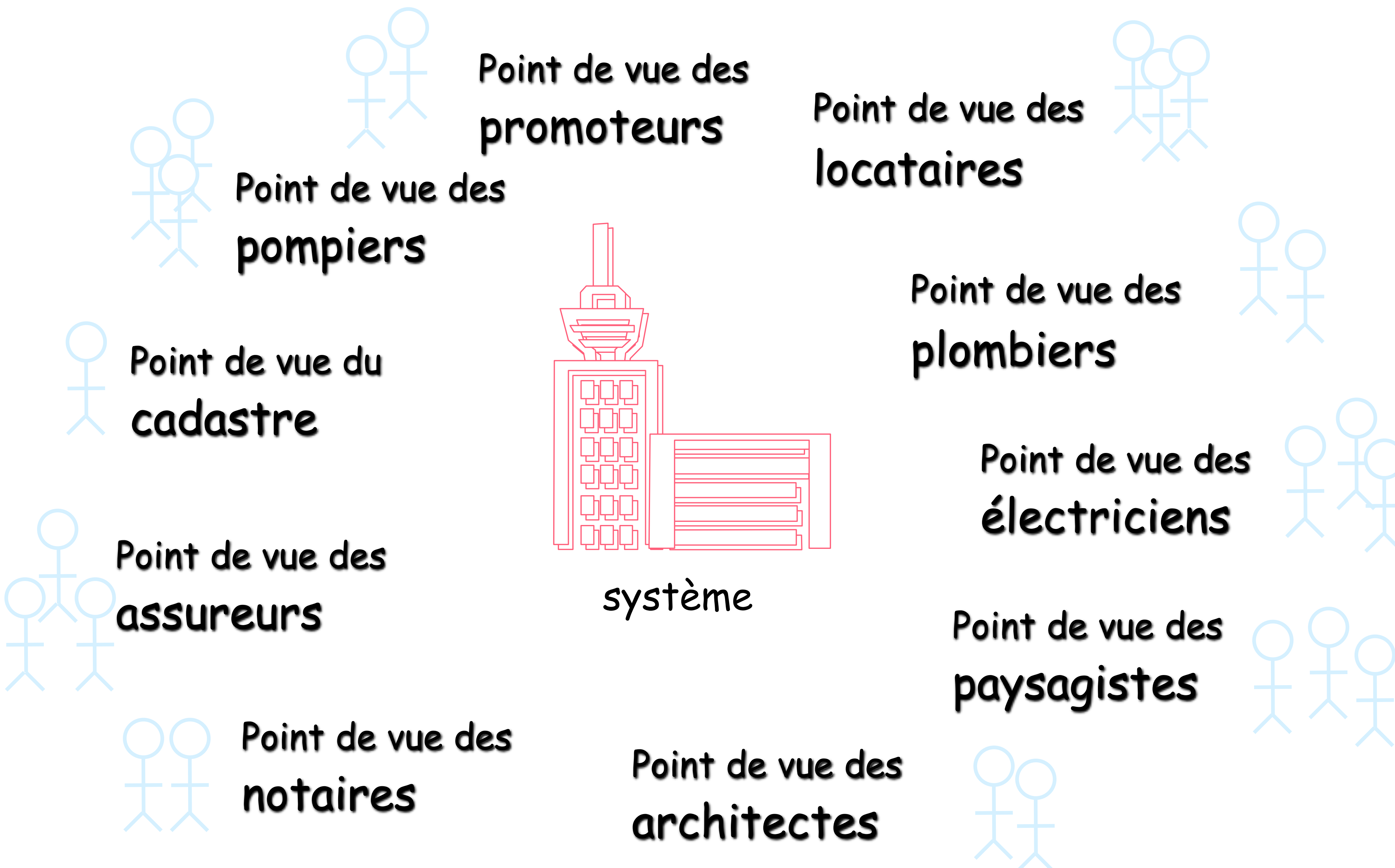


De nombreux acteurs

- des préoccupations différentes
- des métiers différents
- des compétences variées
- des outils variés
- différents éléments logiciels

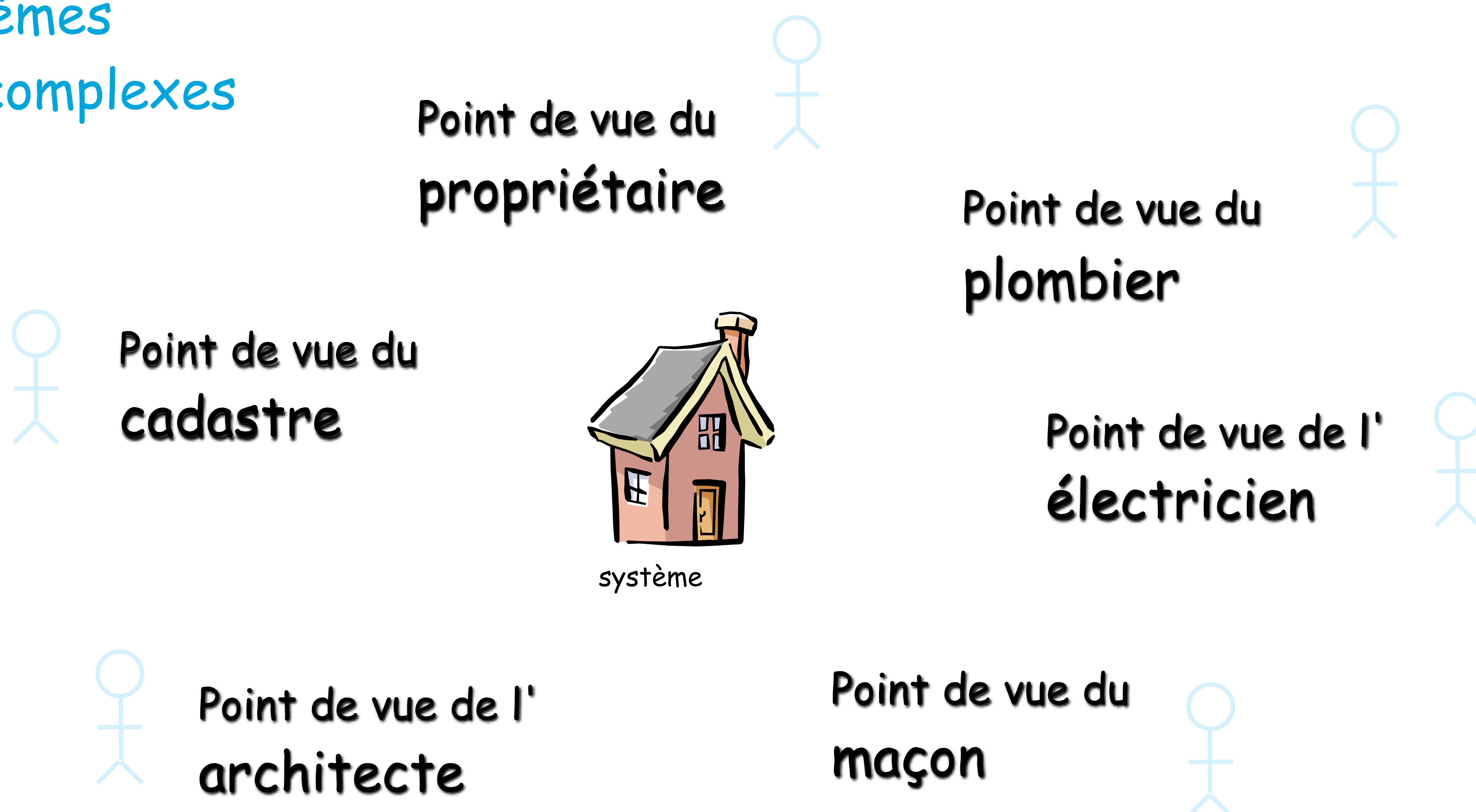
Séparation des préoccupations

Séparations des préoccupations



Séparations des préoccupations

Utile même pour des systèmes "moins" complexes

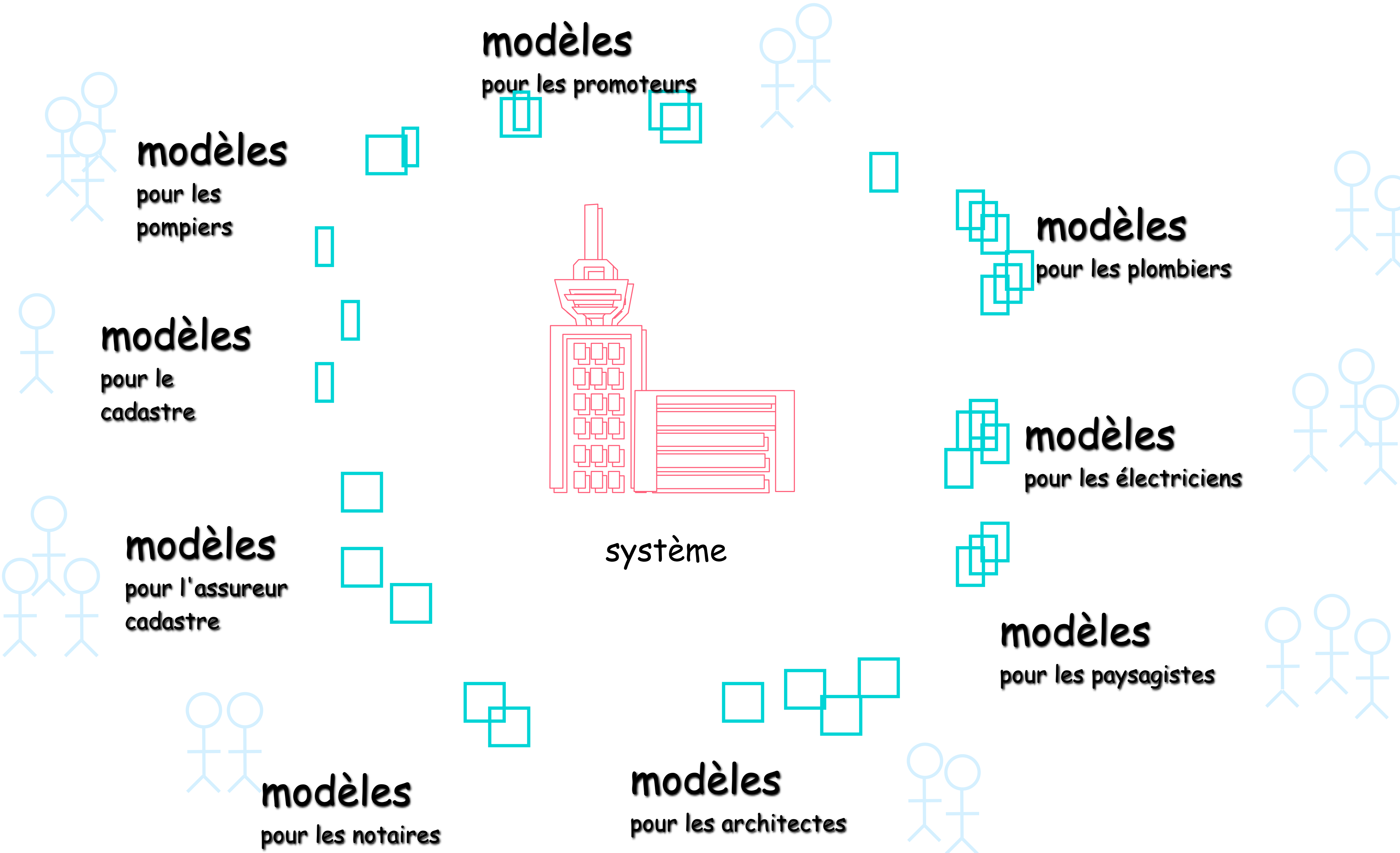


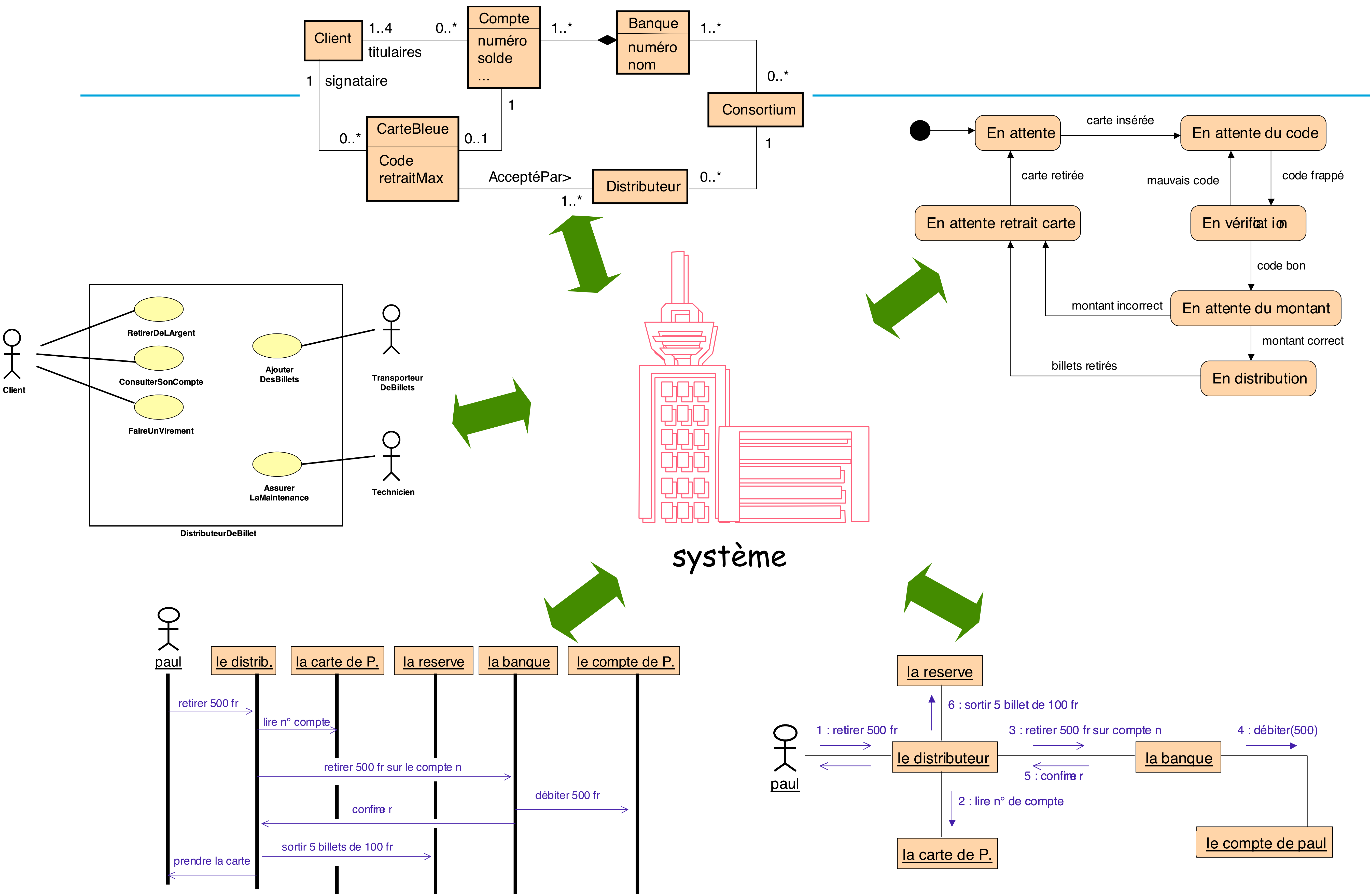
Problématique

- Complexité croissante des logiciels
- Séparations des préoccupations
- Séparations des métiers
- Multiplicité des besoins
- Multiplicité des plateformes
- Évolution permanente

Logiciel = Code ?
Est-ce la solution ?

