Modeling in Software Engineering

Benoit Combemale

PhD in Computer Science, Full Professor of Software Engineering

University of Rennes 1 (ESIR & IRISA)

benoit.combemale@irisa.fr - http://www.combemale.fr

Version: Sep. 2023

Objectifs du cours

- appréhender la complexité des systèmes modernes
- comprendre les enjeux du Génie Logiciel (système ?)
- 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

Outline

① Issues in Software Engineering

② Evolution in Software Engineering

③ State of the Practice

④ Modeling in Software Engineering

Outline

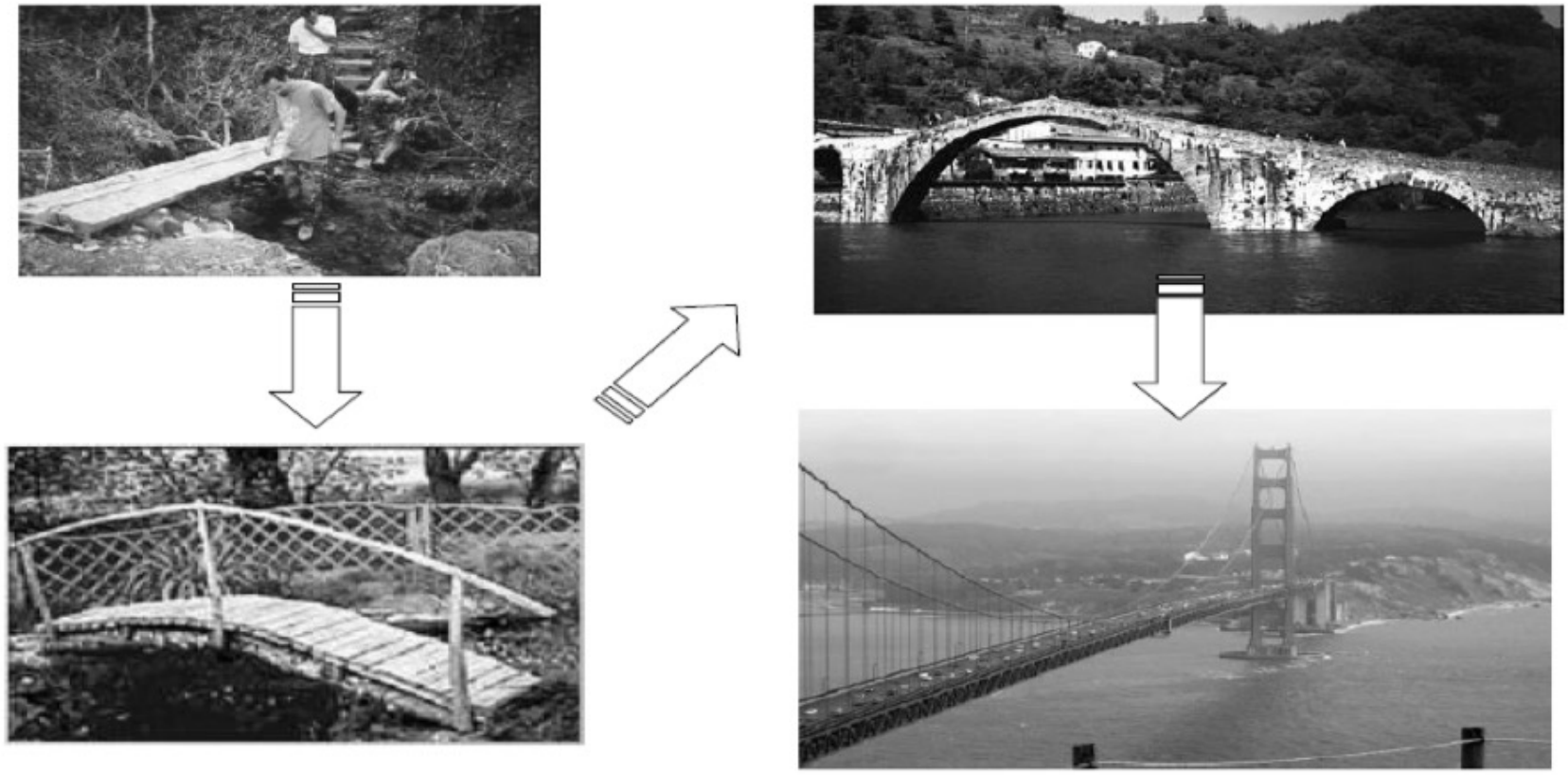
① Issues in Software Engineering

② Evolution in Software Engineering

③ State of the Practice

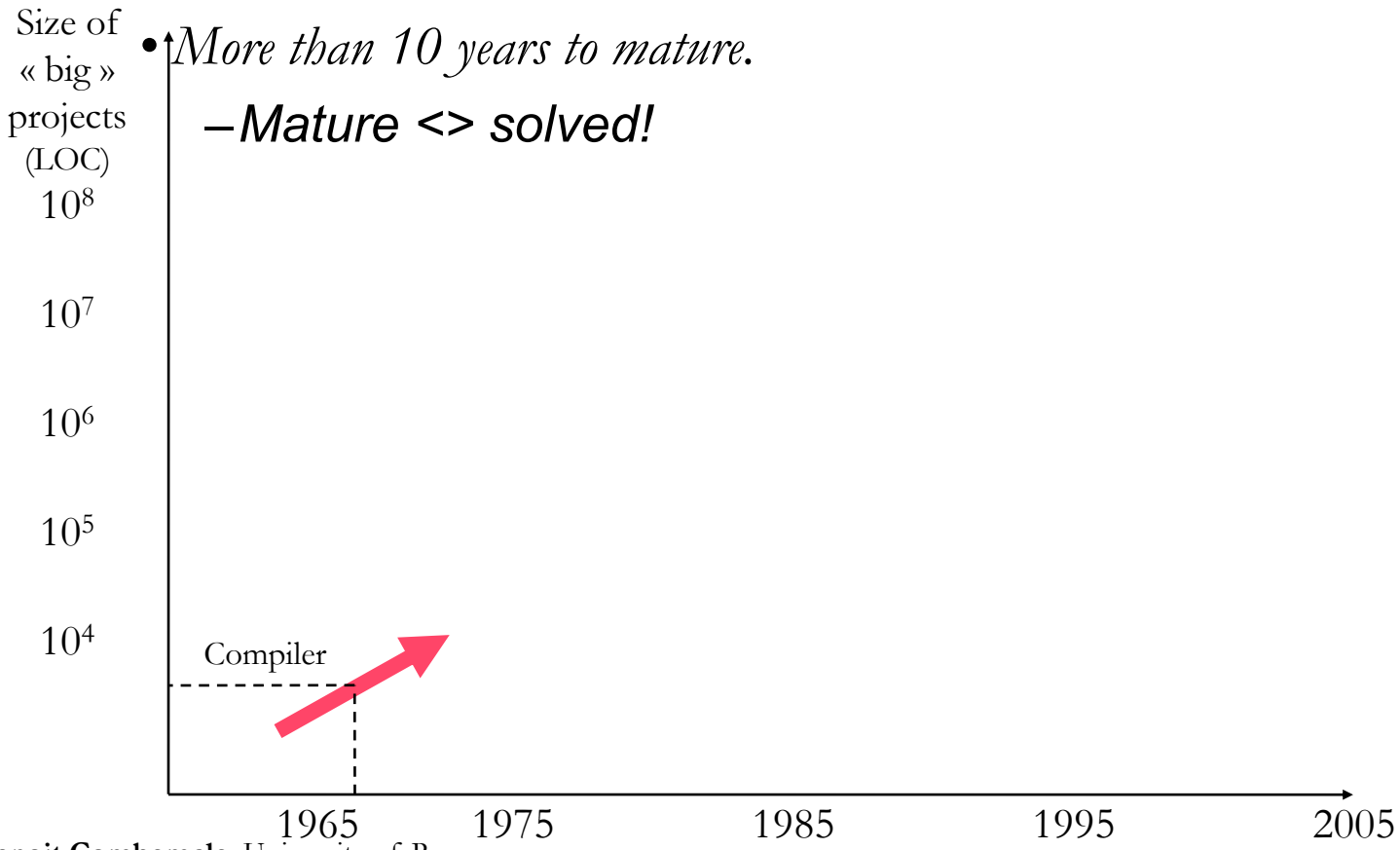
④ Modeling in Software Engineering

Software Complexity



Problems addressed in SE

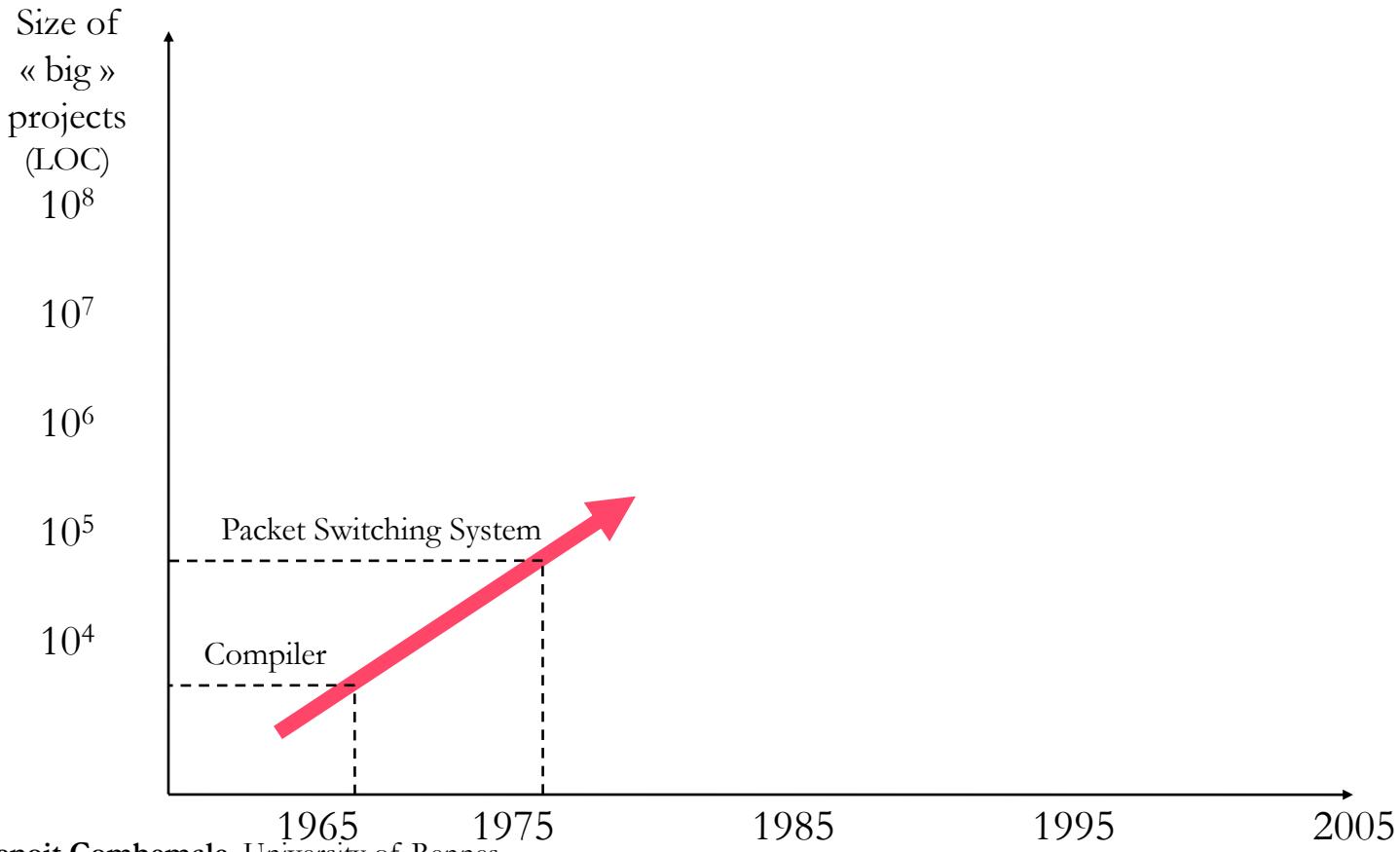
- 1960's: Cope with inherent complexity of software (Correctness)
  - Milestone: Floyd '*assigning meaning to programs*'