Multiples modèles d'un système





Client
1..4
0..*
titulaires
Compte
numéro
solde
...
1..*
Banque
numéro
nom
1..*
1
signataire
0..*
Consortium
0..*
CarteBleue
Code
retraitMax
0..1
1
AcceptéPar>
Distributeur
1..*
0..*
1
En attente
carte insérée
En attente du code
carte retirée
mauvais code
code frappé
En attente retrait carte
En vérification
code bon
montant incorrect
En attente du montant
montant correct
billets retirés
En distribution
RetirerDeLArgent
ConsulterSonCompte
FaireUnVirement
Ajouter DesBillets
Assurer LaMaintenance
Client
Transporteur DeBillets
Technicien
DistributeurDeBillet
système
paul
le distrib.
la carte de P.
la reserve
la banque
le compte de P.
retirer 500 fr
lire n° compte
retirer 500 fr sur le compte n
débiter 500 fr
confirmer
sortir 5 billets de 100 fr
prendre la carte
la reserve
6 : sortir 5 billet de 100 fr
1 : retirer 500 fr
3 : retirer 500 fr sur compte n
4 : débiter(500)
paul
le distributeur
la banque
5 : confirmer
2 : lire n° de compte
la carte de P.
le compte de paul

Why modeling: master complexity

- Modeling, in the broadest sense, is the *cost-effective use of something in place of something else for some cognitive purpose*. It allows us to use something that is *simpler*, *safer* or *cheaper* than reality instead of reality for some purpose.

- A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.

Jeff Rothenberg, «The Nature of Modeling», 1989.

Modeling in Science & Engineering

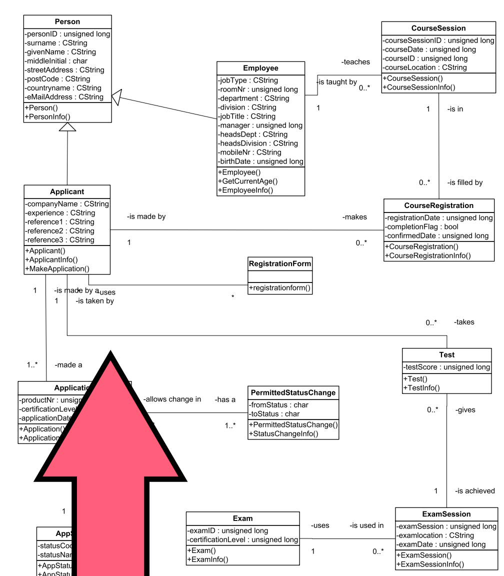
A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*

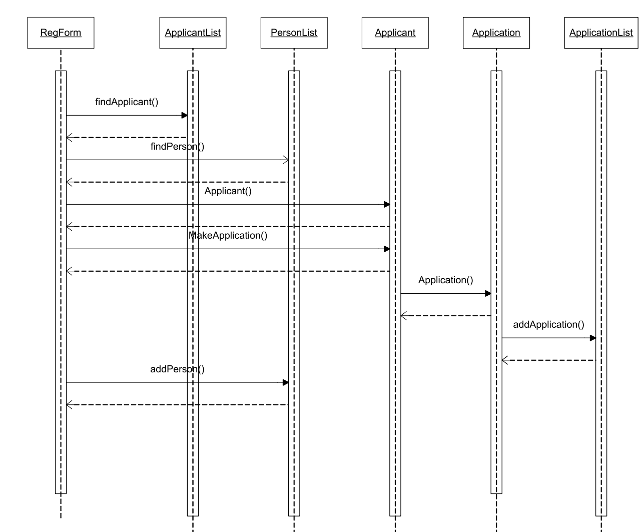
Specificity of Engineering: Model something not yet existing (in order to build it)

M_1

(modeling space)

Is represented by





M_0

(the world)

36

Des Modèles plutôt que du Code

- Un modèle est une simplification/abstraction de la réalité
- Nous construisons donc des modèles afin de mieux comprendre les systèmes que nous développons
- Nous modélisons des systèmes complexes parce que nous somme incapables de les comprendre dans leur totalité
- Le code ne permet pas de simplifier/abstraire la réalité

La modélisation: qu'elle utilisation ?

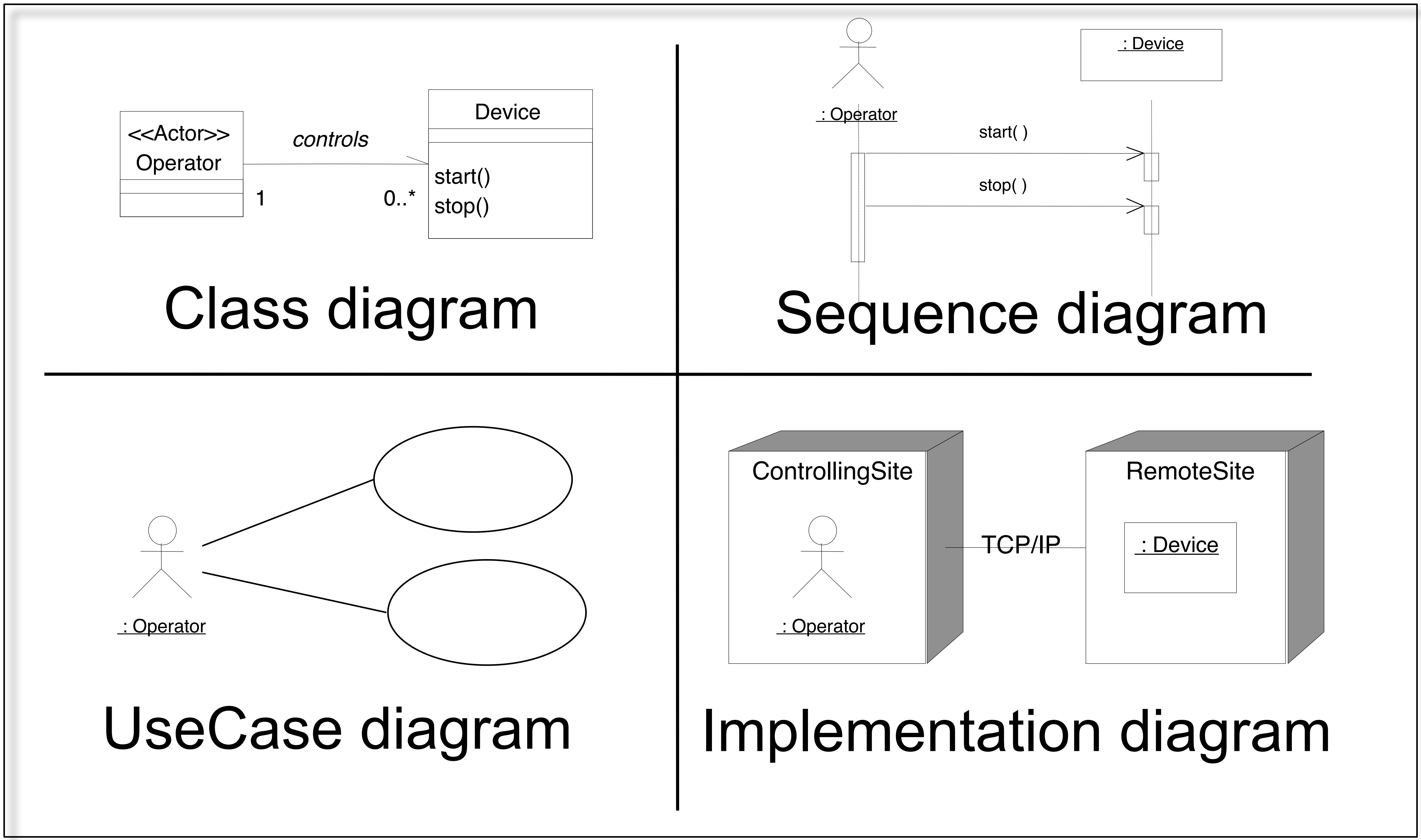
- Pour réfléchir :
  - représentation abstraite
  - séparation des préoccupations
- Pour communiquer :
  - représentation graphique
  - génération de documentation
- Pour automatiser le développement :
  - génération de code
  - application de patrons
  - migration
- Pour vérifier :
  - validation et vérification de modèles (e.g., simulation, model-checking...)
  - model-based testing

Synthèse

- Des Modèles plutôt que du Code
  - Un modèle est la simplification/abstraction de la réalité
  - Nous construisons donc des modèles afin de mieux comprendre les systèmes que nous développons
  - Nous modélisons des systèmes complexes parce que nous somme incapables de les comprendre dans leur totalité
  - Le code ne permet pas de simplifier/abstraire la réalité

UML: one model,

4 main dimensions, multiple views



The X diagrams of UML

- Modeling along 4 main viewpoints:
  - Static Aspect (*Who?*)
    - Describes objects and their relationships (Class & Object Diagrams)
    - Structuring with packages
  - User view (*What?*)
    - Use cases Diagram
  - Dynamic Aspects (*When?*)
    - Sequence Diagram
    - Collaboration Diagram
    - State Diagram
    - Activity Diagram
  - Implementation Aspects (*Where?*)
    - Component Diagram & deployment diagram

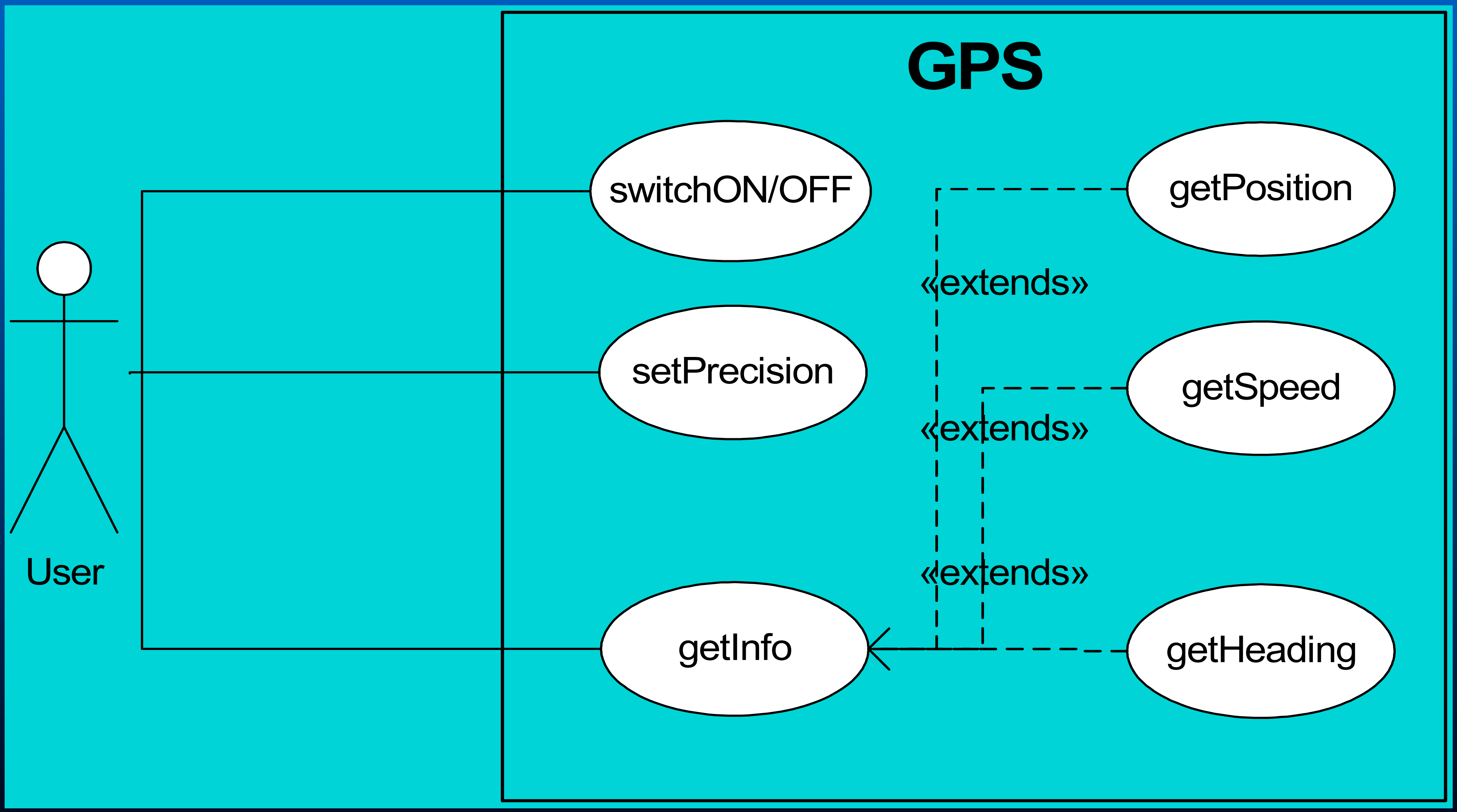
Example

- Modeling a (simplified) GPS device
  - Get position, heading and speed
    - » by receiving signals from a set of satellites
  - Notion of Estimated Position Error (EPE)
    - » Receive from more satellites to get EPE down
  - User may choose a trade-off between EPE & saving power
    - » Best effort mode
    - » Best route (adapt to speed/variations in heading)
    - » PowerSave

(Case Study borrowed from N. Plouzeau, K. Macedo & JP. Thibault. Big thanks to them)

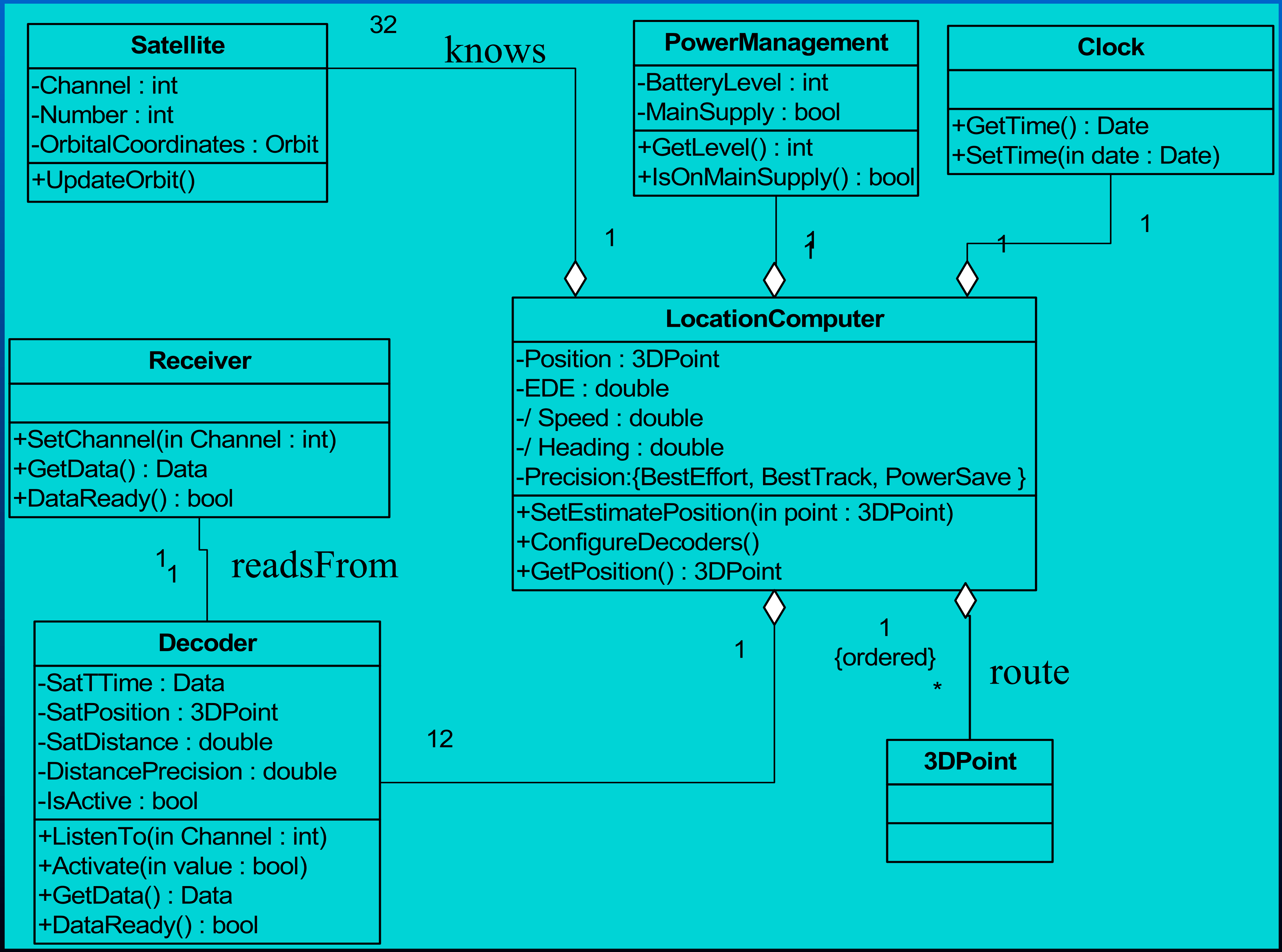
Modeling a (simplified) GPS device

- Use case diagram



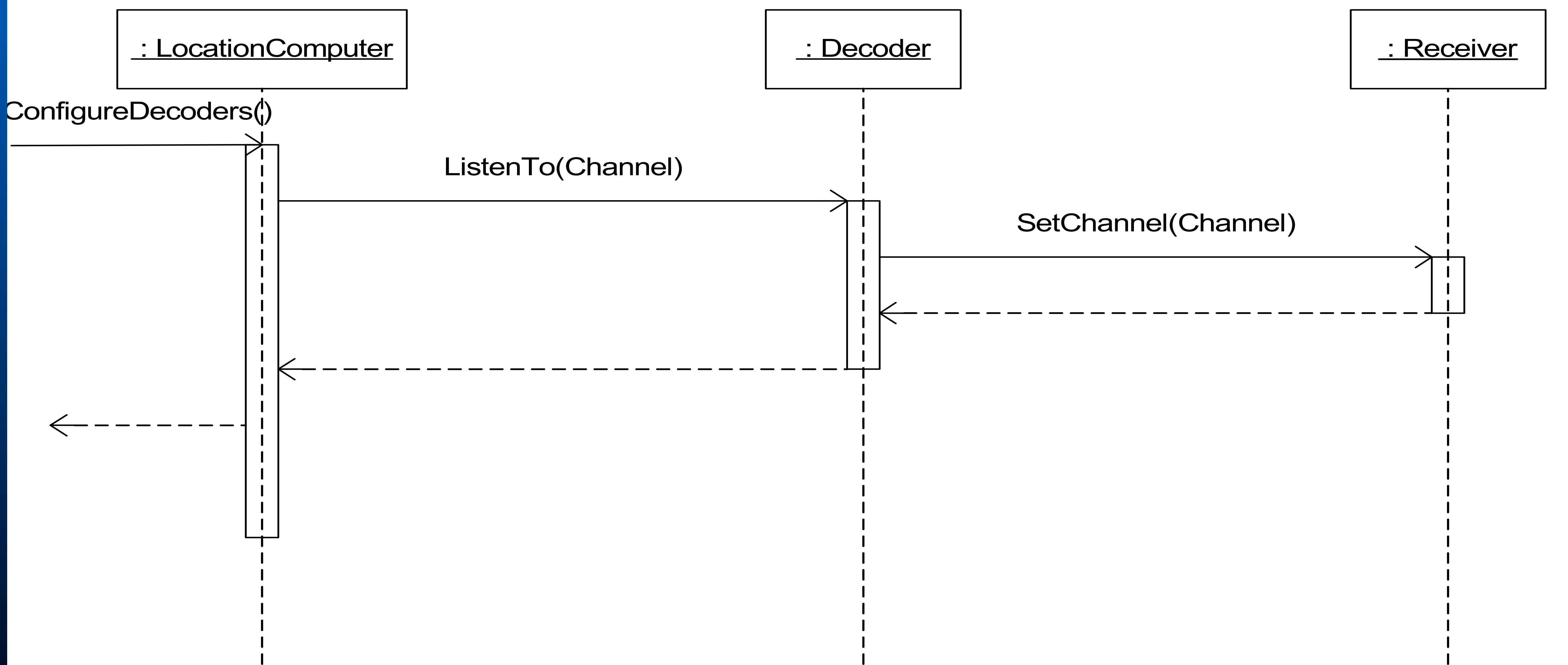
Modeling a (simplified) GPS device

- Class diagram



Modeling a (simplified) GPS device

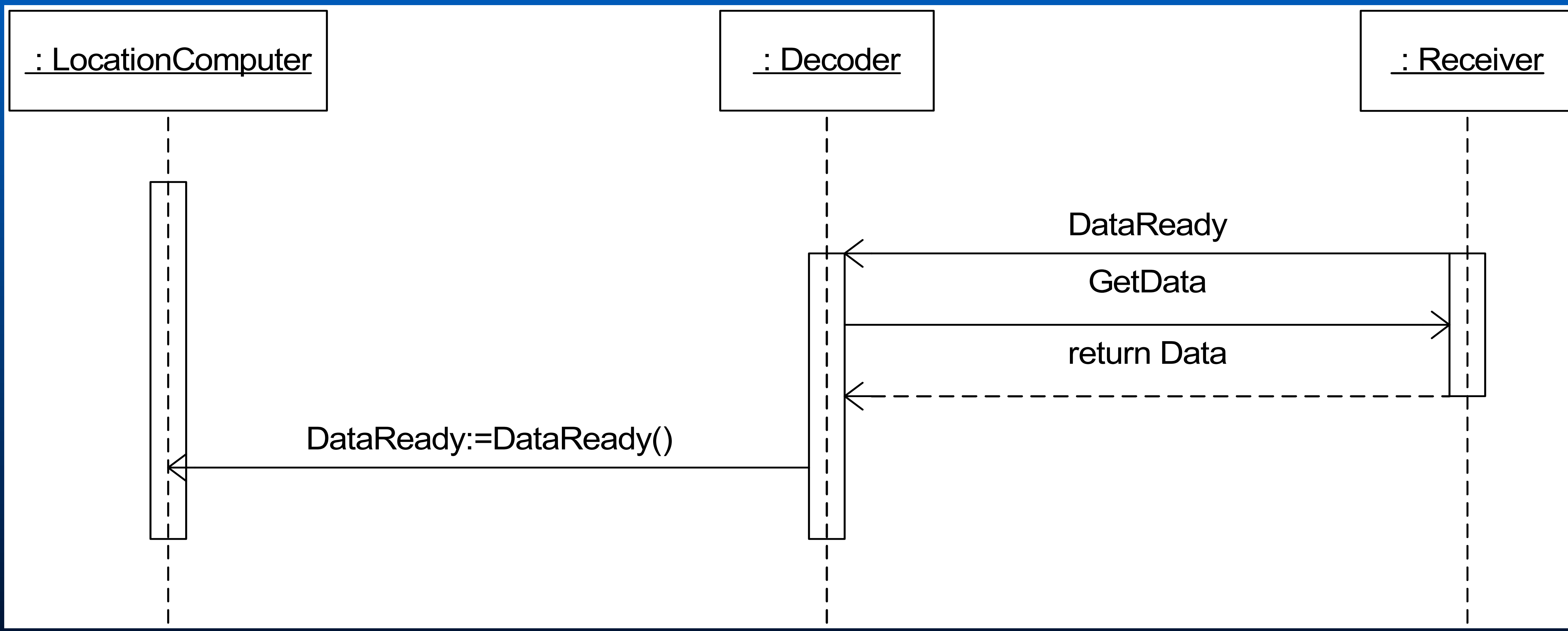
- Sequence diagram: configuring decoders



45

Modeling a (simplified) GPS device

- Sequence diagram: interrupt driven architecture



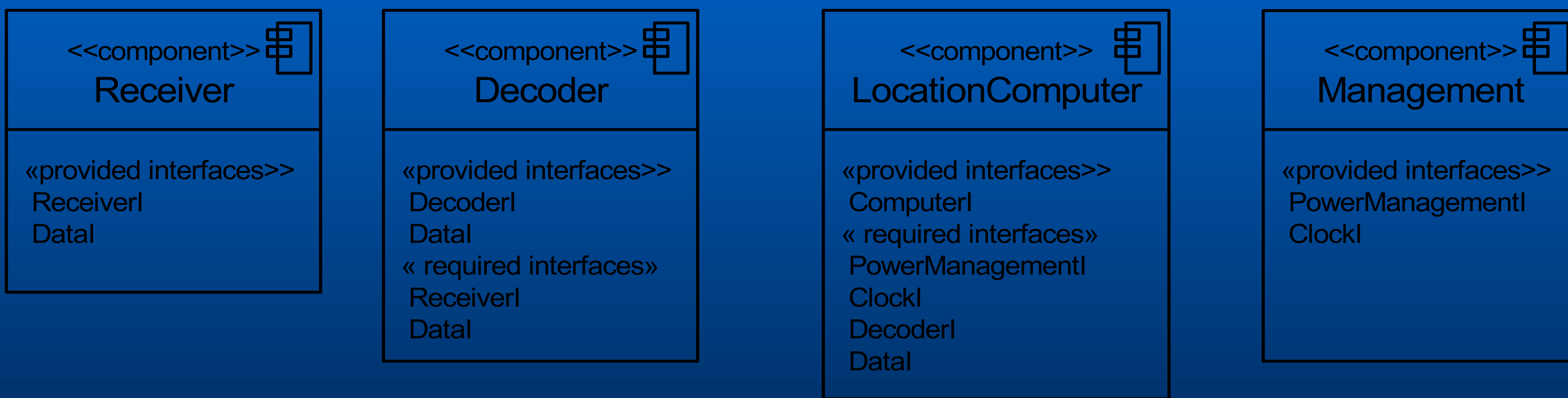
- Many more sequence diagrams needed…

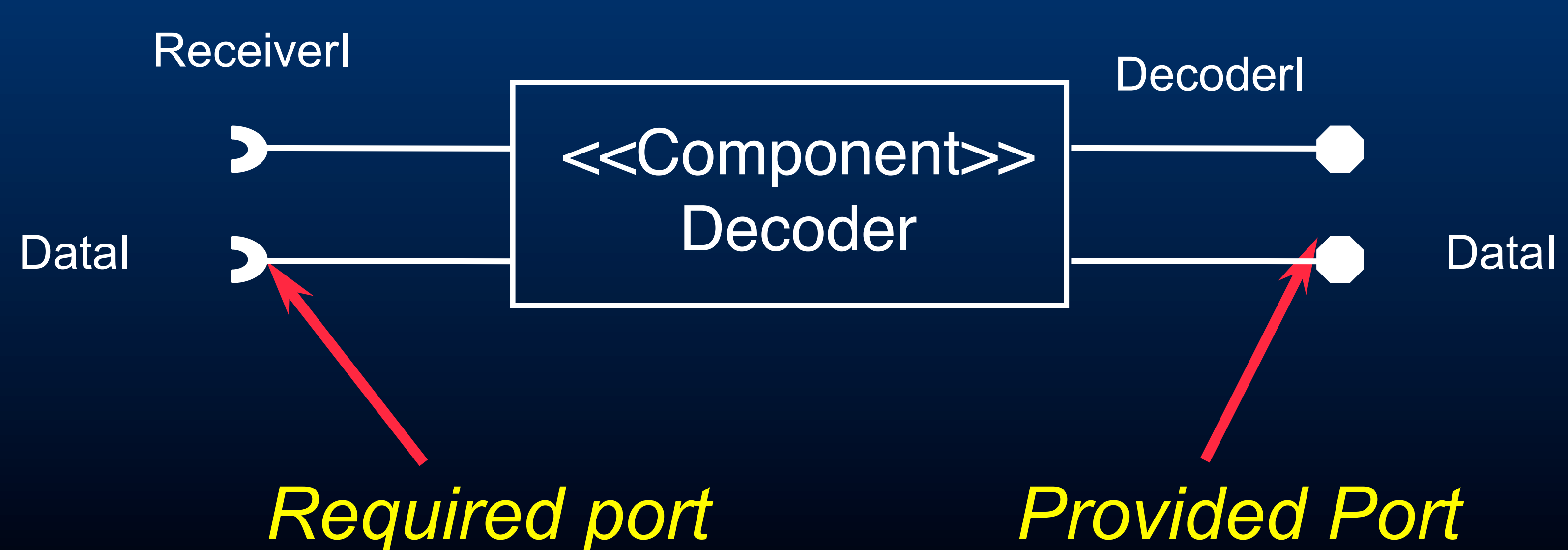
Modeling a (simplified) GPS device

- Targeting multiple products with the same (business) model
  - Hand held autonomous device
  - Plug-in device for Smart Phone
  - Plug-in device for laptop (PCMCIA/USB)
  - May need to change part of the software after deployment
- We choose a component based delivery of the software