- *More than 10 years to mature.*
  - *Mature <> solved!*

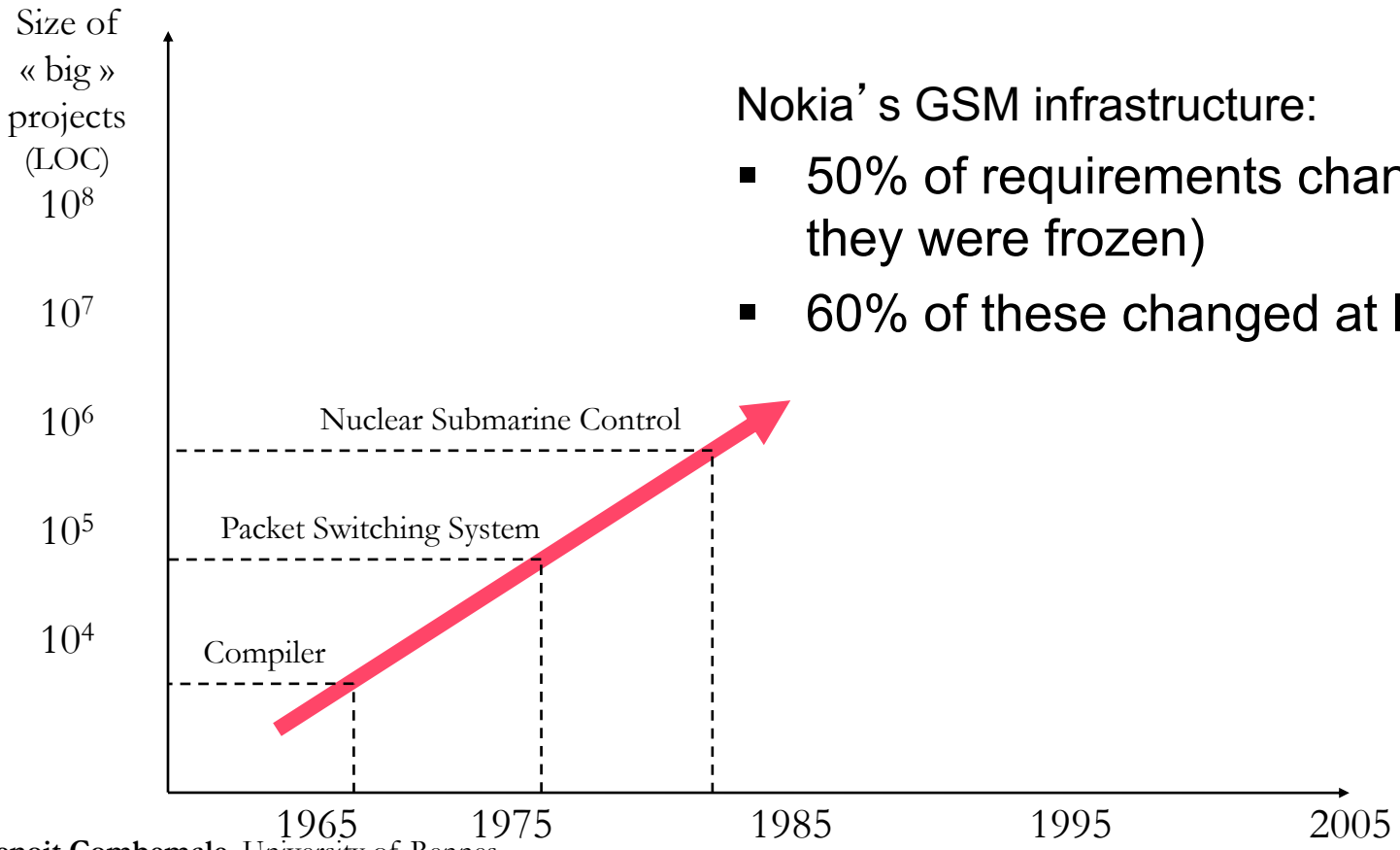
Problems addressed in SE

- 1970's: Cope with project size
  - Milestone: Parnas, Yourdon: *modularity & structure*
    - *More than 10 years to mature*



Problems addressed in SE

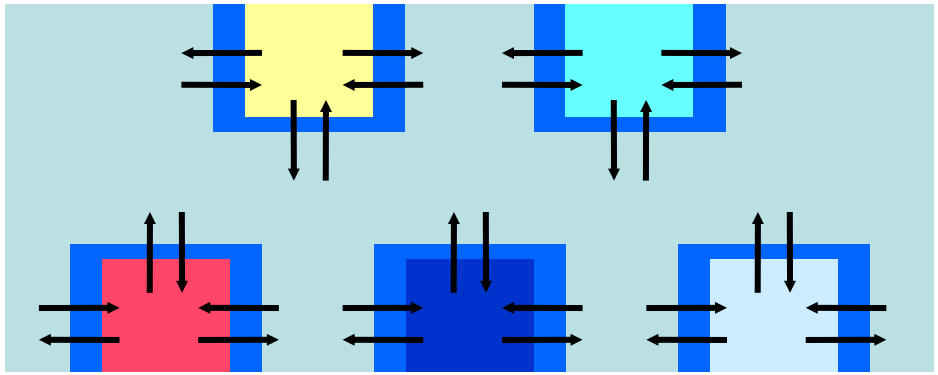
- 1980's: Cope with variability in requirements
  - Milestone: Jackson, Meyer: *modeling, object orientation*
    - *More than 10 years to mature*



Nokia's GSM infrastructure:

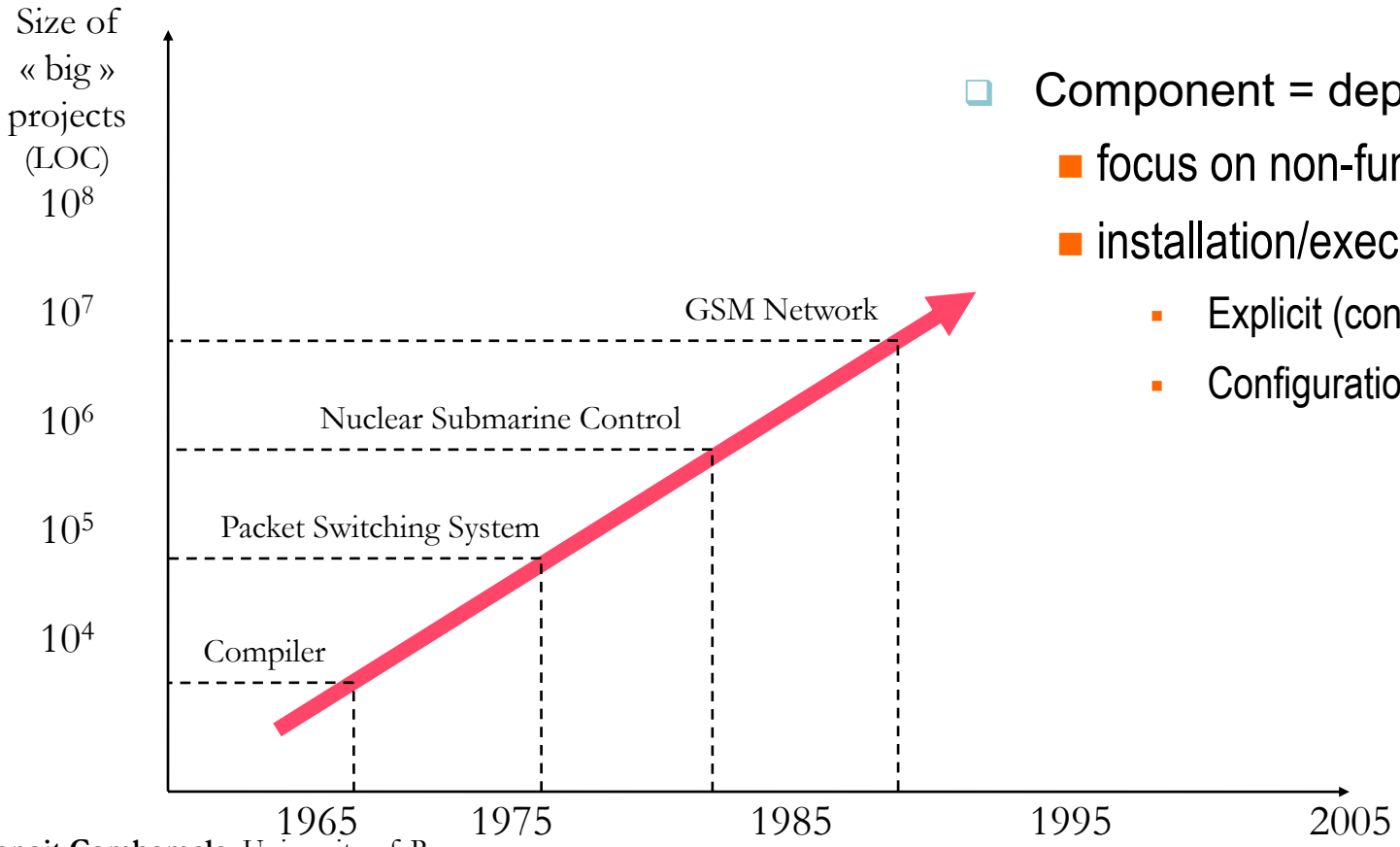
- 50% of requirements changed (after they were frozen)
- 60% of these changed at least twice!

OO approach: frameworks



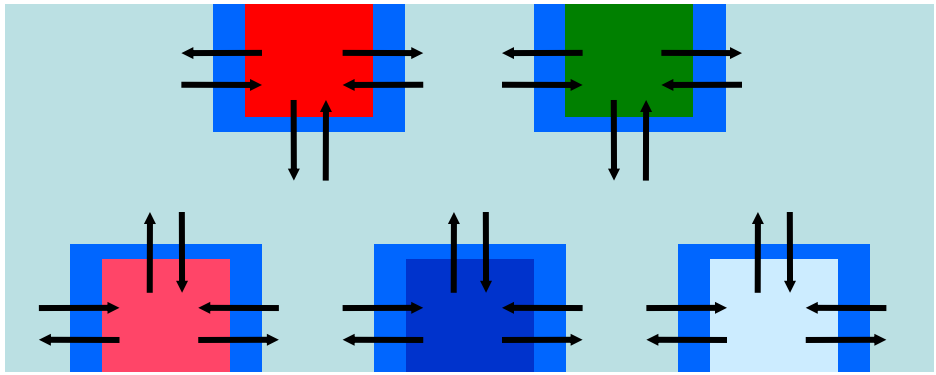
Problems addressed in SE

- 1990's: Cope with distributed systems and mass deployment:
  - Milestone: MS (COM), Szyperski: *product-lines & components*



- Component = deployment unit
  - focus on non-functional properties
  - installation/execution concept
    - Explicit (contractual) dependencies
    - Configuration and connection

OO approach: Models and Components

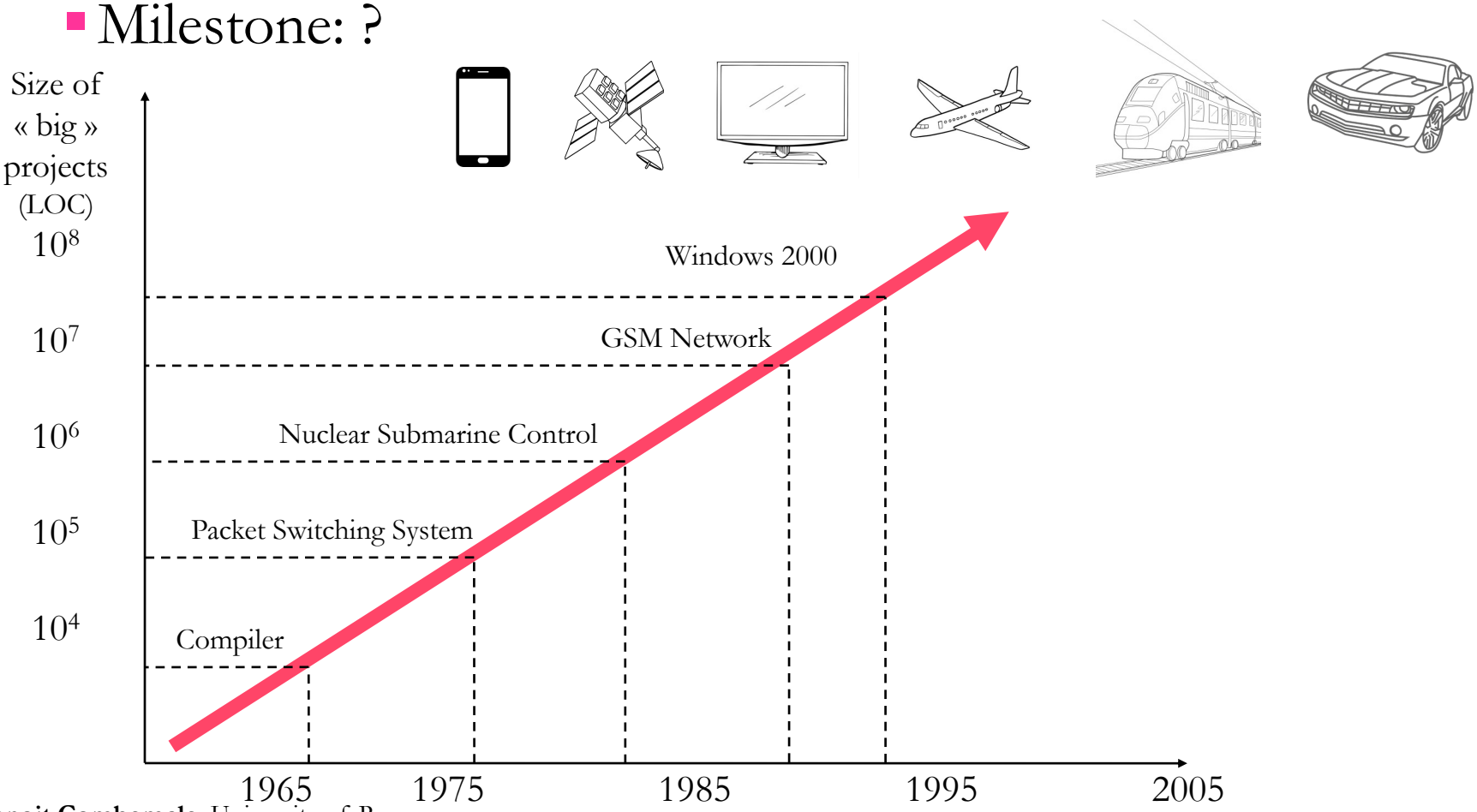


- Frameworks
  - Changeable software, from distributed/unconnected sources even after delivery, by the end user
  - Guarantees ?

    Functional , synchronization, performance, QoS

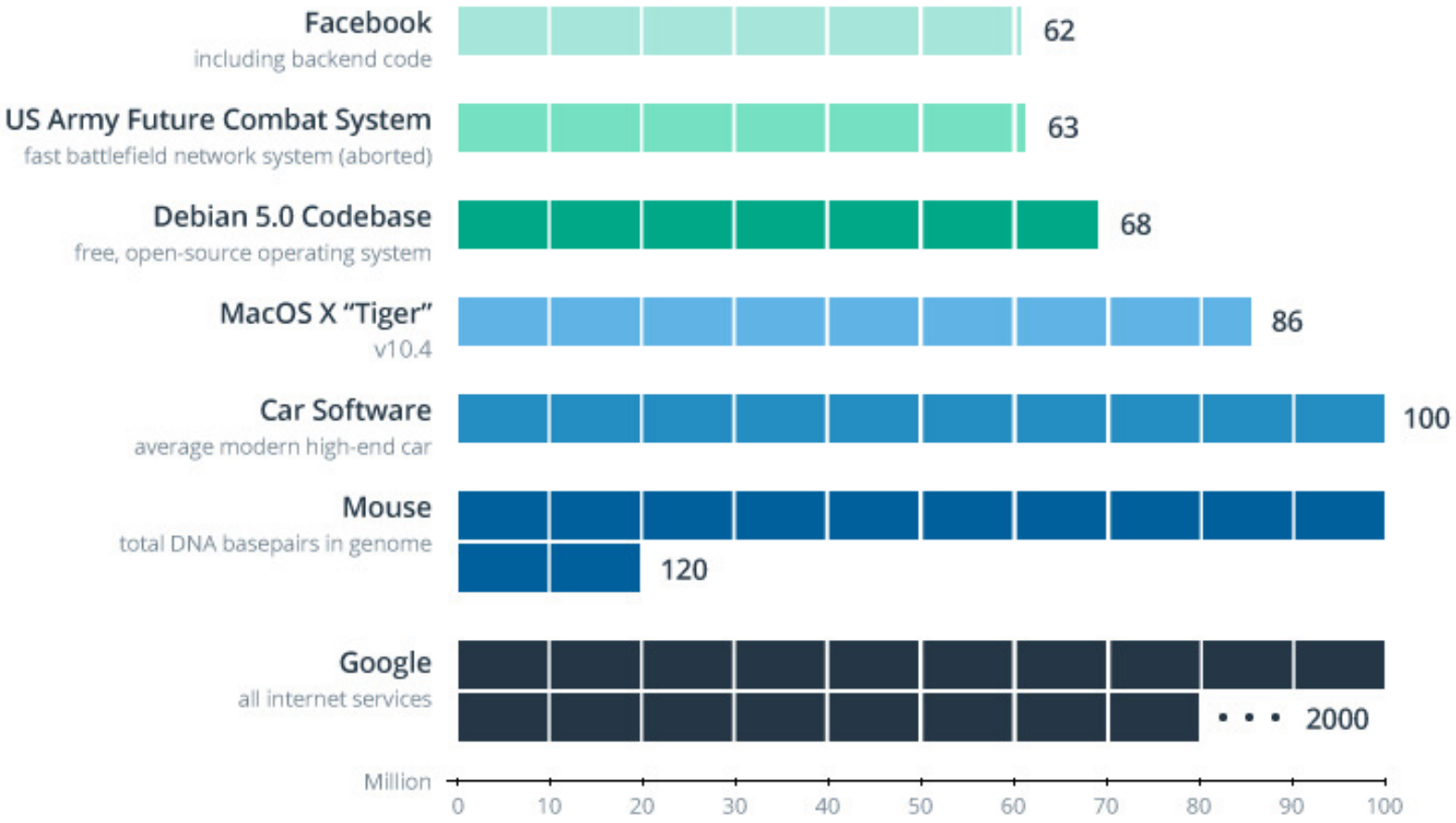
Problems addressed in SE

- 2000's: pervasive software integration, accelerating technological changes (platforms)
  - Milestone: ?



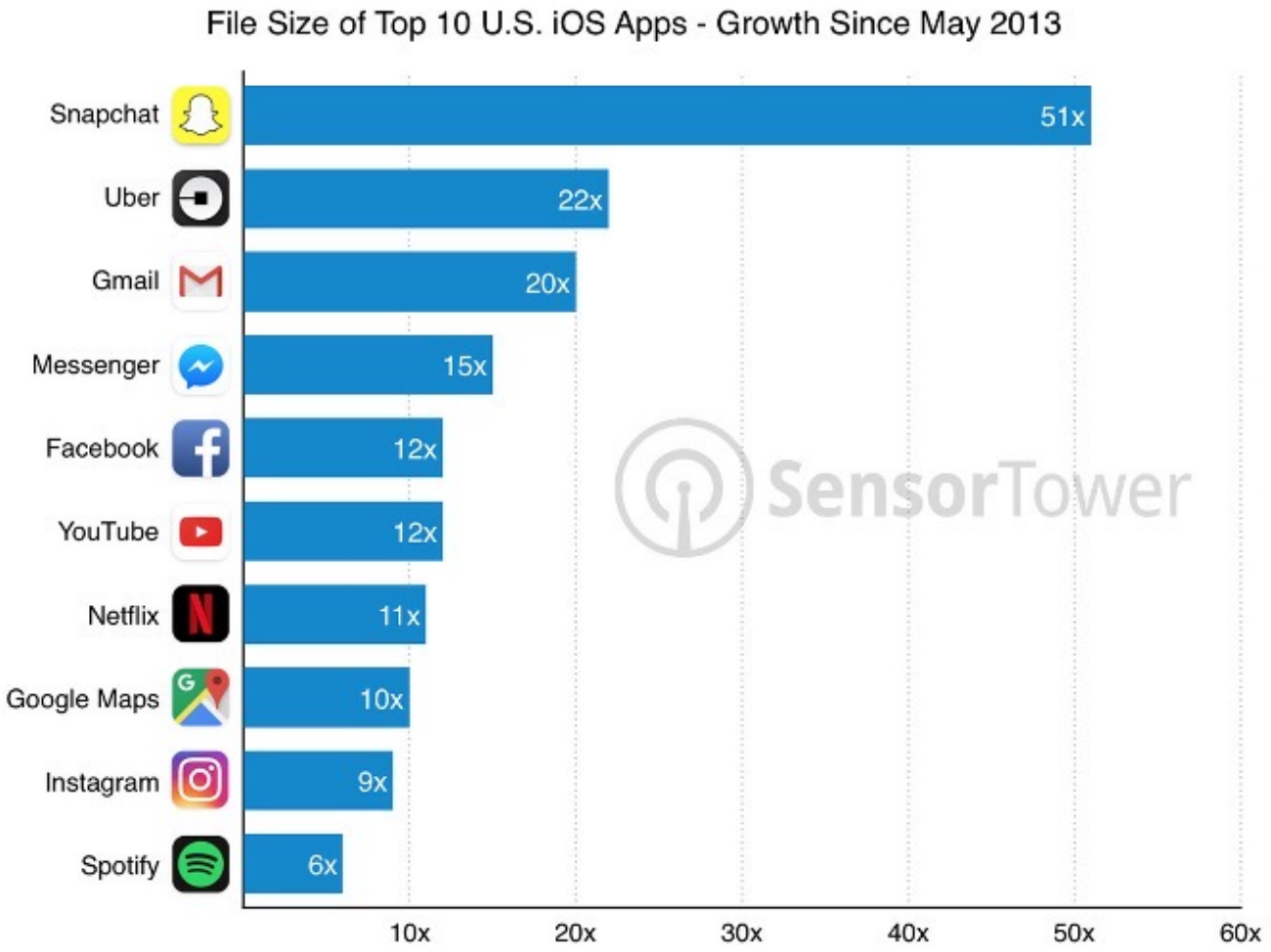

CHANGE
AHEAD

Software Complexity



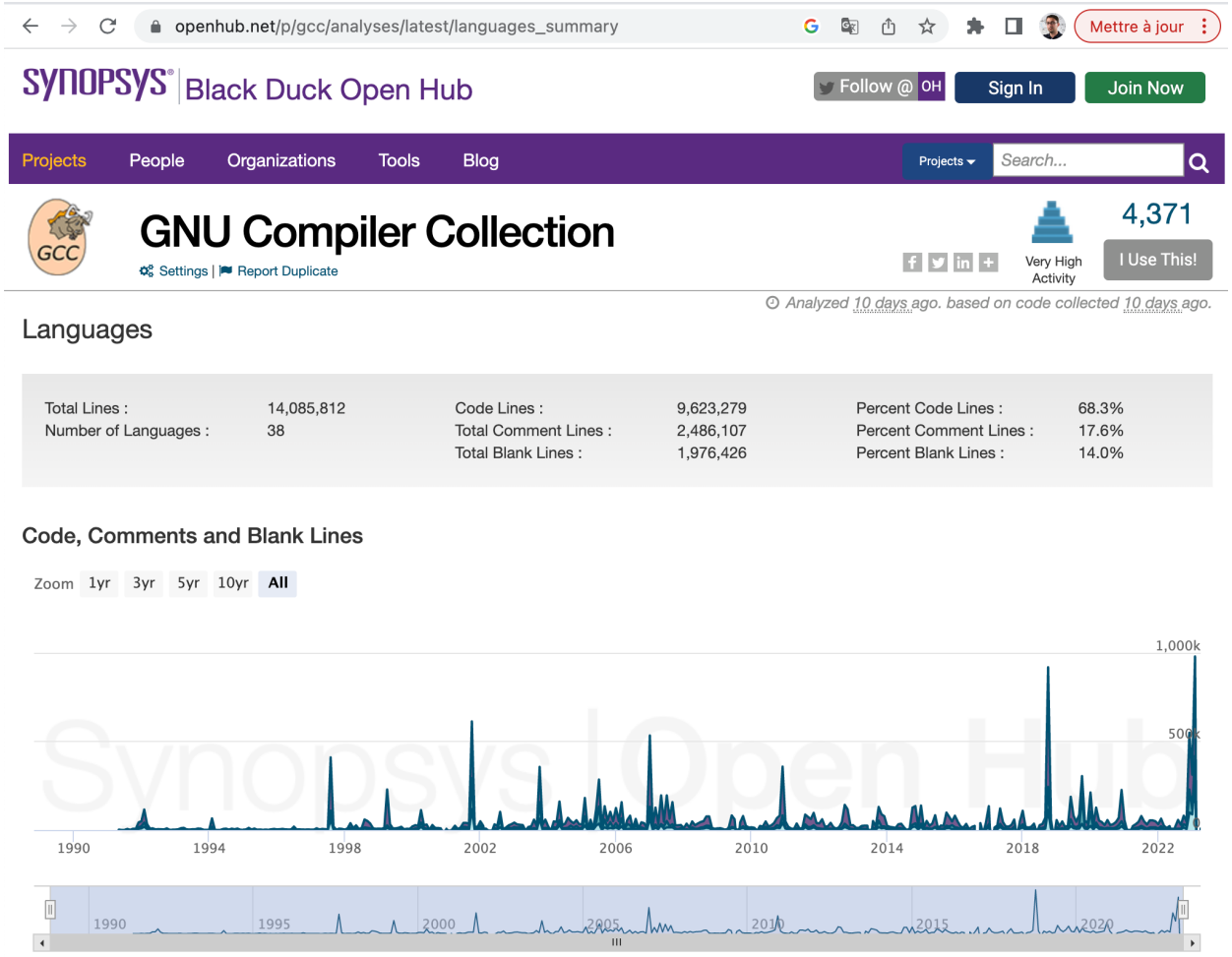
See https://informationisbeautiful.net/visualizations/million-lines-of-code/

Software Complexity



See https://www.programmez.com/dossier/page/ecoconception/reduire-la-taille-des-packages-et-mises-jour. Retrieved 2023-08-31.