Modeling a (simplified) GPS device

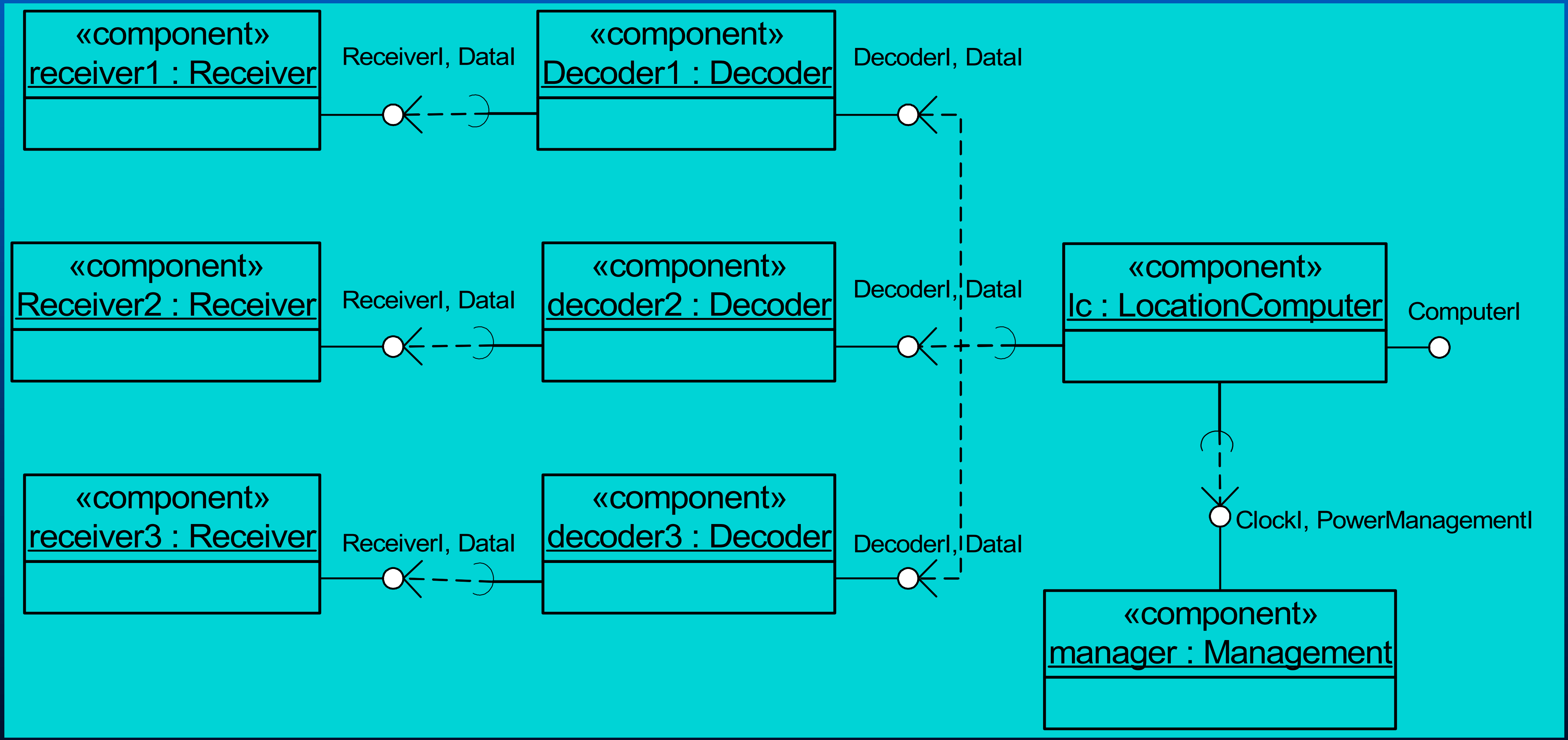
- Component diagram

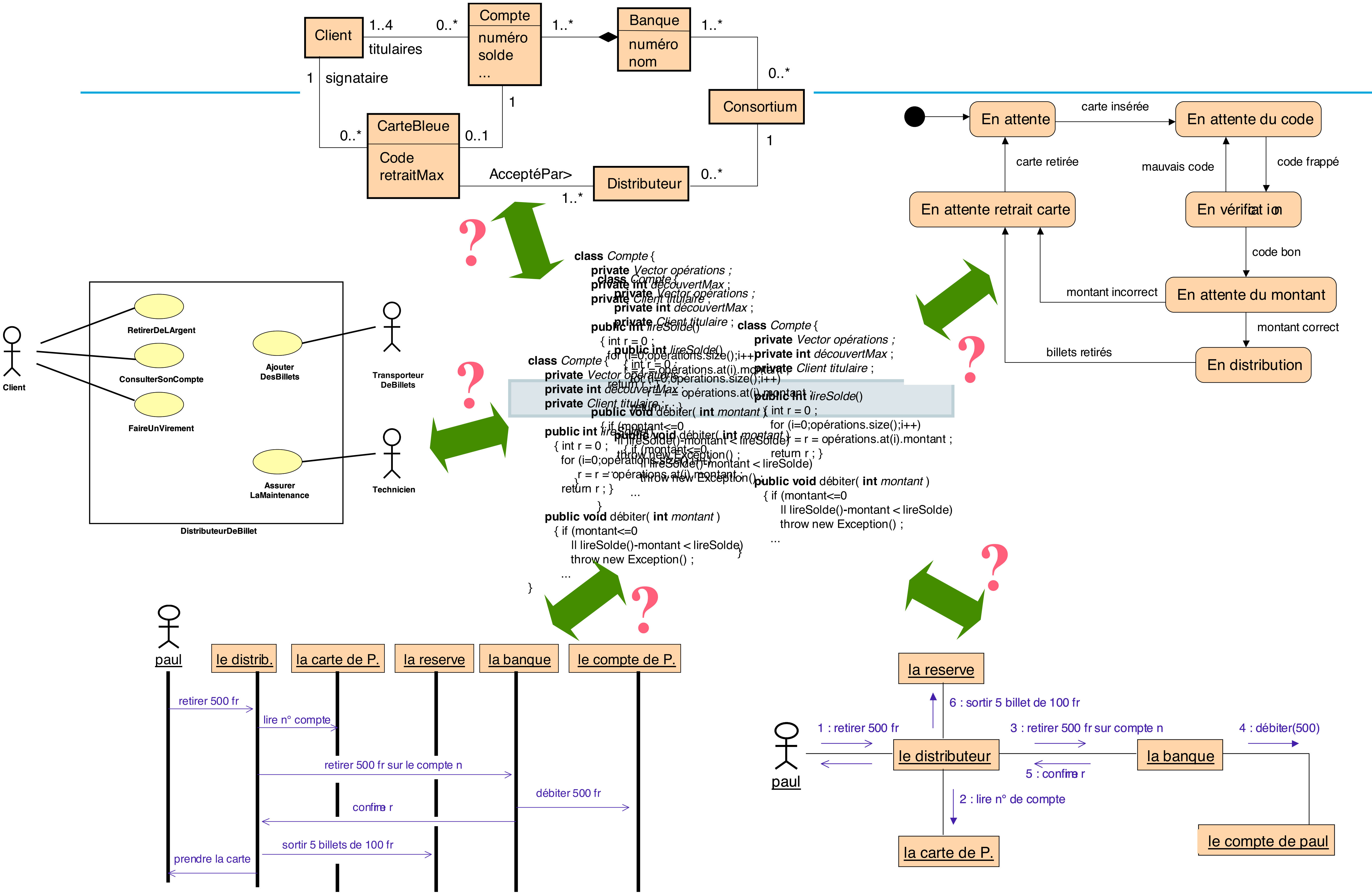




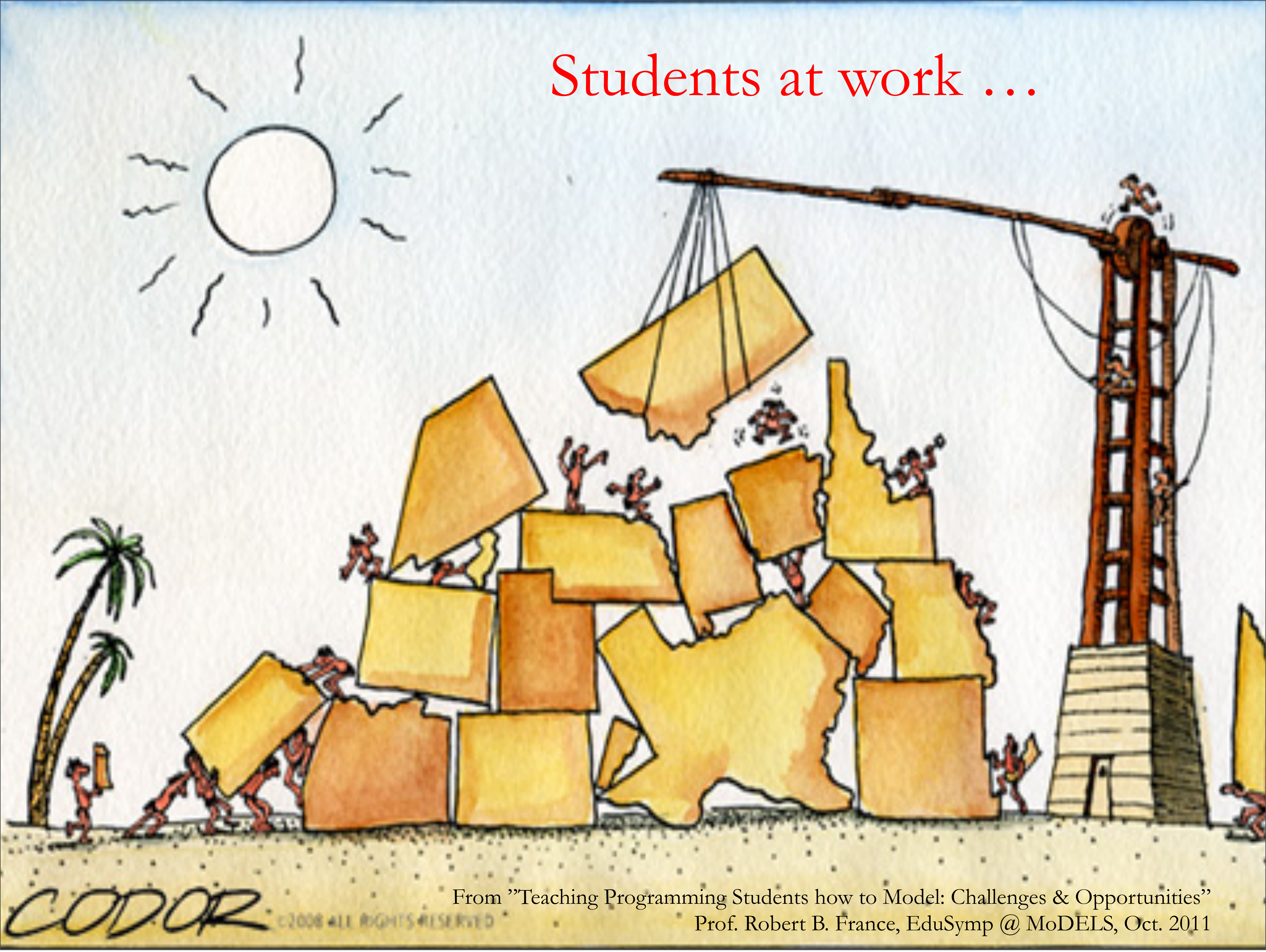
Modeling a (simplified) GPS device

- Deployment diagram




Client
Compte
numéro
solde
...
Banque
numéro
nom
Consortium
CarteBleue
Code
retraitMax
Distributeur
1..4
0..*
titulaires
1..*
1
signataire
0..1
AcceptéPar>
RetirerDeLArgent
ConsulterSonCompte
FaireUnVirement
Ajouter
DesBillets
Assurer
LaMaintenance
Transporteur
DeBillets
Technicien
DistributeurDeBillet
En attente
carte insérée
En attente du code
carte retirée
mauvais code
code frappé
En attente retrait carte
code bon
montant incorrect
En attente du montant
montant correct
billets retirés
En distribution
class Compte {
private Vector opérations ;
private int découvertMax ;
private Client titulaire ;
public int lireSolde()
{ int r = 0 ;
for (i=0;opérations.size();i++)
r = r = opérations.at(i).montant ;
return r ; }
public void débiter(int montant)
{ if (montant<=0
ll lireSolde()-montant < lireSolde)
throw new Exception() ;
...
}
paul
le distrib.
la carte de P.
la reserve
la banque
le compte de P.
retirer 500 fr
lire n° compte
retirer 500 fr sur le compte n
débiter 500 fr
confirmer
sortir 5 billets de 100 fr
prendre la carte
1 : retirer 500 fr
le distributeur
2 : lire n° de compte
3 : retirer 500 fr sur compte n
4 : débiter(500)
5 : confirmer
6 : sortir 5 billet de 100 fr
le compte de paul

Students at work …



From "'Teaching Programming Students how to Model: Challenges & Opportunities"
Prof. Robert B. France, EduSymp @ MoDELS, Oct. 2011

From ”Teaching Programming Students how to Model: Challenges & Opportunities”
Prof. Robert B. France, EduSymp @ MoDELS, Oct. 2011

“Use of modeling techniques distinguishes a software engineer from a software developer (or programmer)”

“The earlier you start to code the longer it takes to complete the program”

**“A good modeler is a good programmer;
a good programmer is not always a good modeler”**

**“Learning a programming language is easy,
learning how to program is difficult”**

Prof. Robert B. France
Colorado State University
http://www.cs.colostate.edu/~france/

Model and Reality in Software

- **Sun Tse:** *Do not take the map for the reality*
- **William James:** *The concept 'dog' does not bite*
- **Magritte:**

- Software Models: from contemplative to productive

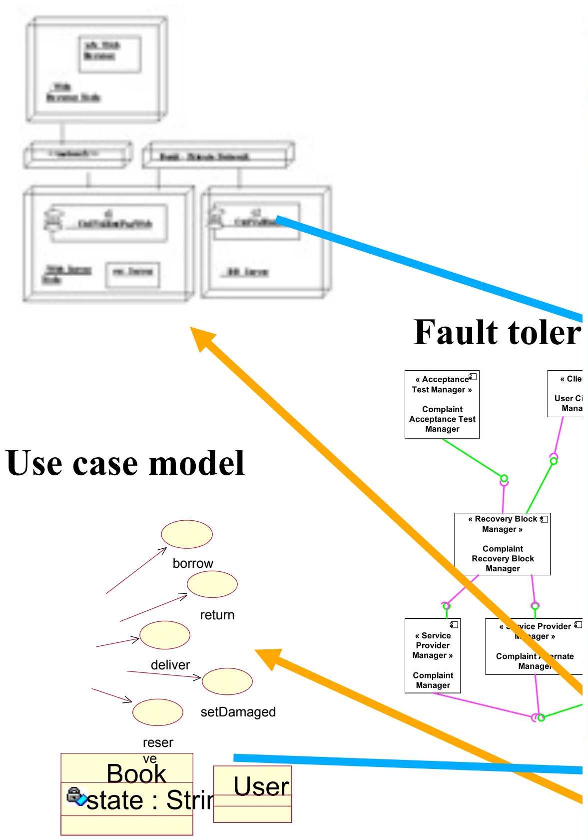
Mais sinon...