Software Complexity



See https://www.openhub.net/p/gcc. Retrieved 2023-05-01.

Software Complexity

But also…

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C++	3,460,458	869,918	20.1%	739,205	5,069,581	36.0%
C	2,389,131	684,958	22.3%	475,972	3,550,061	25.2%
Ada	831,230	396,568	32.3%	290,062	1,517,860	10.8%
Go	793,870	169,011	17.6%	101,033	1,063,914	7.6%
D	675,619	157,575	18.9%	116,462	949,656	6.7%
Autoconf	531,266	2,363	0.4%	82,745	616,374	4.4%
Fortran (Free-format)	239,471	71,199	22.9%	45,037	355,707	2.5%
Modula-2	151,218	77,965	34.0%	51,802	280,985	2.0%
HTML	128,638	778	0.6%	12,645	142,061	1.0%
Make	121,979	4,616	3.6%	14,574	141,169	1.0%
Assembly	77,779	20,729	21.0%	13,448	111,956	0.8%
XML	65,934	550	0.8%	6,331	72,815	0.5%
shell script	39,165	8,469	17.8%	5,849	53,483	0.4%
Objective-C	28,618	5,603	16.4%	8,369	42,590	0.3%
Fortran (Fixed-format)	23,571	2,080	8.1%	1,992	27,643	0.2%
Python	11,584	3,633	23.9%	2,798	18,015	0.1%
Rust	10,497	1,004	8.7%	2,317	13,818	0.1%
Automake	9,420	1,610	14.6%	1,724	12,754	0.1%
Postscript	8,674	157	1.8%	234	9,065	0.1%
TeX/LaTeX	8,668	3,611	29.4%	1,053	13,332	0.1%
Perl	6,122	1,840	23.1%	1,074	9,036	0.1%
AWK	4,833	785	14.0%	661	6,279	0.0%
Pascal	1,083	141	11.5%	219	1,443	0.0%
Mathematica	1,061	47	4.2%	242	1,350	0.0%
C#	879	506	36.5%	230	1,615	0.0%
DCL	756	162	17.6%	12	930	0.0%
JavaScript	490	21	4.1%	90	601	0.0%
CMake	310	44	12.4%	59	413	0.0%
OCaml	285	29	9.2%	44	358	0.0%
Haskell	208	0	0.0%	27	235	0.0%
CSS	203	42	17.1%	48	293	0.0%
Vim Script	193	71	26.9%	48	312	0.0%
AMPL	34	0	0.0%	9	43	
Brainfuck	10	4	28.6%	3	17	
Emacs Lisp	7	12	63.2%	4	23	
XSL Transformation	6	6	50.0%	4	16	
Matlab	5	0	0.0%	0	5	0.0%
DOS batch script	4	0	0.0%	0	4	0.0%
Totals	9,623,279	2,486,107		1,976,426	14,085,812	

See https://www.openhub.net/p/gcc. Retrieved 2023-05-01.

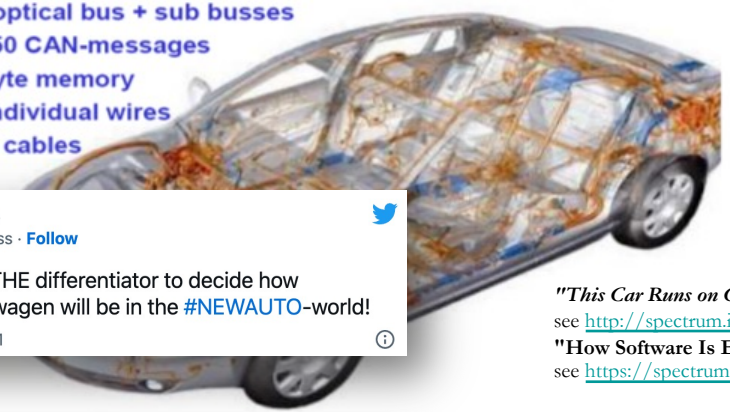
- Interoperability

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



Herbert Diess
@Herbert_Diess · Follow

Software will be THE differentiator to decide how successful Volkswagen will be in the #NEWAUTO-world!

9:12 AM · Dec 10, 2021

"This Car Runs on Code", By Robert N. Charrette, IEEE Spectrum, Feb. 2009, see http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code

"How Software Is Eating the Car", By Robert N. Charrette, IEEE Spectrum, Jun. 2021, see https://spectrum.ieee.org/software-eating-car

Software Complexity

- *"The avionics system in the F-22 Raptor […] consists of about 1.7 million lines of software code."*
- *"F-35 Joint Strike Fighter […] will require about 5.7 million lines of code to operate its onboard systems."*
- *"Boeing's new 787 Dreamliner […] requires about 6.5 million lines of software code to operate its avionics and onboard support systems."*
- *"if you bought a premium-class automobile recently, it probably contains close to 100 million lines of software code. […] All that software executes on 70 to 100 microprocessor-based electronic control units (ECUs) networked throughout the body of your car."*
- *"Alfred Katzenbach, the director of information technology management at Daimler, has reportedly said that the radio and navigation system in the current S-class Mercedes-Benz requires over 20 million lines of code alone and that the car contains nearly as many ECUs as the new Airbus A380 (excluding the plane's in-flight entertainment system)."*
- *"IBM claims that approximately 50 percent of car warranty costs are now related to electronics and their embedded software"*

"This Car Runs on Code", By Robert N. Charrette, IEEE Spectrum, Feb. 2009, see http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code

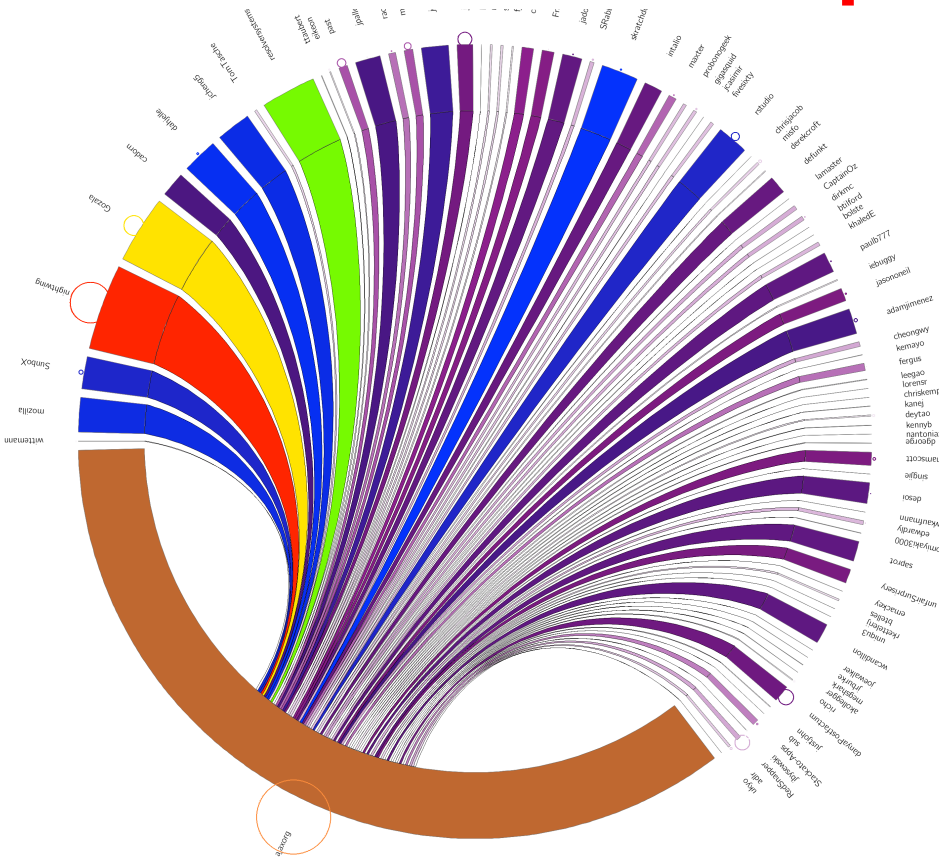
Software Complexity

- ***"Once, software was a part of the car. Now, software determines the value of a car,"* notes Manfred Broy**
- ***"high-end cars like the BMW 7-series with advanced technology like advanced driver-assist systems (ADAS) may contain 150 ECUs or more, while pick-up trucks like Ford's F-150 top 150 million lines of code. Even low-end vehicles are quickly approaching 100 ECUs and 100 million of lines of code as more features that were once considered luxury options, such as adaptive cruise control and automatic emergency braking, are becoming standard."***
- ***"as of 2017, some 40% of the cost of a new car can be attributed to semiconductor-based electronic systems, a cost doubling since 2007."***
- ***"Volvo has a superset of about 120 ECUs from which it selects to create a system architecture present within every Volvo vehicle. Altogether, they comprise a total of 100 million lines of source code." This source code "contains 10 million conditional statements as well as 3 million functions, which are invoked some 30 million places in the source code." Notes Vard Antinyan, a software quality expert at Volvo Cars.***

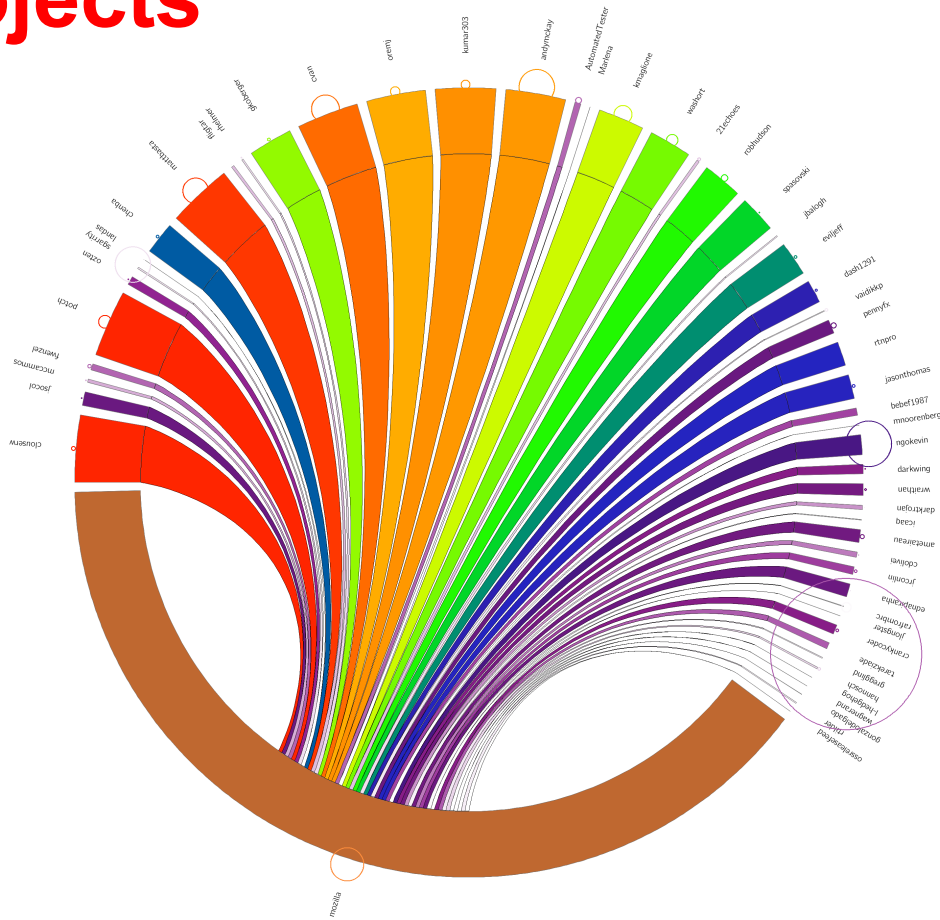
"How Software Is Eating the Car", By Robert N. Charrette, IEEE Spectrum, Jun. 2021, see https://spectrum.ieee.org/software-eating-car

Software Complexity

- Collaborative projects



137 contributors, 5000 commits, 1300 forks
https://github.com/ajaxorg/ace



97 contributors, 10000+ commits, 173 forks
https://github.com/mozilla/zamboni

Software Complexity

Long term availability…

AIRBUS A300 Life Cycle

Program began in 1972, production stopped in 2007

2007-1972 = 35 years...

Support will last until 2050

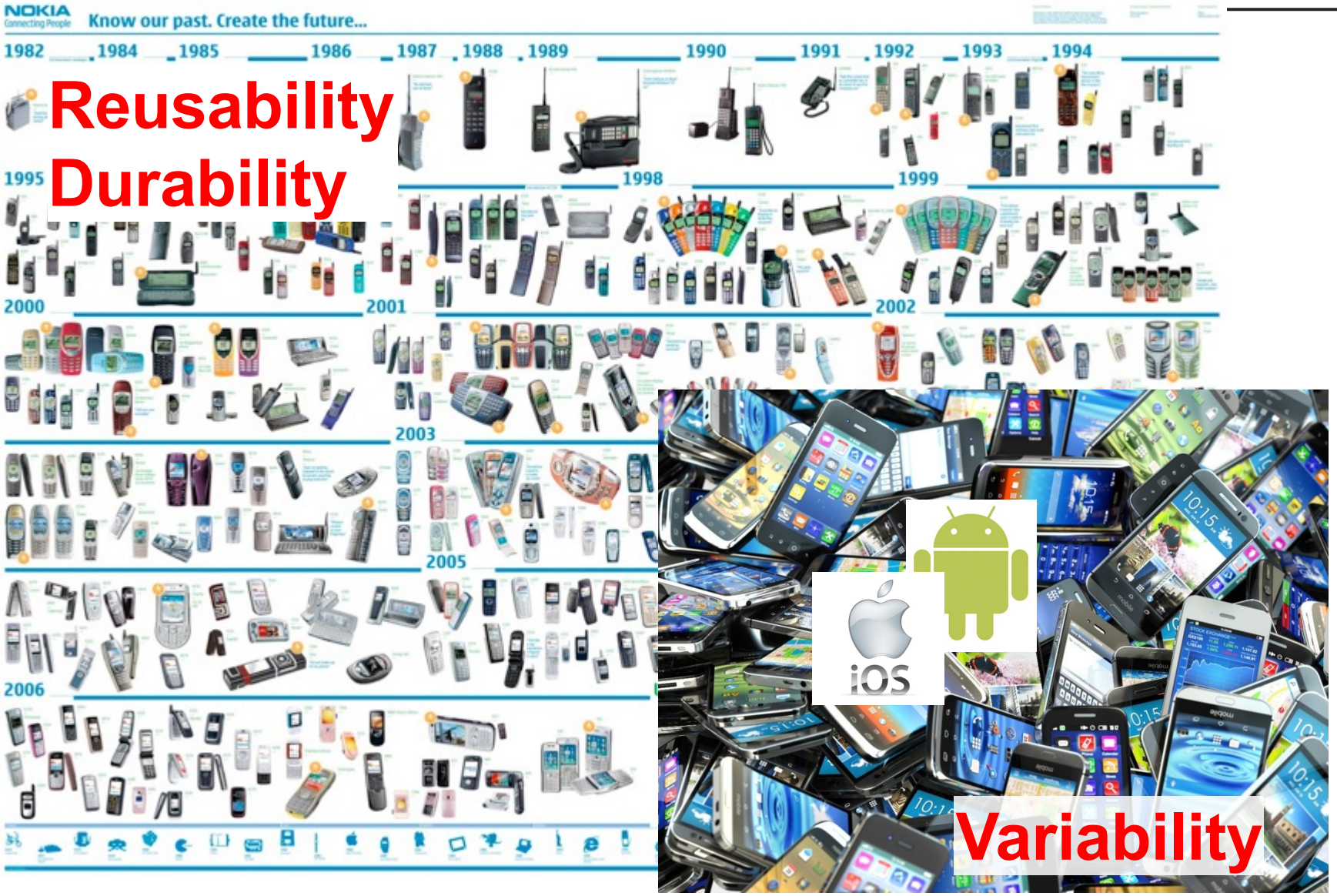
2050-1972 = 78 years !!

On board software development for very long lifecycle products

From the OPEES ITEA2 project (2009-2012)

AIRBUS

Software Complexity



Software Complexity

Distributed
Large-scale

- **Google (2012 Update from Larry Page, CEO):**
  - *Over 850,000 Android devices are activated daily through a network of 55 manufacturers and more than 300 carriers.*
  - *Google Chrome browser has over 200 million users.*
  - *Google launched Gmail in 2004 and now is used by more than 350 million people.*
  - *YouTube has over 800 million monthly users who upload an hour of video per second.*

See http://investor.google.com/corporate/2012/ceo-letter.html

Software Complexity

- Google, 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

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

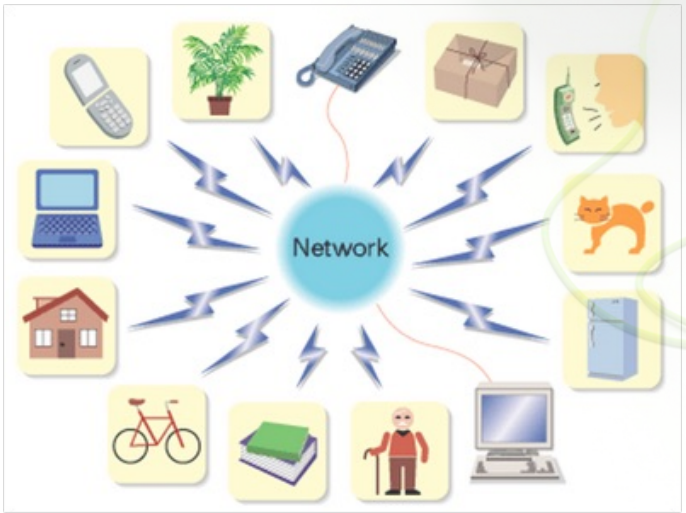
Software Complexity

- **Free Mobile (10/01/2012):**
  - *"A 9h45, le site mobile.free.fr cumulait déjà plus d'un million d'accès simultanés alors que le site proposait seulement une page invitant à patienter" !!*

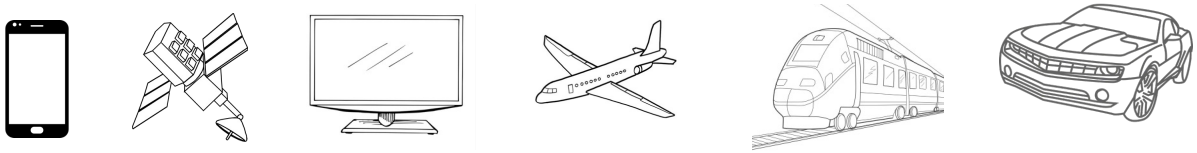
- **OVH outage (13/10/2021):**
  - *"Suite à une erreur humaine durant la reconfiguration du network sur notre DC à VH (US-EST), nous avons un souci sur la toute la backbone. Nous allons isoler le DC VH puis fixer la conf."*

Software Complexity

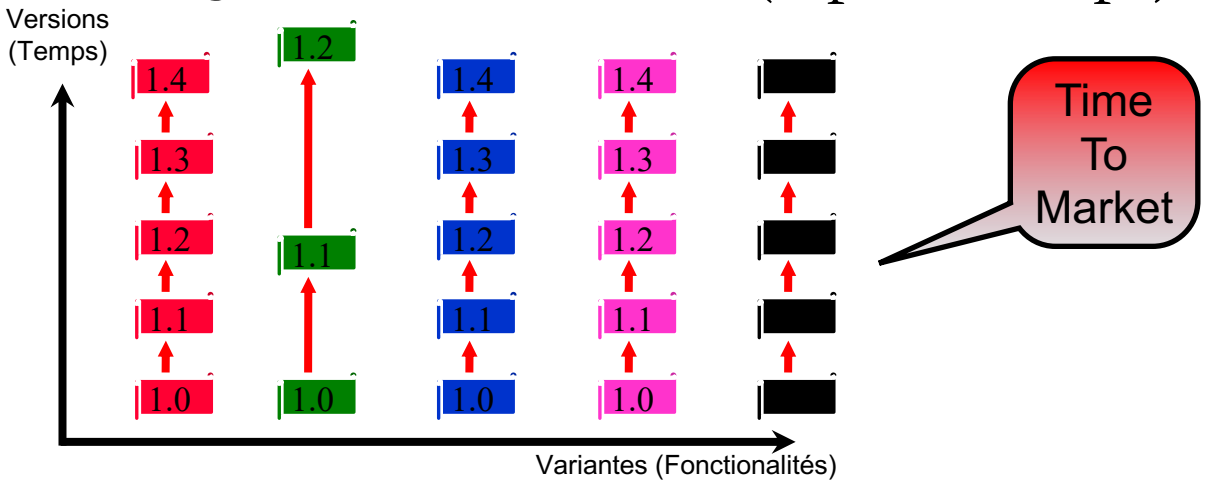
- Autonomic Computing
- Cloud Computing
- PaaS, SaaS, IoS, IoT...