Distribution

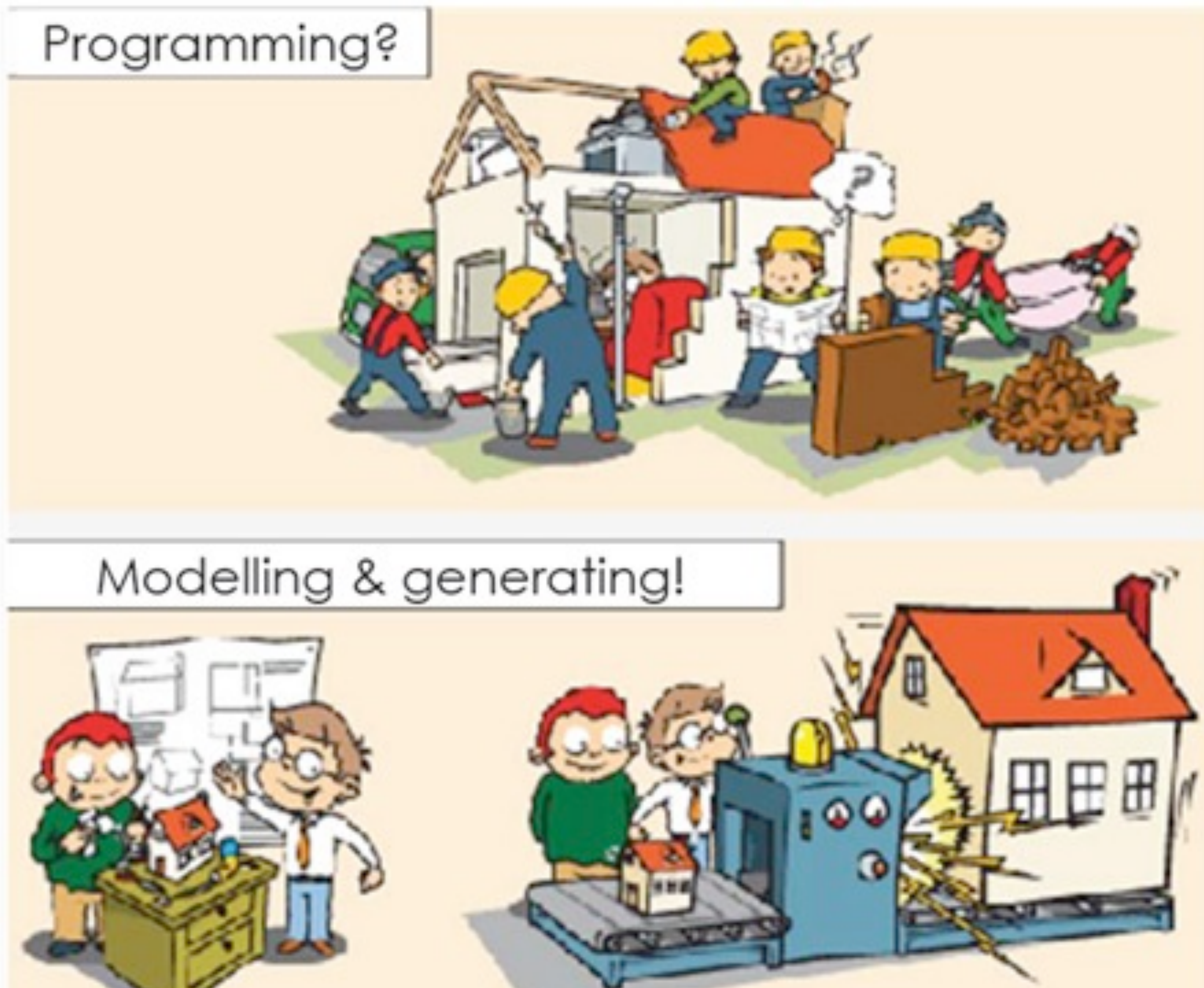
Security

Fault toler

Use case model



AOMDE = Pleonasm because
*a **Model** is an **Abstraction** of an*
*r a given **Purpose***



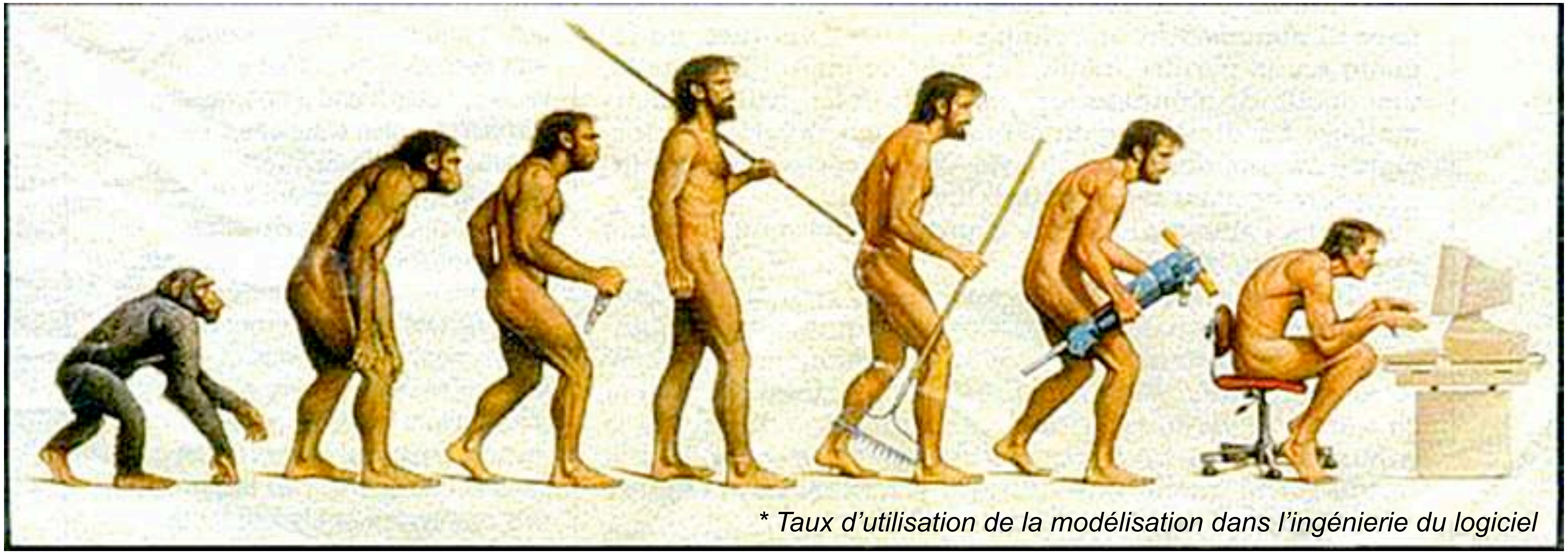
A metaphor for Model Driven Engineering, J. Haan, 2009.

Design Model

Model

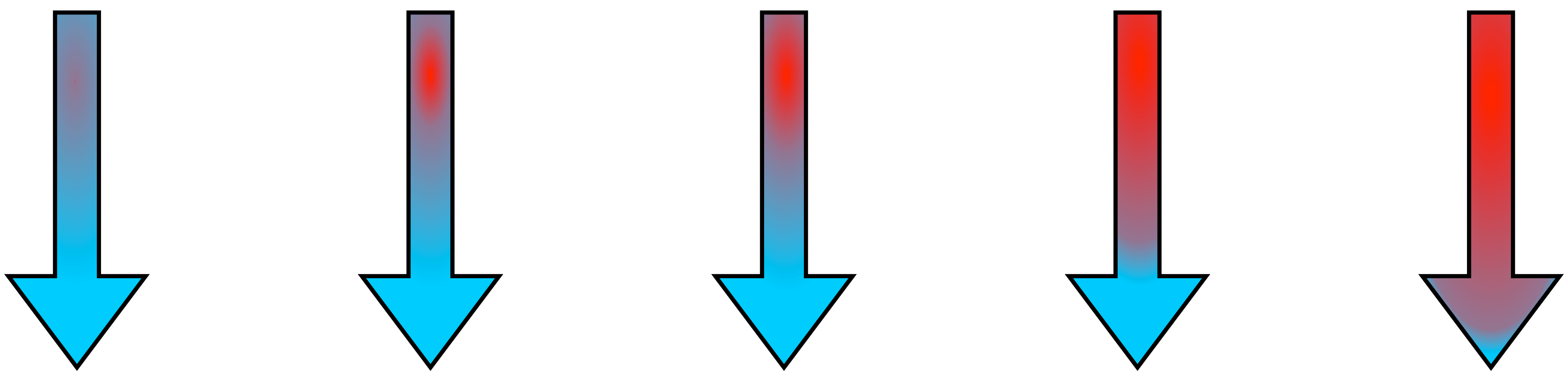
La modélisation dans l’ingénierie du logiciel

ESIR – ECOLE SUPERIEURE D’INGENIEURS DE RENNES



* Taux d’utilisation de la modélisation dans l’ingénierie du logiciel

Analysis
Design
Development
Test
Deployment
Maintenance
Runtime



?

design, documentation

analysis ... detailed design

requirement management, systems engineering ... V&V, (code, test) generation.

... DSML, transformation/ composition ... maintenance ... runtime.

ET VOUS ?

état de la pratique dans l’industrie

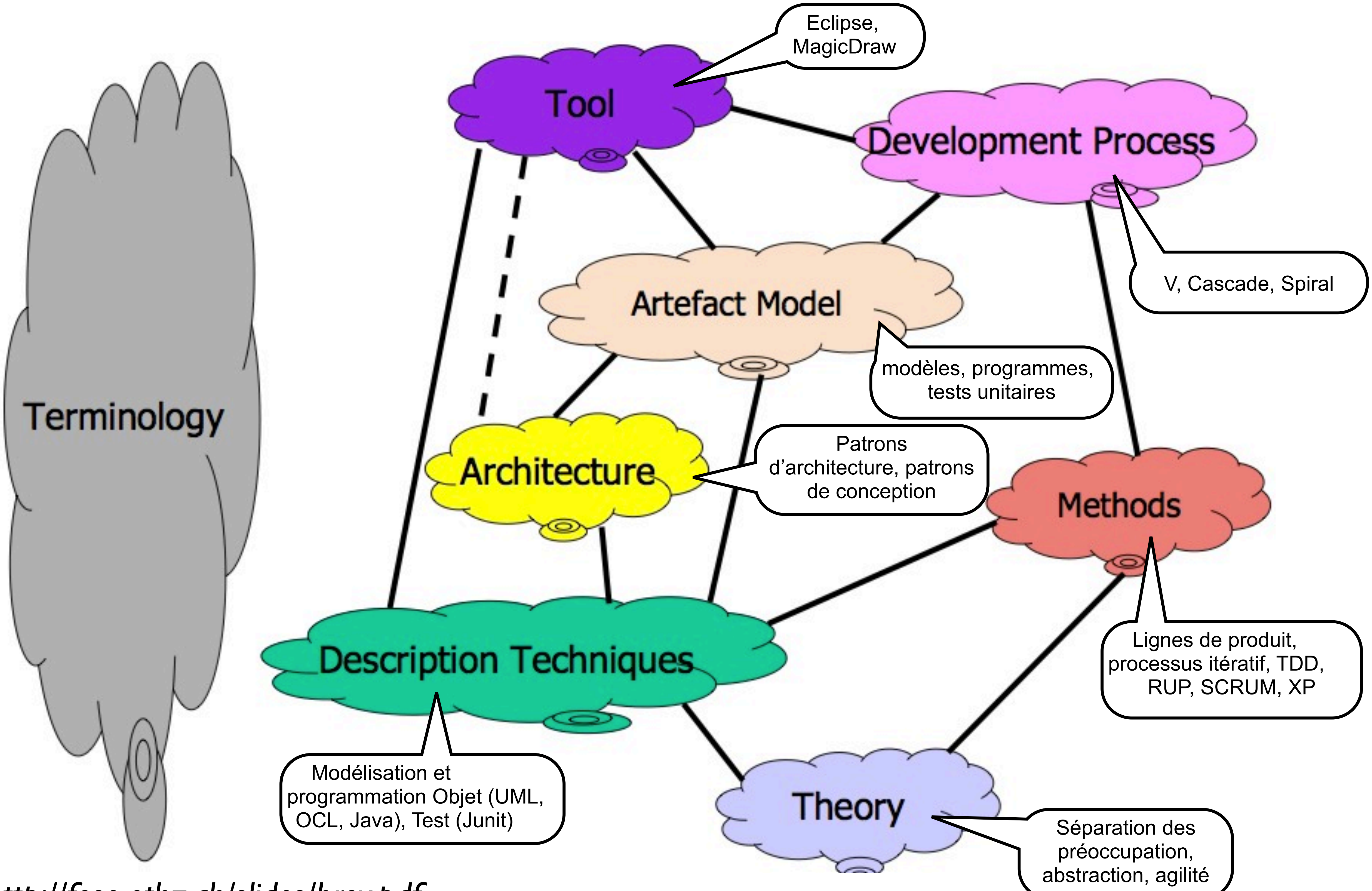
Software Development Process

Software Engineering Components

- Development Methods
- Techniques / Best Practices
- Languages
- Tools

Seamless Method- and Model-based Software and Systems Engineering

The Future of Software Engineering Symposium
22-23 November 2010, ETH Zurich



from http://fose.ethz.ch/slides/broy.pdf

Activités
du développement de logiciels

Valider le logiciel

Assembler les composants

Développer un des composants

Définir comment il sera développé

Définir ce qui sera développé

- L'organisation de ces activités et leur enchaînement définit le *cycle de développement* du logiciel

Cycle de vie en V normalisé AFNOR

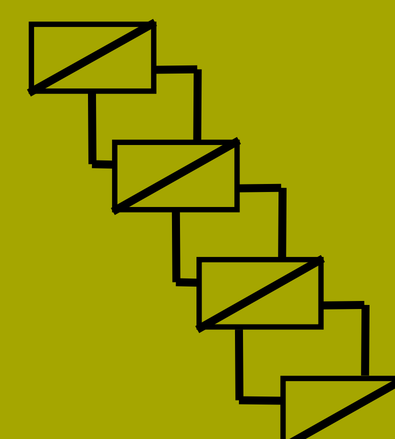
Analyse
- Expression du besoin
- Analyse détaillée

Conception
- Etude technique préalable
- Conception préliminaire
- Conception détaillée