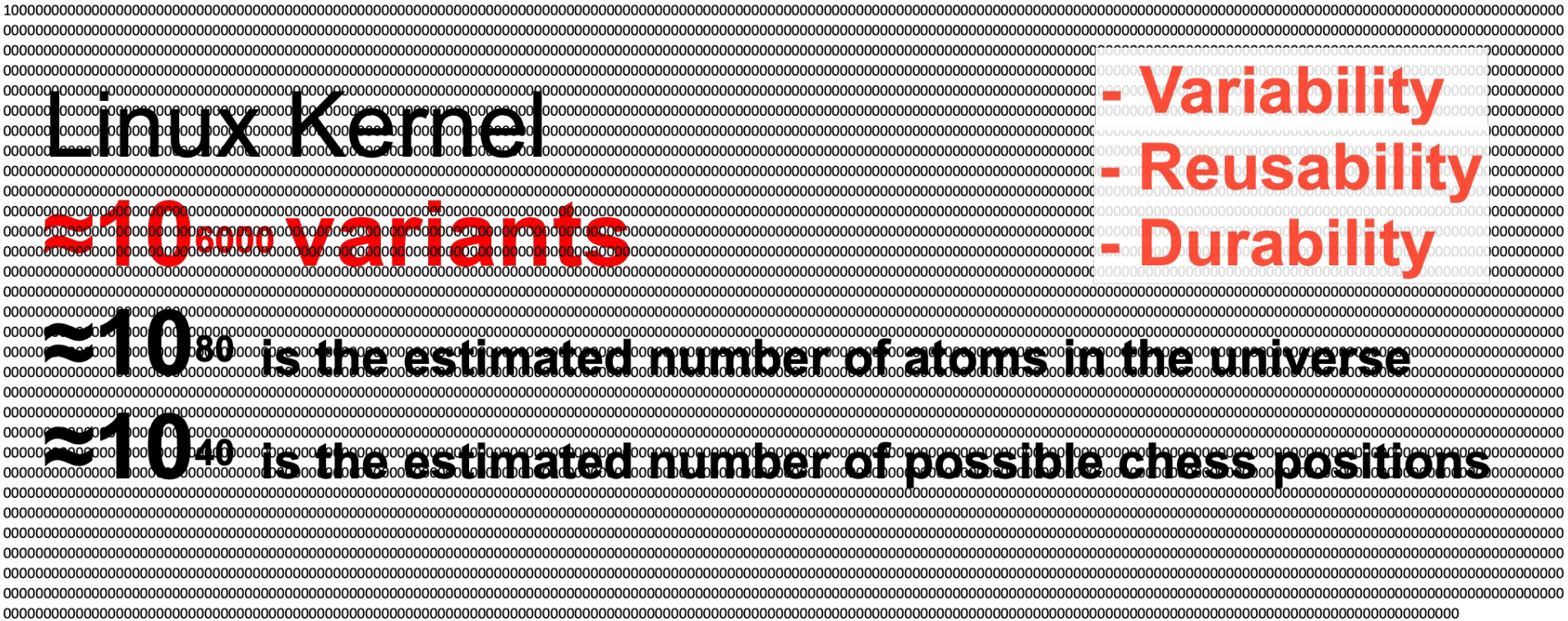
Software Complexity



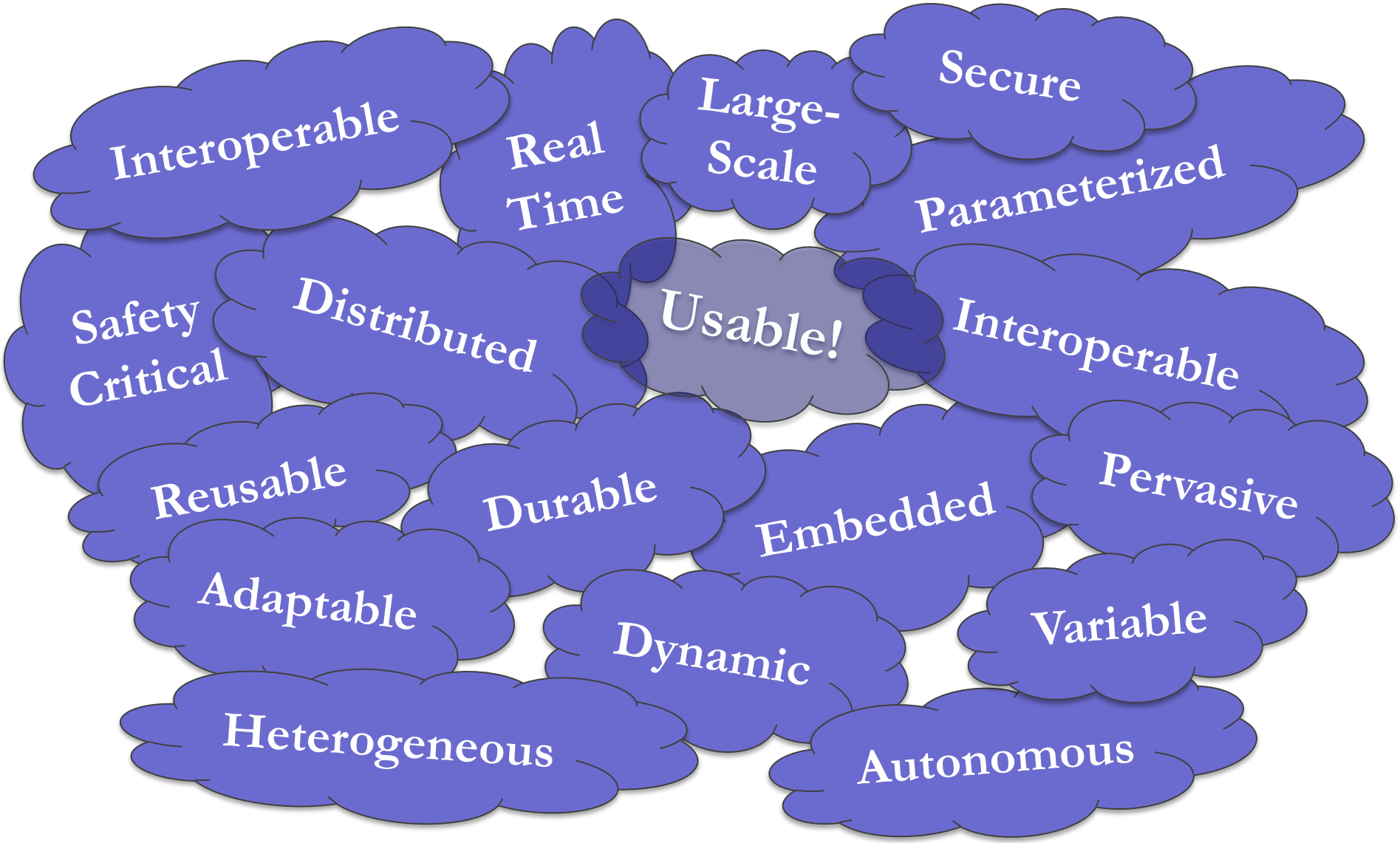
- Importance des aspects non fonctionnels
  - systèmes répartis, parallèles et asynchrones
  - qualité de service : fiabilité, latence, performances...
- Flexibilité accrue des aspects fonctionnels
  - notion de **lignes de produits** (espace, temps)



Software Complexity



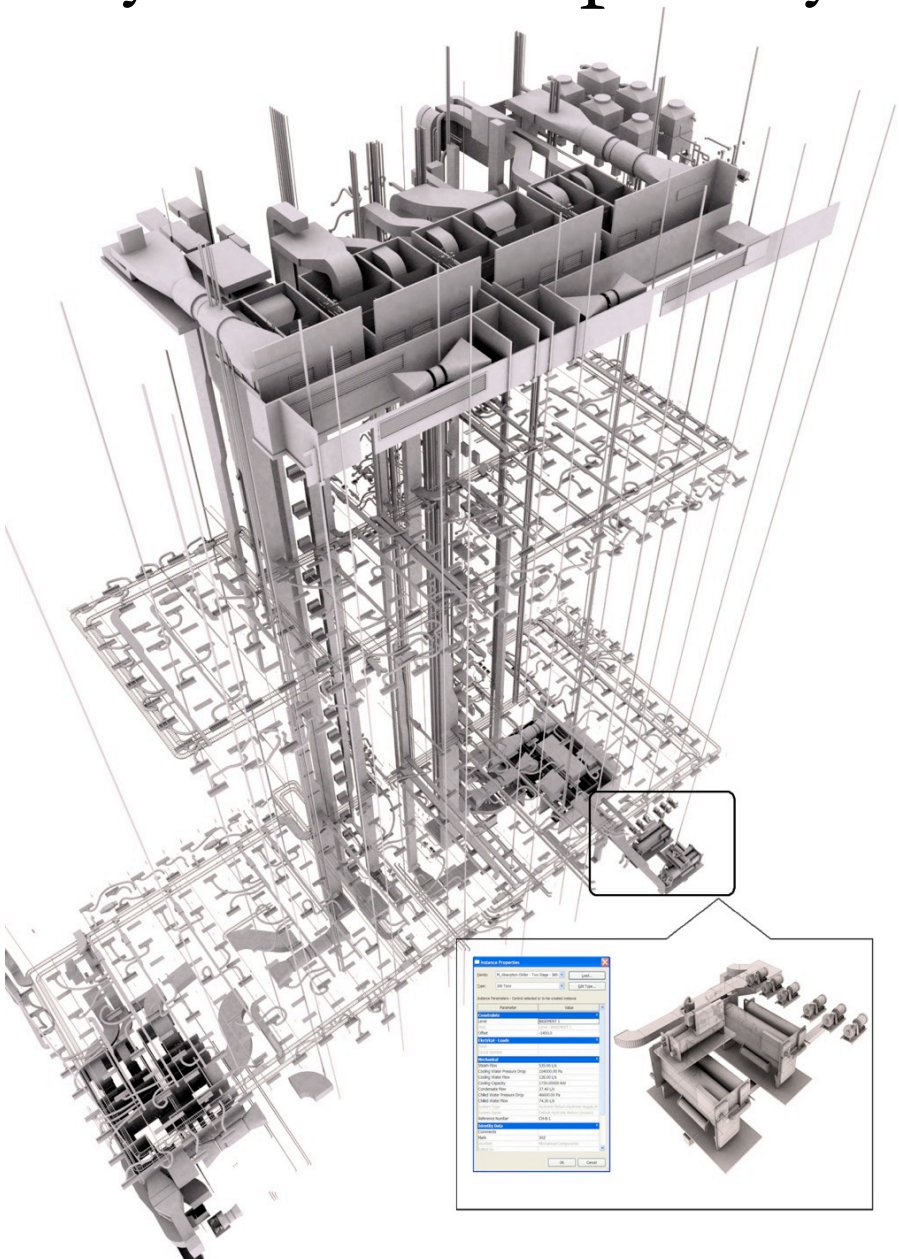
Software Complexity: Some Dimensions

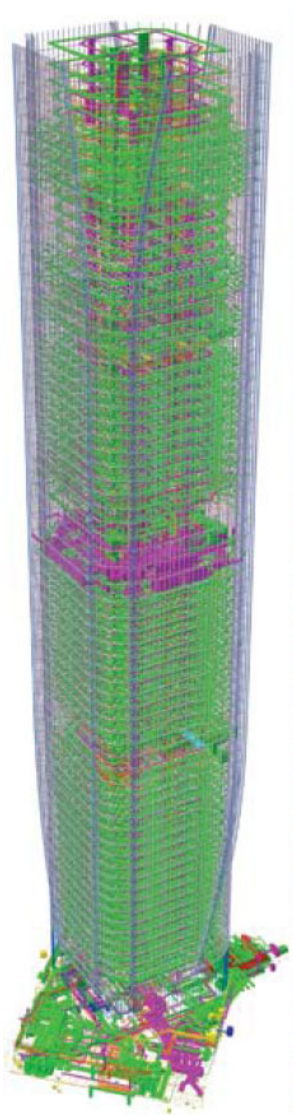


Défaillances

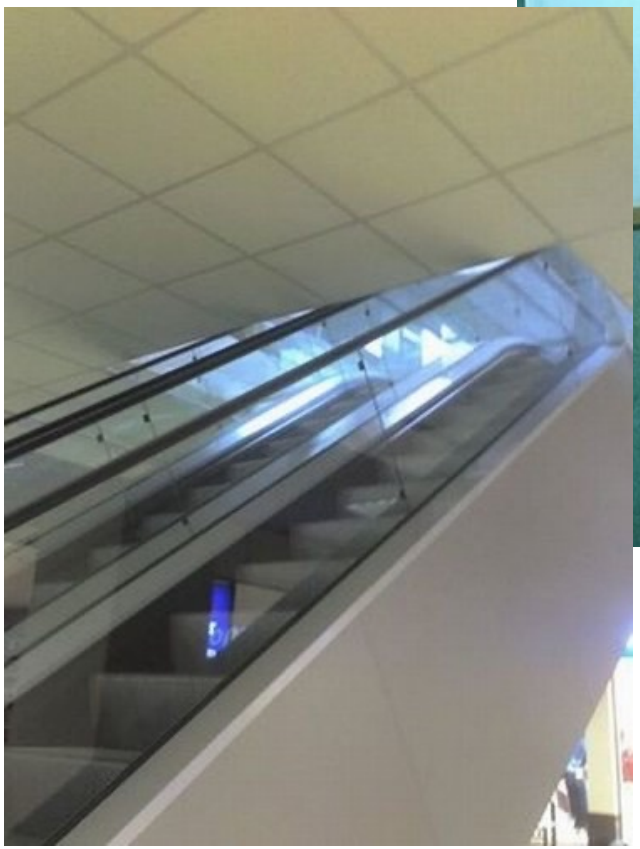
- Catastrophe humaine ou financière:
  - Therac-25 (1985-1987) – radiologie et contrôle d'injection de substances radioactives
  - Iran Air Flight 655 (1988) – guerre d'Irak et missile américain – système radar
  - London Ambulance System (1992) – central et dispatch ambulances
  - Ariane 5 (1996)
  - Mars Climate Orbiter (1999) – sonde spatiale – unité de mesure
  - Bourse de Londres (Taurus, 1993) – SI qui n'a pas pu être déployé
  - SI du FBI (2005) – SI qui n'a pas pu être déployé
  - Un canon-robot anti-aérien sud-africain tue neuf soldats dans un mode tout automatique qui avait été rajouté (2007)
- Image de marque :
  - FT et Bouygues en 2004 – crash des serveurs – indisponibilité 48h
  - Playstation 3 : le système de jeux en ligne croyait qu'il y avait un 29 février 2010, ce qui bloqua le fonctionnement des jeux en ligne toute la journée du 1 mars 2010
  - IPhone 4 : problème de réveil après le passage à l'heure d'hiver à l'automne 2010, après le passage à 2011
- Succès financier: Windows ;)
- Sans conséquence mais énervant : bugs @ Univ. Rennes, …
- D'autres : http://en.wikipedia.org/wiki/Computer_bug

System Complexity





Failures in Systems Engineering



Outline

① Issues in Software Engineering

② Evolution in Software Engineering

③ State of the Practice

④ Modeling in Software Engineering

Software Engineering

The production of operational software satisfying defined standards of quality...

... includes programming, but is more than programming!

The five components of Software Engineering [Meyer]:

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

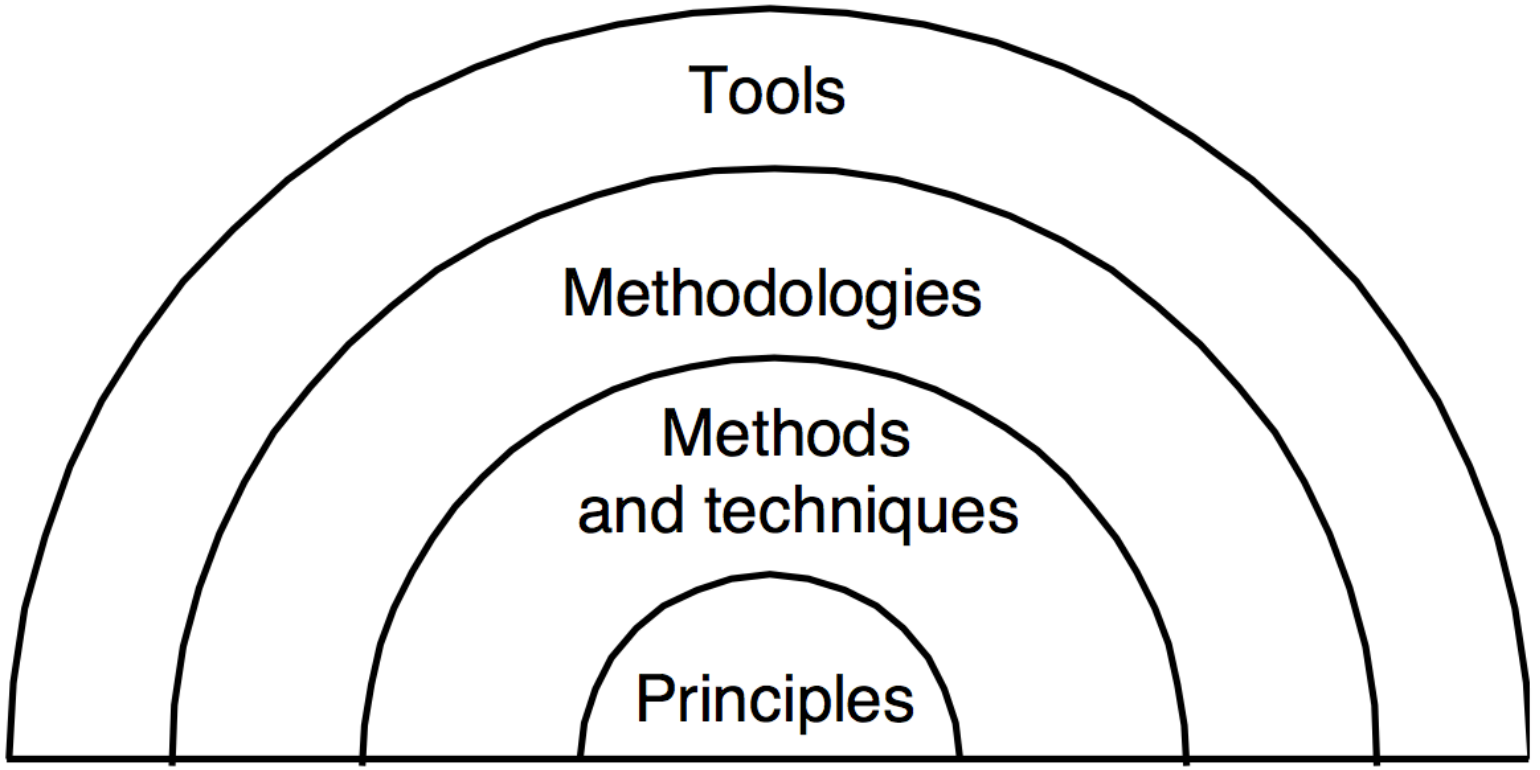
Software Engineering: definition

- is a **profession** dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build.

- is a **systematic approach** to the analysis, design, assessment, implementation, test, maintenance and re-engineering of a software by applying engineering to the software.

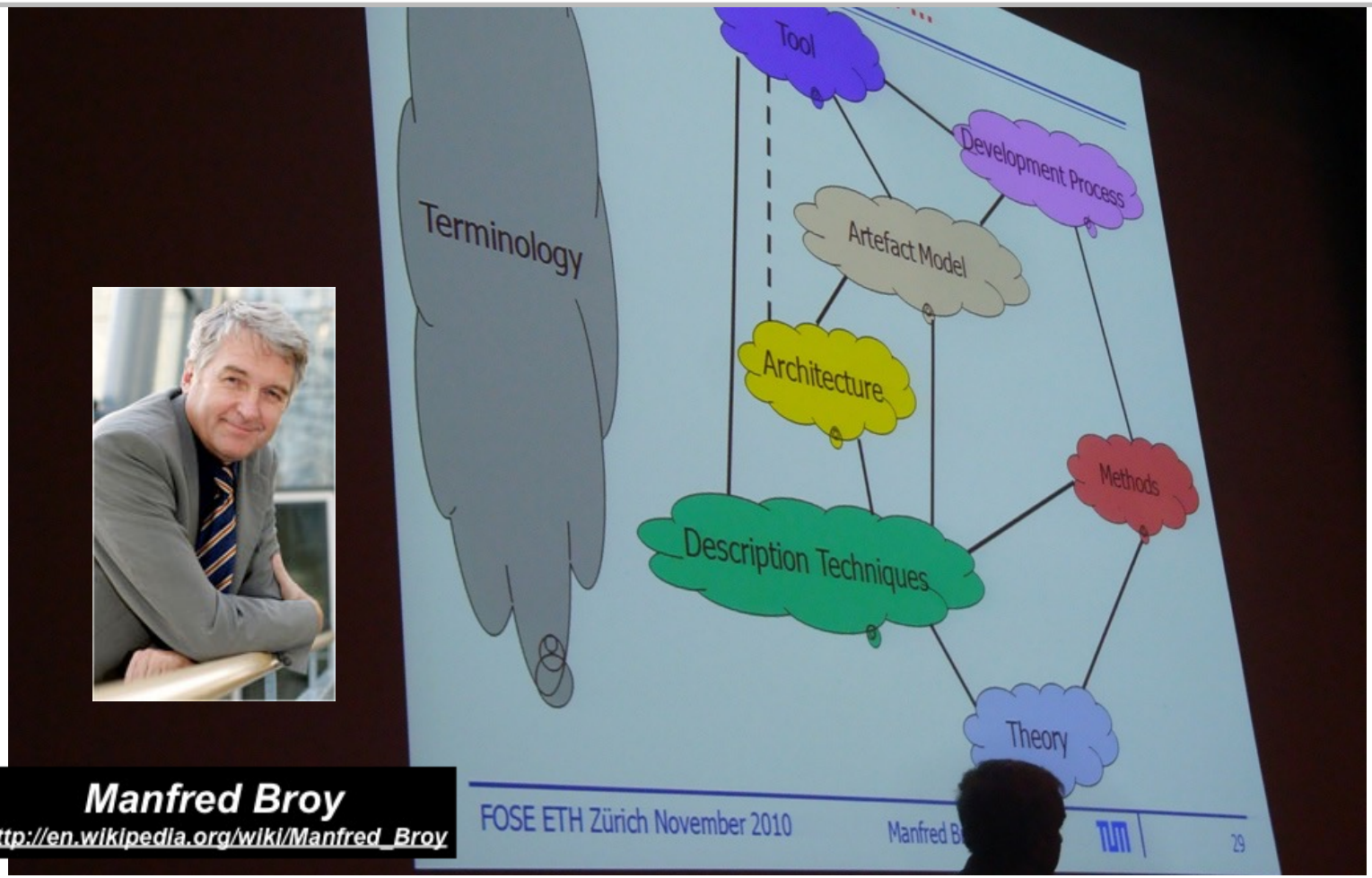
- first appeared in the 1968 NATO Software Engineering Conference (to provoke thought regarding the perceived "software crisis" at the time).

Software Engineering: Basics

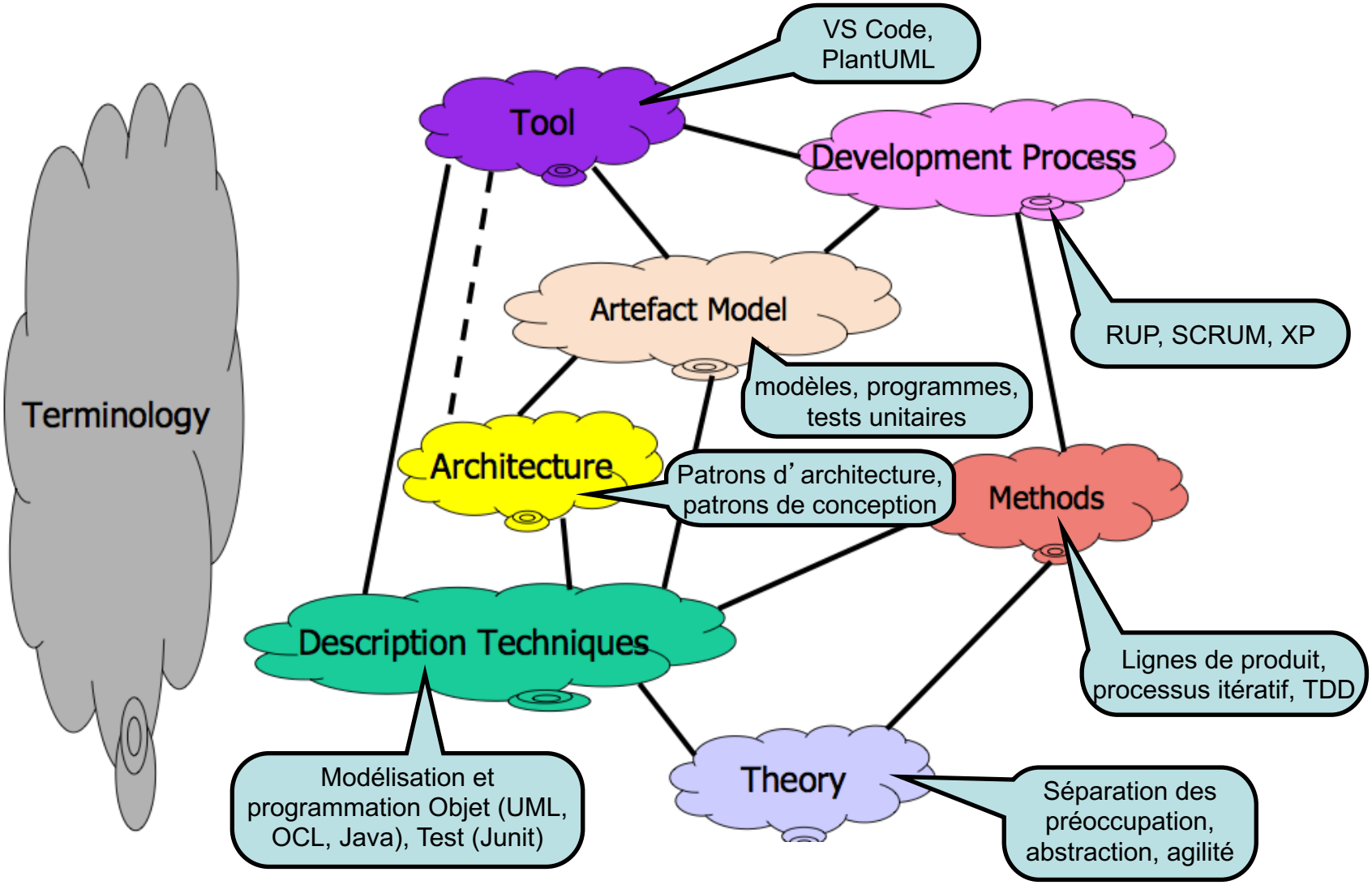


Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli.
Fundamentals of Software Engineering, 2nd edition. 2002.

What we need?