Réalisation
- Codage
- Mise au point
- Tests unitaires

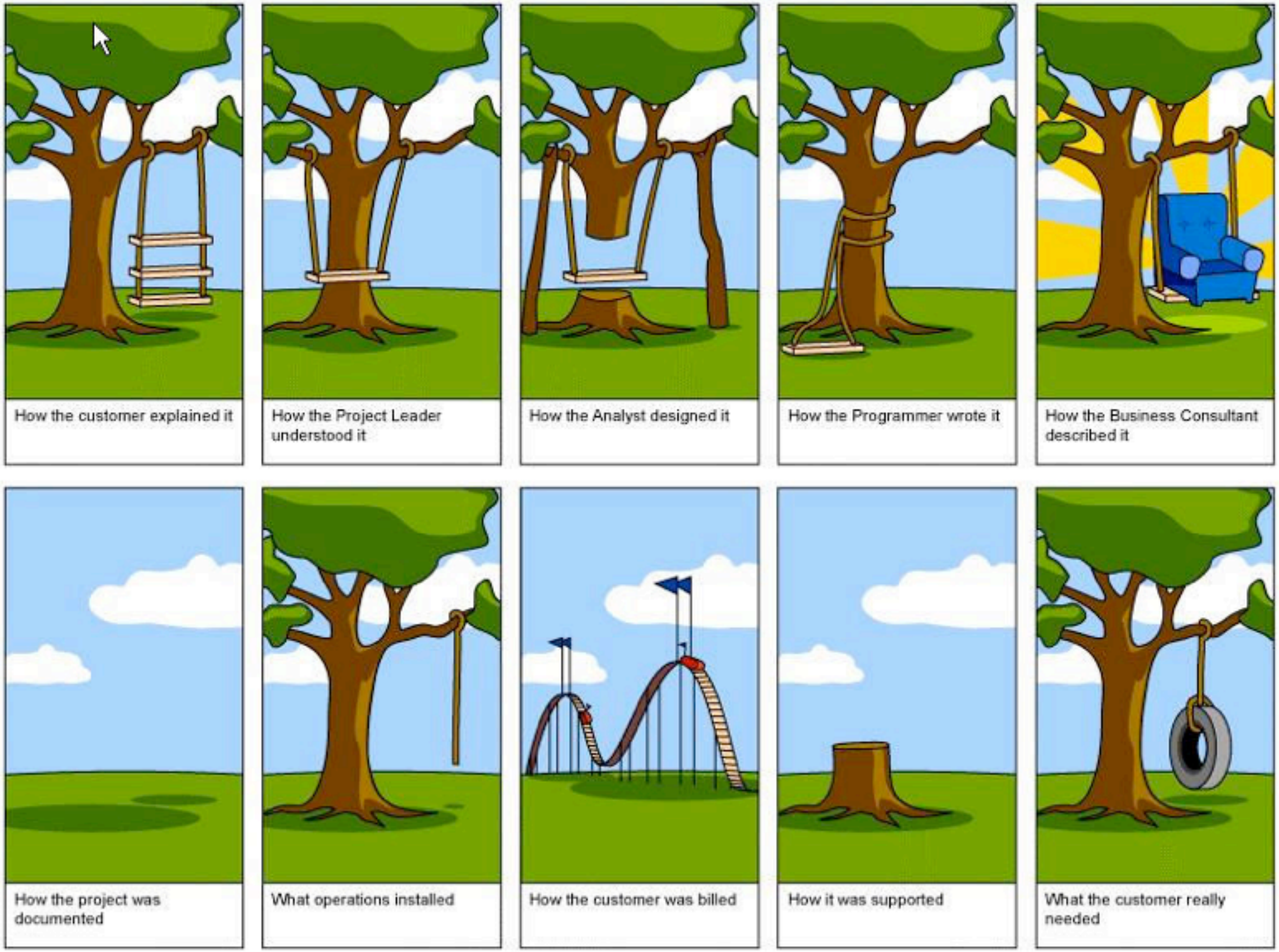
Intégration
- Intégration
- Tests d'Intégration

Validation
- Validation
- Mise en œuvre



Variante US :
Cycle en « cascade »

Constat...


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

61

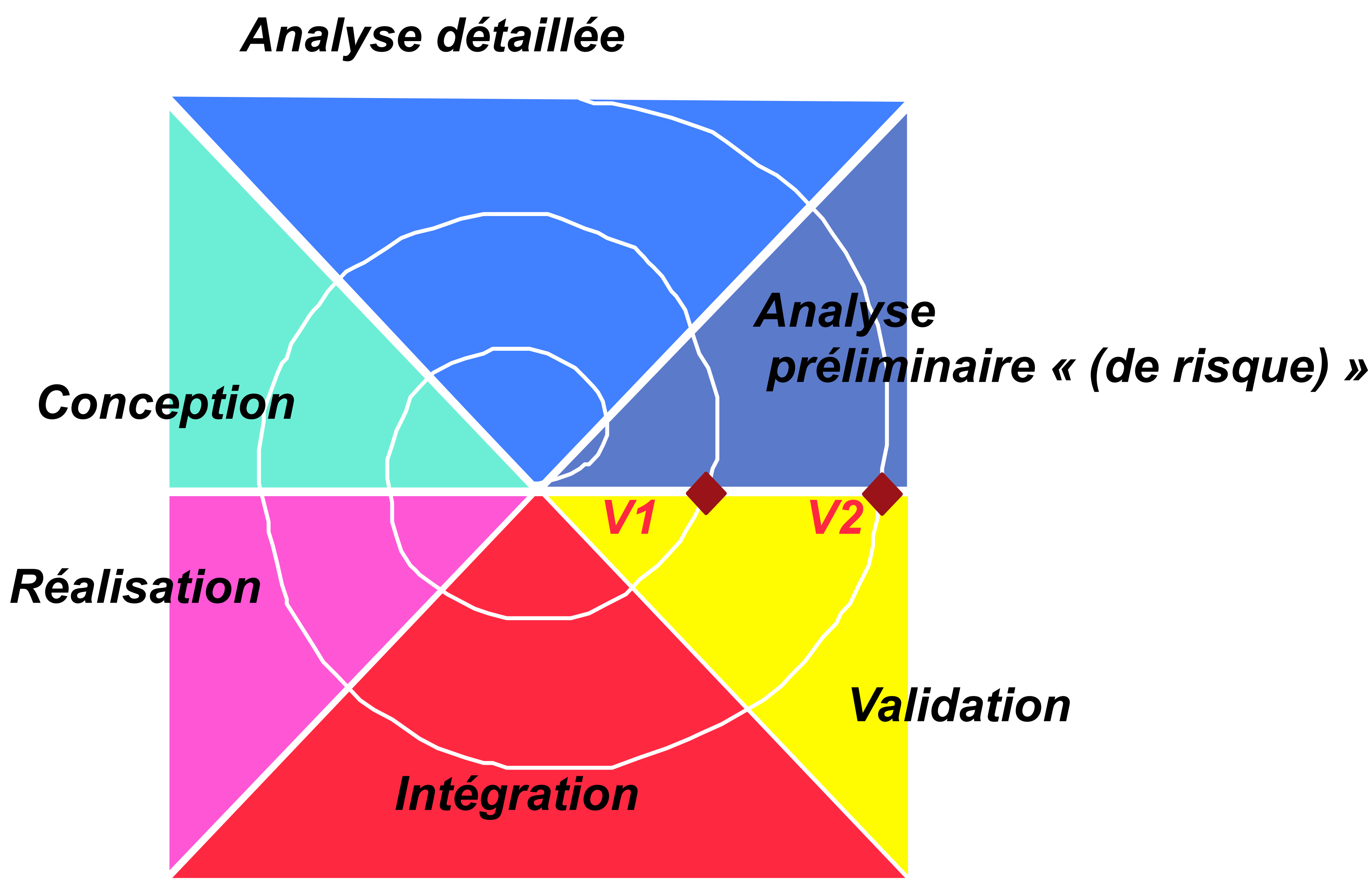
Why do projects fail so often

- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project's status
- Unmanaged risks
- Poor communication among customers, developers, and users
- Use of immature technology
- Inability to handle the project's complexity
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

Problèmes du processus classique

- Organisation « industrielle » héritée du XIXème siècle
  - rassurant pour les managers
  - hiérarchie malsaine dans les rôles
  - antinomie : Coplien's organizational pattern
    - Architects Also Implement
- cycle management <> cycle développement
- linéarité implicite
  - temps d'approbation des documents => effet tampon
  - coût de la (non-) modification d'un document « final »
  - irréaliste pour un projet innovant, donc à risques

Cycle de vie en « spirale »

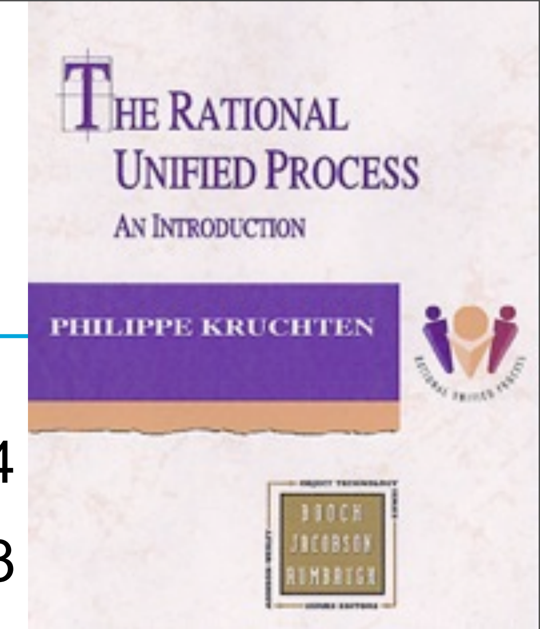


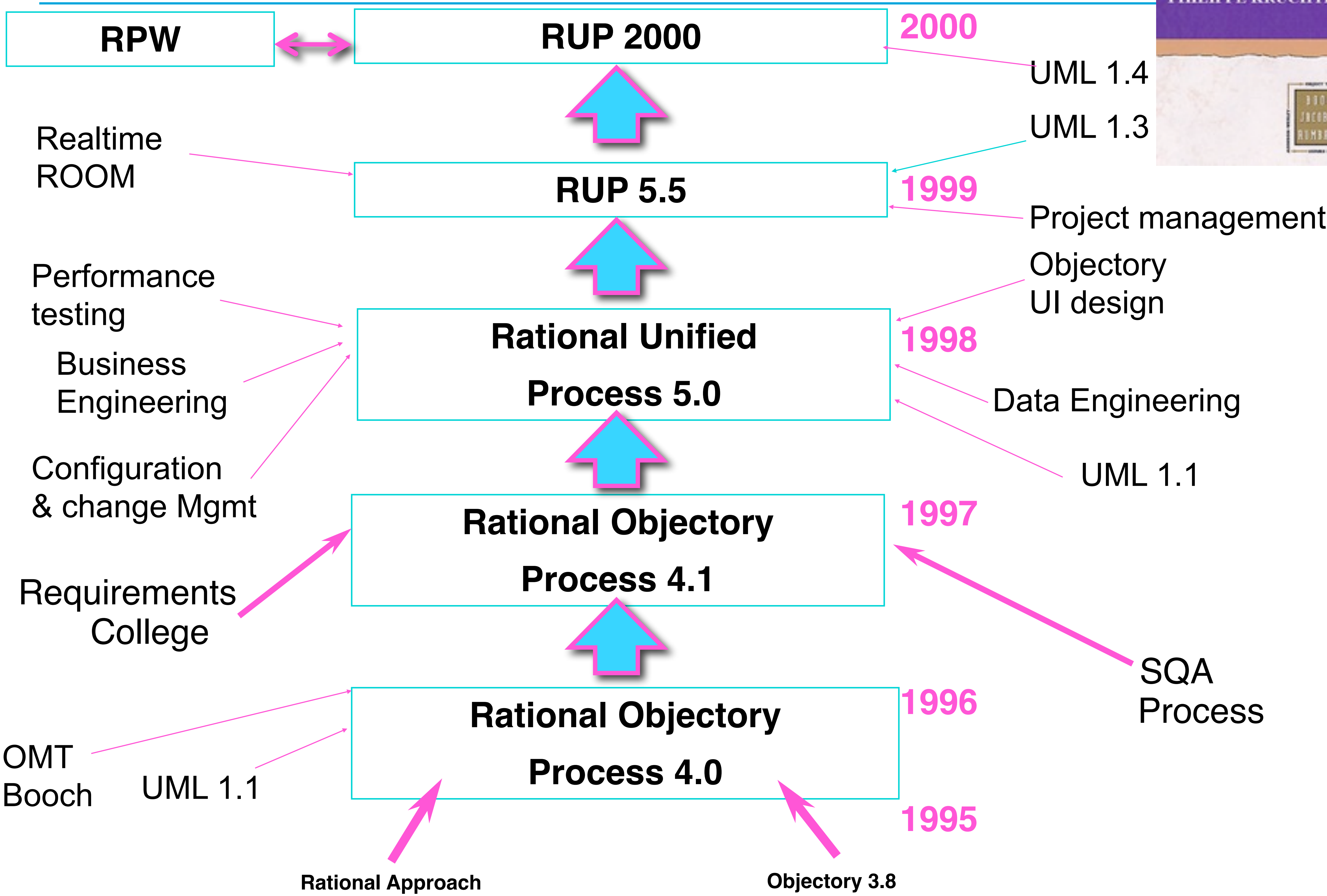
Synergie avec l'approche par objets

Intérêts du cycle de vie en « spirale »

- Bien adapté au développements innovants
  - les progrès sont tangibles : c'est du logiciel qui « tourne » et pas seulement des kilos de documents
  - possibilité de s'arrêter « à temps », i.e. avant que l'irréalisabilité du projet ait créée un gouffre financier
- Moins simple à manager
  - difficile à gérer en situation contractuelle
  - mal contrôlé => on retombe dans le hacking
- Production des incréments asservie sur 2 parmi 3 :
  - période (e.g. release toutes les 2 semaines)
  - fonctionnalités (releases découpés suivant use-cases)
  - niveau de qualité (problème de la mesure)

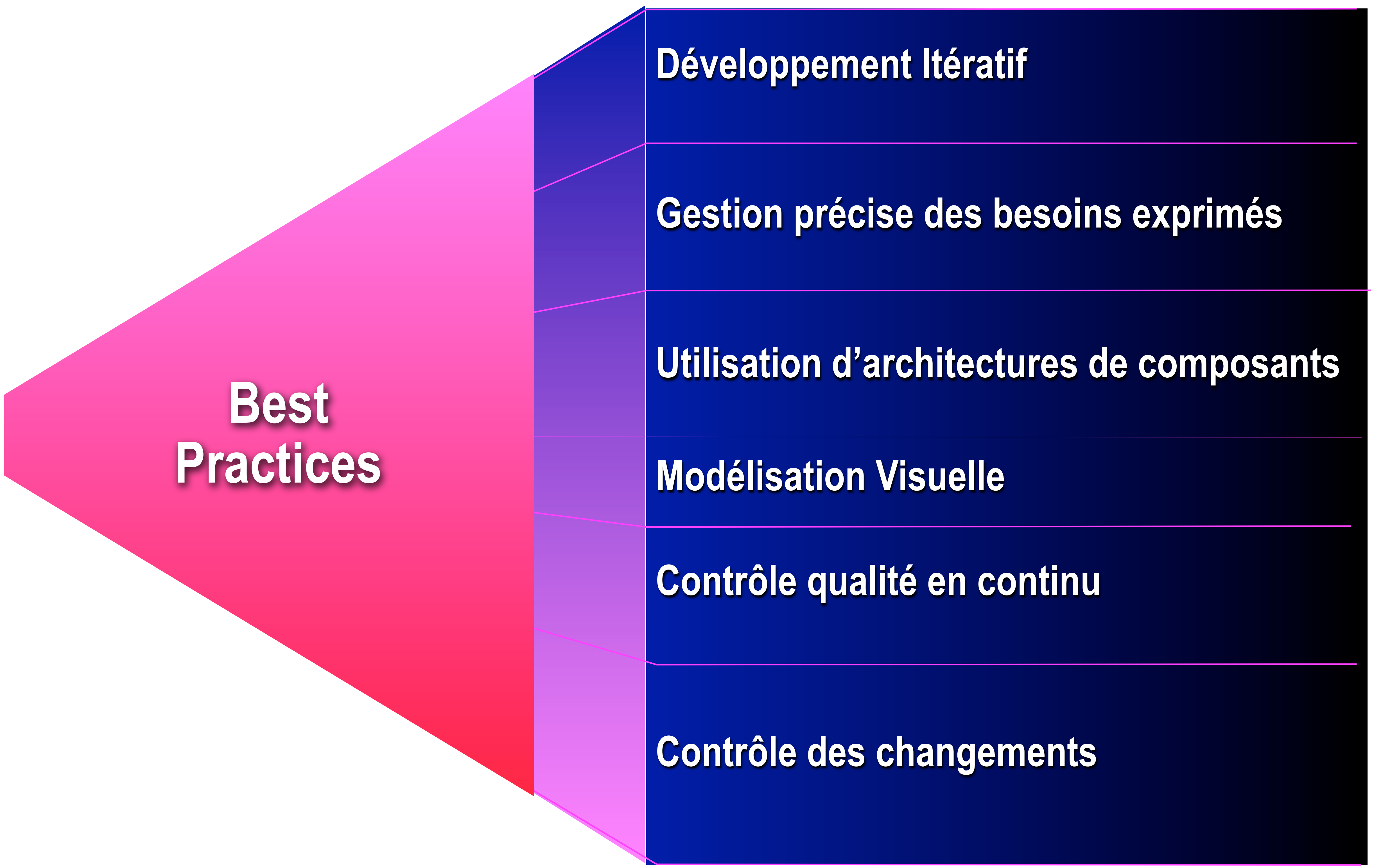
Exemple du RUP





66

Principes du *Rational Unified Process*



Points-clés

- Développer seulement ce qui est nécessaire
- Minimiser la paperasserie
- flexibilité
  - besoins, plan, utilisation des ressources, etc…
- Apprendre de ses erreurs précédentes
- Réévaluer les risques régulièrement
- Établir des critères de progrès
  - objectifs et mesurables
- Automatiser

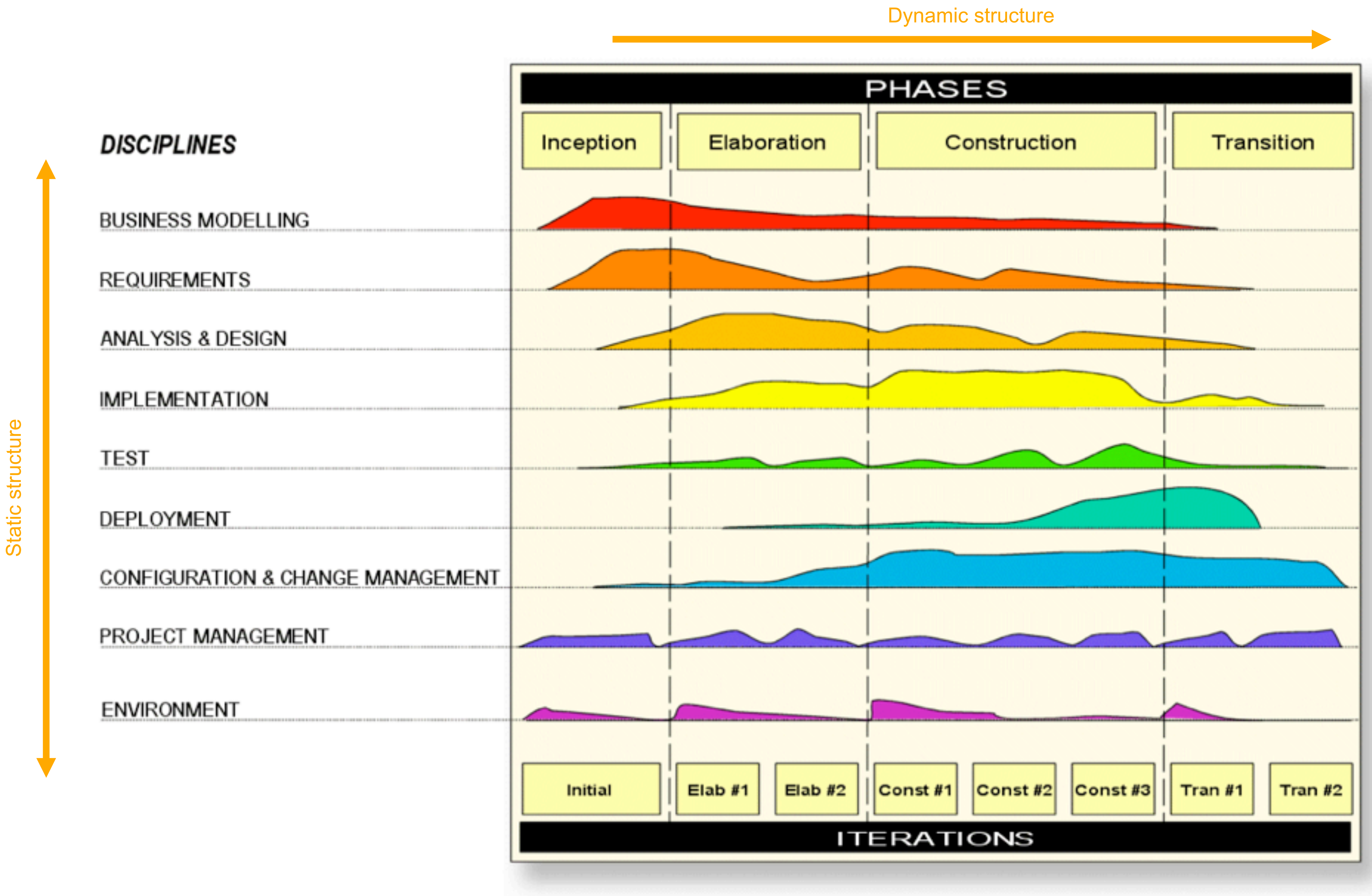
Architecture du processus

- 2 structures orthogonales
- Structure statique
  - Workers, artifacts, activities, workflows
  - authoring et configuration du processus

    ☞ SPEM, ingénierie des méthodes et des processus
- Structure dynamique
  - Structure du cycle de vie : phases, itérations
  - Mise en oeuvre du processus : planification, exécution

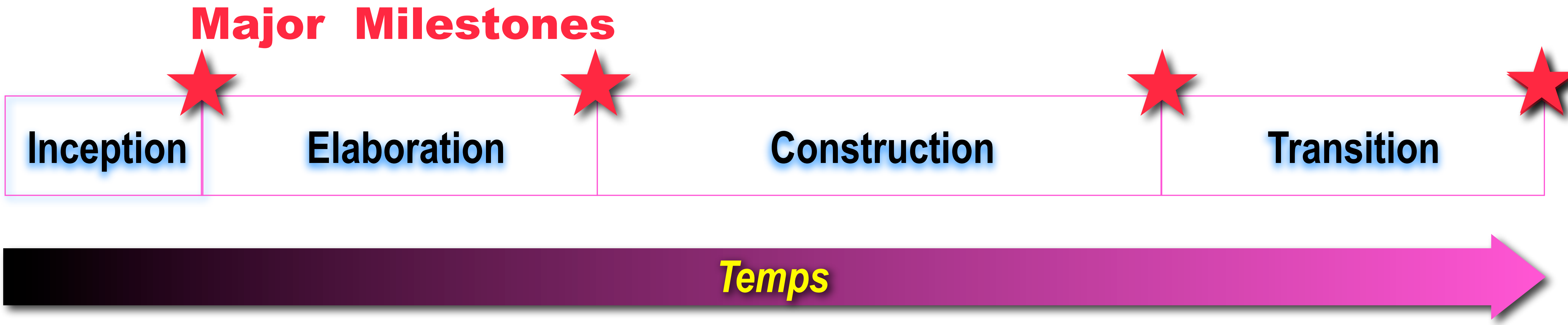
    ☞ gestion des activités, suivi de projet

Les 2 dimentions du processus



70

Phases du développement itératif



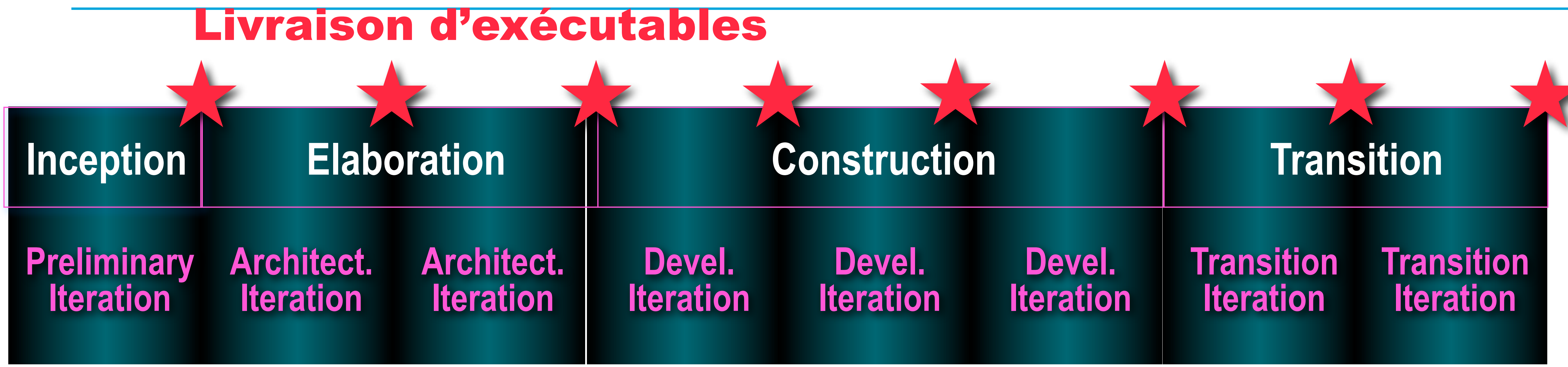
Inception: définition de la porté du projet

Elaboration: planification du projet, spécification des fonctionnalités, architecture de base

Construction: réalisation du produit

Transition: transfert du produit vers les utilisateurs

Itérations



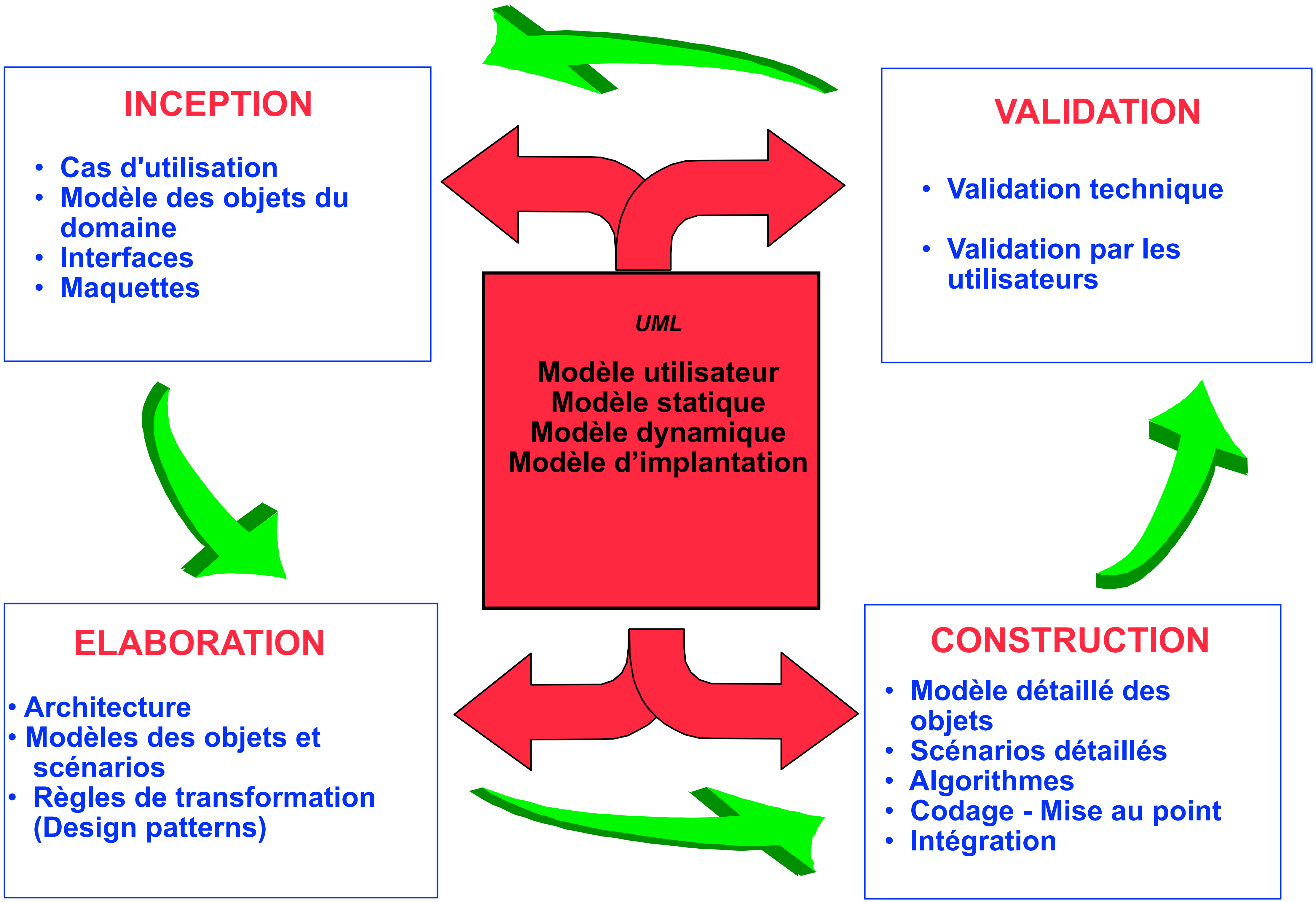
Une itération est une séquence d'activités avec un plan bien établi et un critère d'évaluation, résultant en la livraison d'un logiciel *exécutable*.

Phases et itérations : 2 exemples

- Petit projet de commerce électronique
  - Intégration à un mainframe
  - 5 personnes
- Grand projet d'infrastructure
  - Gros travail d'architecture nécessaire
  - 20 personnes

	No. of Iterations				Project Length	Iteration Length
	Inception	Elaboration	Construction	Transition		
e-business	0.2	1	3	1	3-4 months	2-3 weeks
infrastructure	1	3	3	2	9-12 months	5-7 weeks

Vision «générique» d'un cycle UML

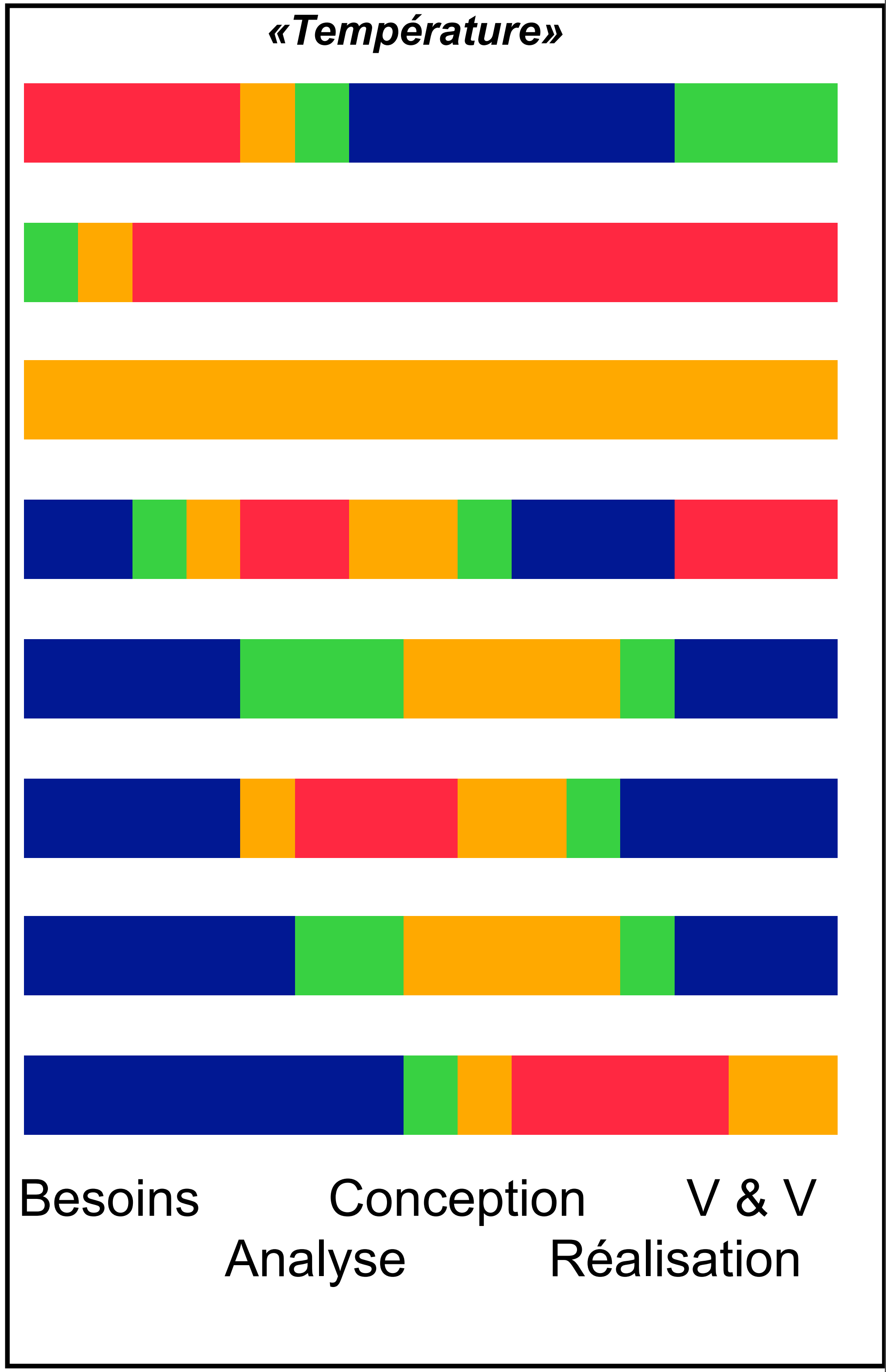


Processus de développement avec UML

- Approche itérative, incrémentale, dirigée par les cas d'utilisation
  - Expression des besoins
  - Analyse
    - Elaboration d 'un modèle « idéal »
  - Conception
    - passage du modèle idéal au monde réel
  - Réalisation et Validation

Température des diagrammes UML

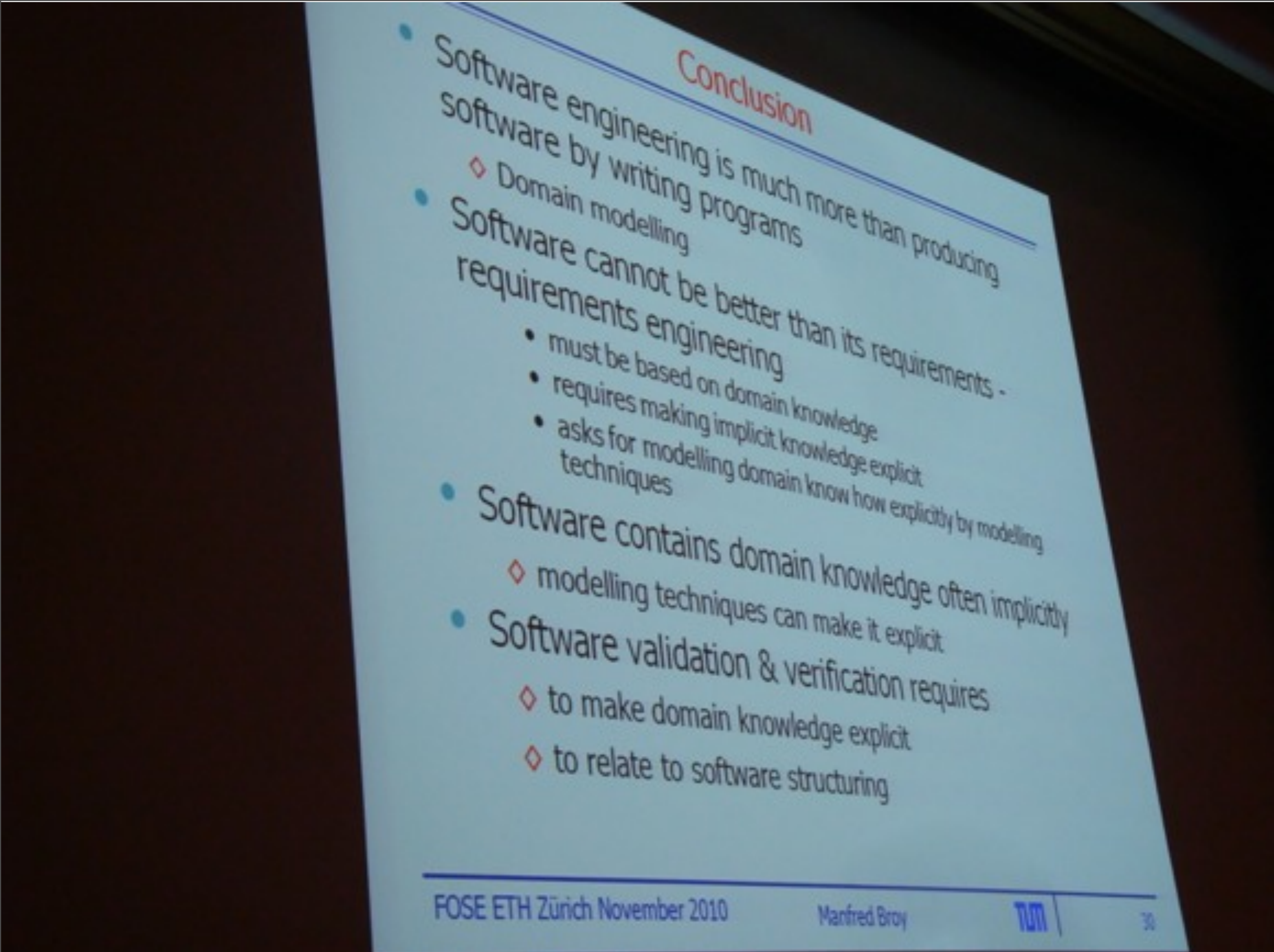
- Diagramme de cas d'utilisations
- Diagramme de classes
- Diagramme de paquetages
- Diagramme de séquences
- Diagramme de collaborations
- Diagramme d'états-transitions
- Diagramme d'activités
- Diagramme de déploiement



Conclusion

Conclusion

- Couplage fort du matériel et du logiciel
  - => ingénierie système
- Les systèmes (logiciels) deviennent de plus en plus prépondérants et complexes
  - *quelques axes de la complexité : temps-réel, distribué, critique, embarqué, pervasif, dynamique, (auto-)adaptable, …*
- Ingénierie du Logiciel
  - séparation des préoccupations
  - montée en abstraction
  - agilité des développements



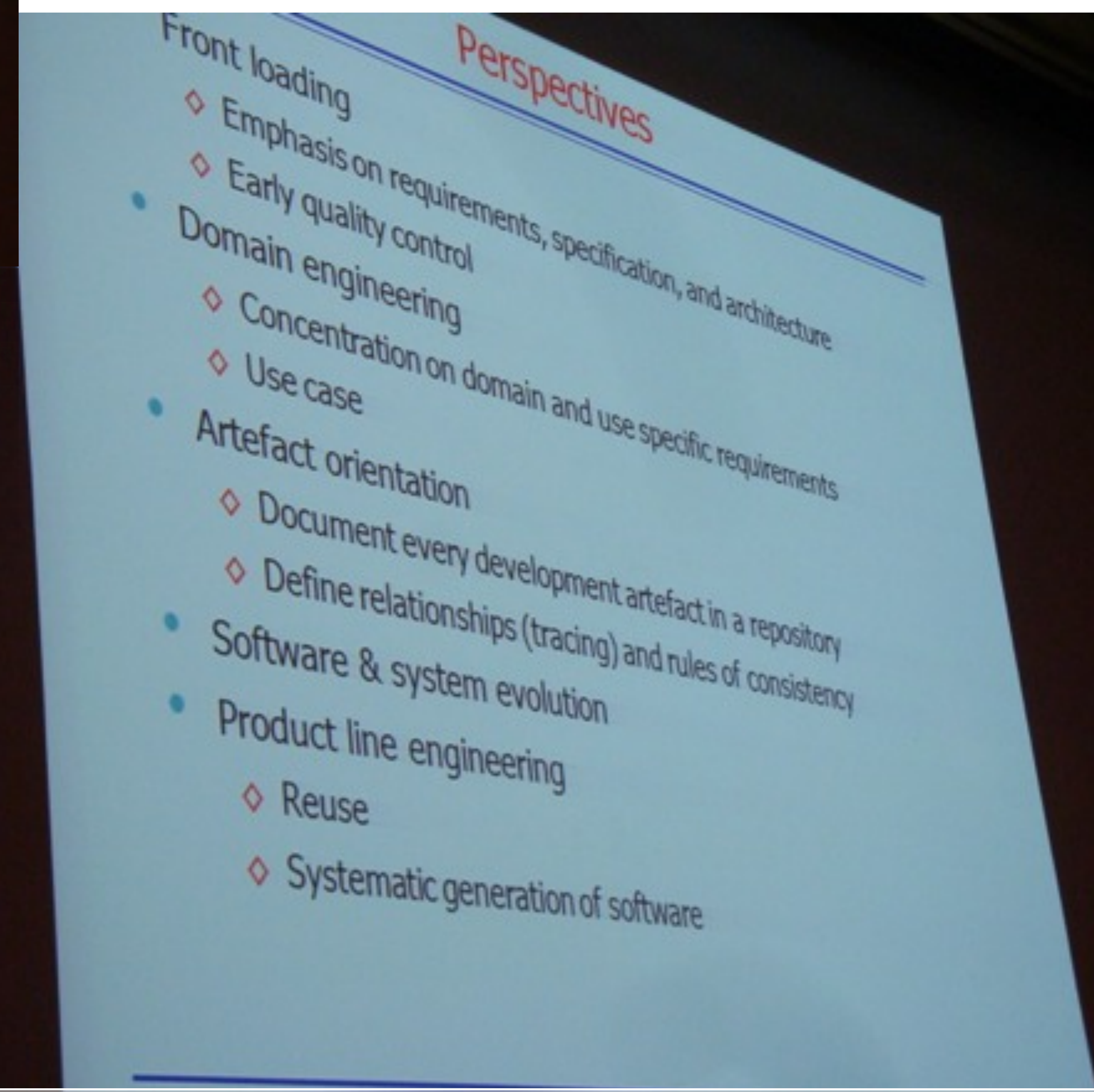
Seamless Method- and Model-based Software and Systems Engineering

The Future of Software Engineering Symposium

22-23 November 2010, ETH Zurich

Manfred Broy

http://en.wikipedia.org/wiki/Manfred_Broy



Seamless Method- and Model-based Software and Systems Engineering

Conclusion

- Software engineering is much more than producing software by writing programs
  - ◇ Domain modelling
- Software cannot be better than its requirements - requirements engineering
  - must be based on domain knowledge
  - requires making implicit knowledge explicit
  - asks for modelling domain know how explicitly by modelling techniques
- Software contains domain knowledge often implicitly
  - ◇ modelling techniques can make it explicit
- Software validation & verification requires
  - ◇ to make domain knowledge explicit
  - ◇ to relate to software structuring

from http://fose.ethz.ch/slides/broy.pdf

Seamless Method- and Model-based Software and Systems Engineering

The Future of Software Engineering Symposium
22-23 November 2010, ETH Zurich

Perspectives

- Front loading
  - Emphasis on requirements, specification, and architecture
  - Early quality control
- Domain engineering
  - Concentration on domain and use specific requirements
  - Use case
- Artefact orientation
  - Document every development artefact in a repository
  - Define relationships (tracing) and rules of consistency
- Software & system evolution
- Product line engineering
  - Reuse
  - Systematic generation of software

FOSE ETH Zürich November 2010 Manfred Broy TUM 3

from http://fose.ethz.ch/slides/broy.pdf