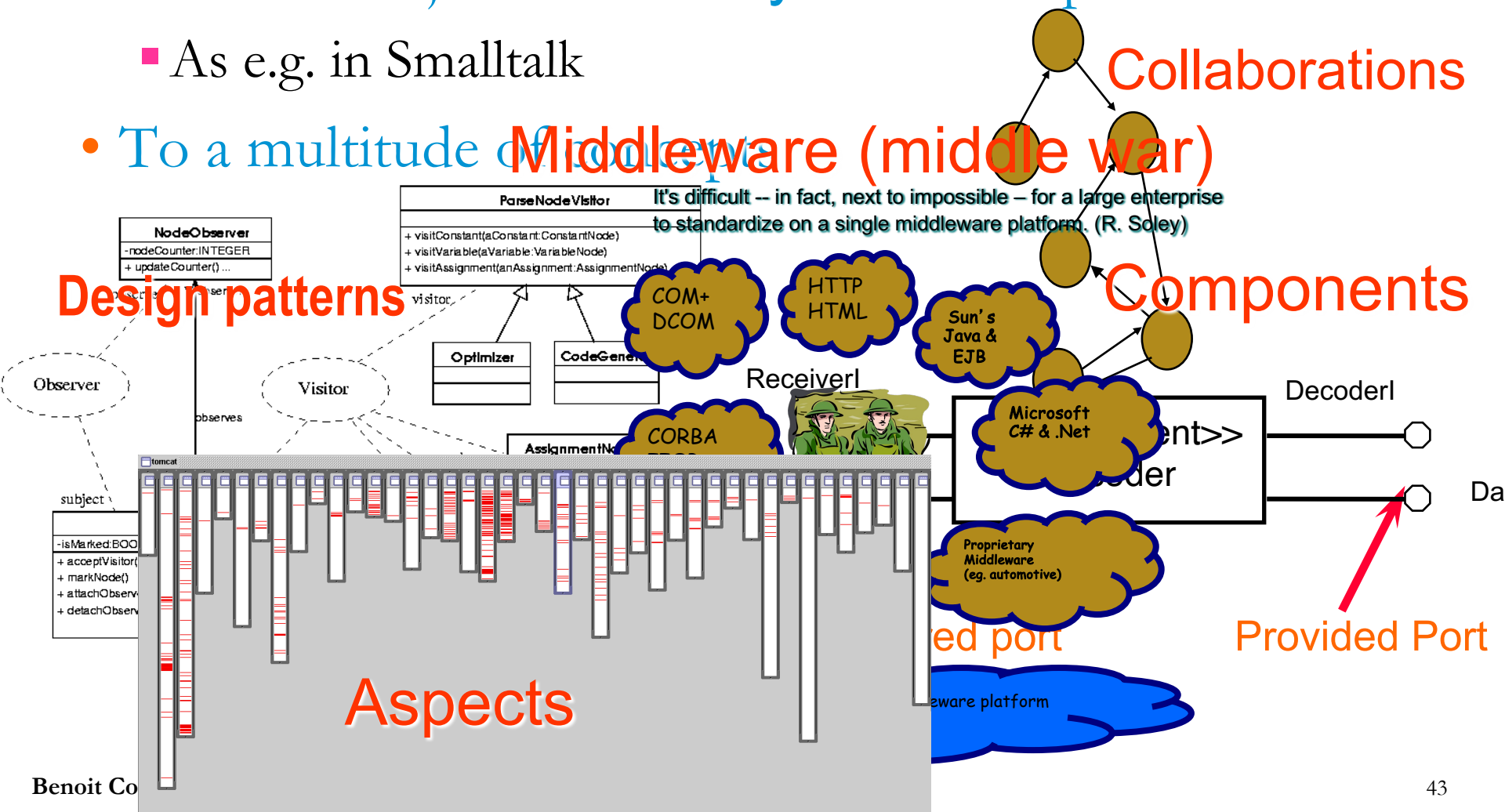

Tool
Development Process
Terminology
Artefact Model
Architecture
Methods
Description Techniques
Theory
FOSE ETH Zürich November 2010
Manfred Broy
http://en.wikipedia.org/wiki/Manfred_Broy

Software Engineering: Basics



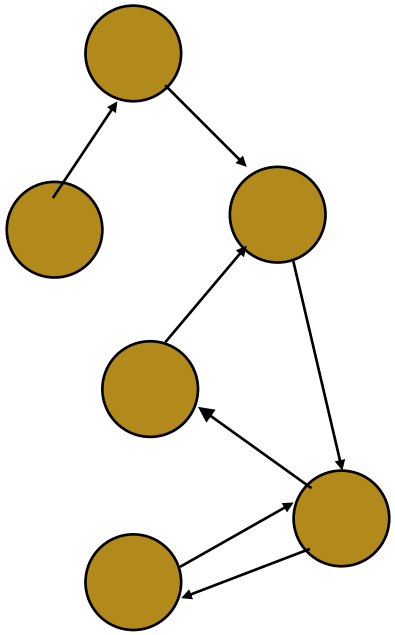
Once upon a time… software development looked simple

- From the object as the **only** one concept
  - As e.g. in Smalltalk
- To a multitude of concepts



Collaborations

- Objects should be as simple as possible
  - To enable modular understanding
- But then where is the complexity?
  - It is in the way objects interact!
  - Cf. Collaborations as a standalone diagram in UML (T. Reenskaug's works)



Design Patterns

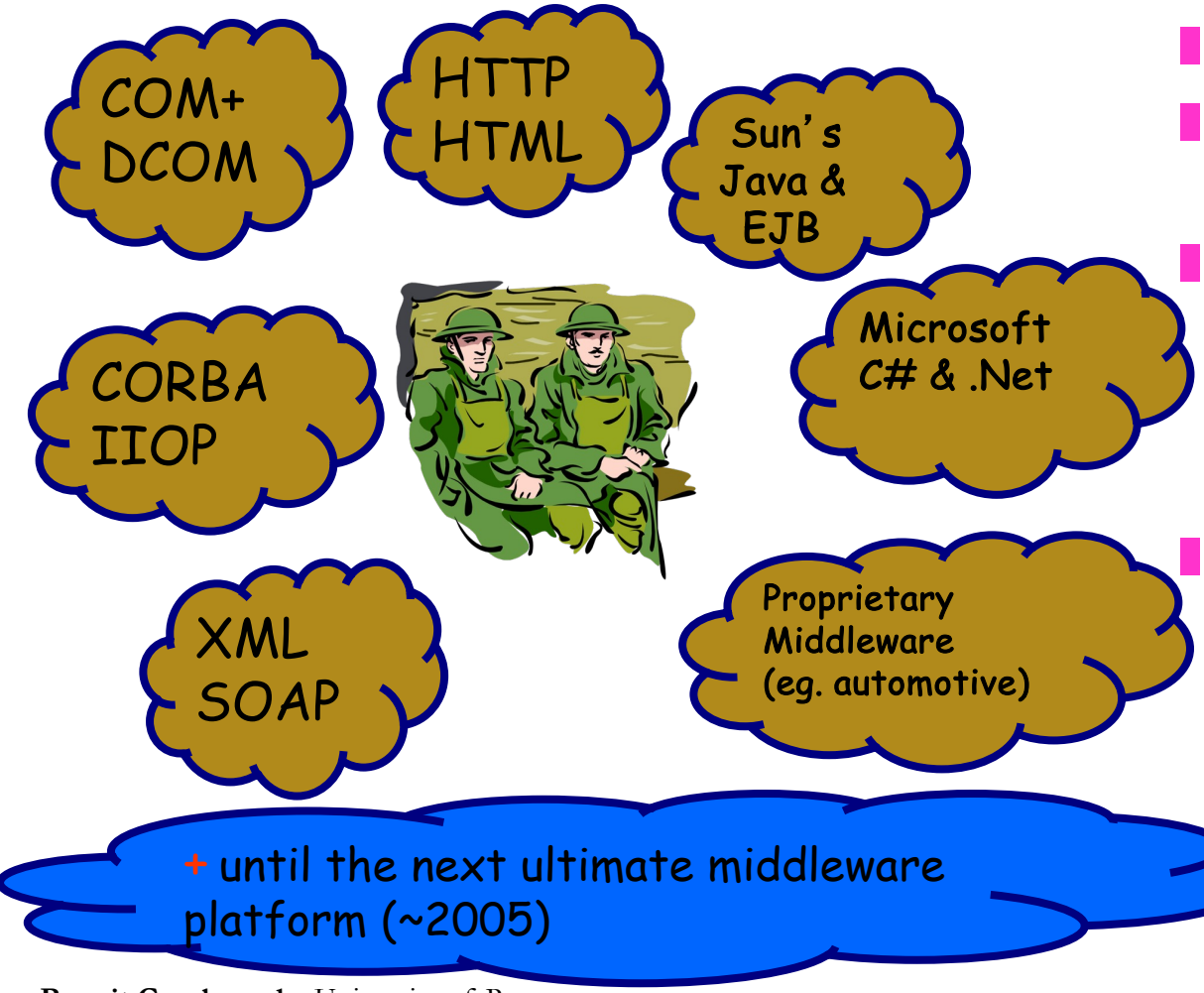
- Embody **architectural know-how** of experts
- As much about problems as about solutions
  - pairs problem/solution in a context
- About non-functional forces
  - reusability, portability, and extensibility…
- Not about classes & objects but **collaborations**
  - Actually, design pattern applications *are* parameterized collaborations

From Objects to Components

- Object = instance of a class
- Class = reusable unit of software
  - focus on structural and functional properties
  - development concept
- Component = deployment unit
  - focus on non-functional properties
  - installation/execution concept
    - Explicit dependencies
    - Configuration and connection

Middleware or Middle War?

It's difficult -- in fact, next to impossible – for a large enterprise to standardize on a single middleware platform. (R. Soley)



- No clear winner until now
- And probably not in the near future
- Migration is expensive and disruptive

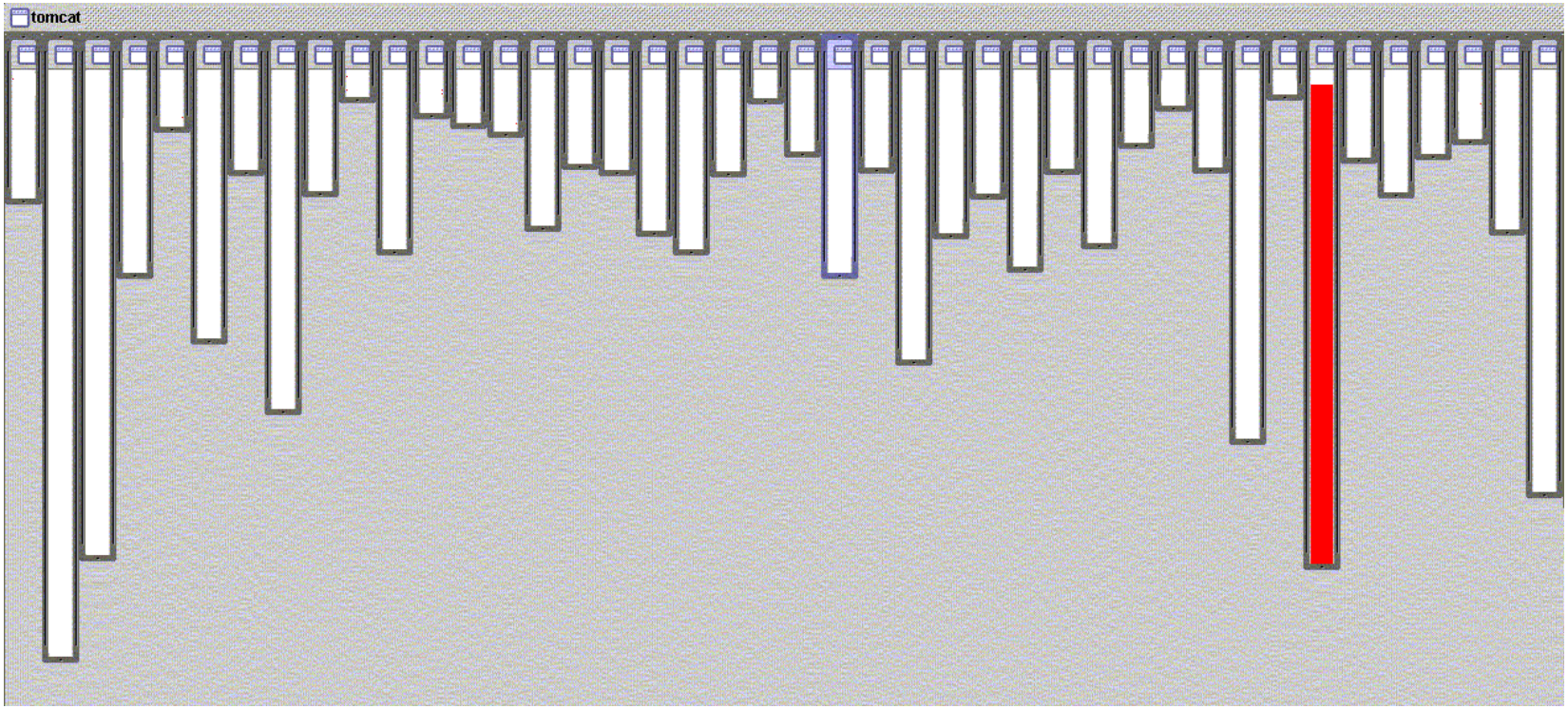
- The OMG tried to send in the ‘’ blue helmets’’ with the MDA initiative

Aspect Oriented Programming

- Kiczales et al., ECOOP'97
  - MIT's one of 10 key technologies for 2010
- Encapsulation of cross-cutting concerns in OO programs
  - Persistence, contract checking, etc.
- Weaving at some specific points (join points) in the program execution
  - *Hence more than macros on steroids*
- AspectJ for AOP in Java
  - Some clumsiness in describing dynamic join points
- What about Aspect Oriented Design ?

Good modularity

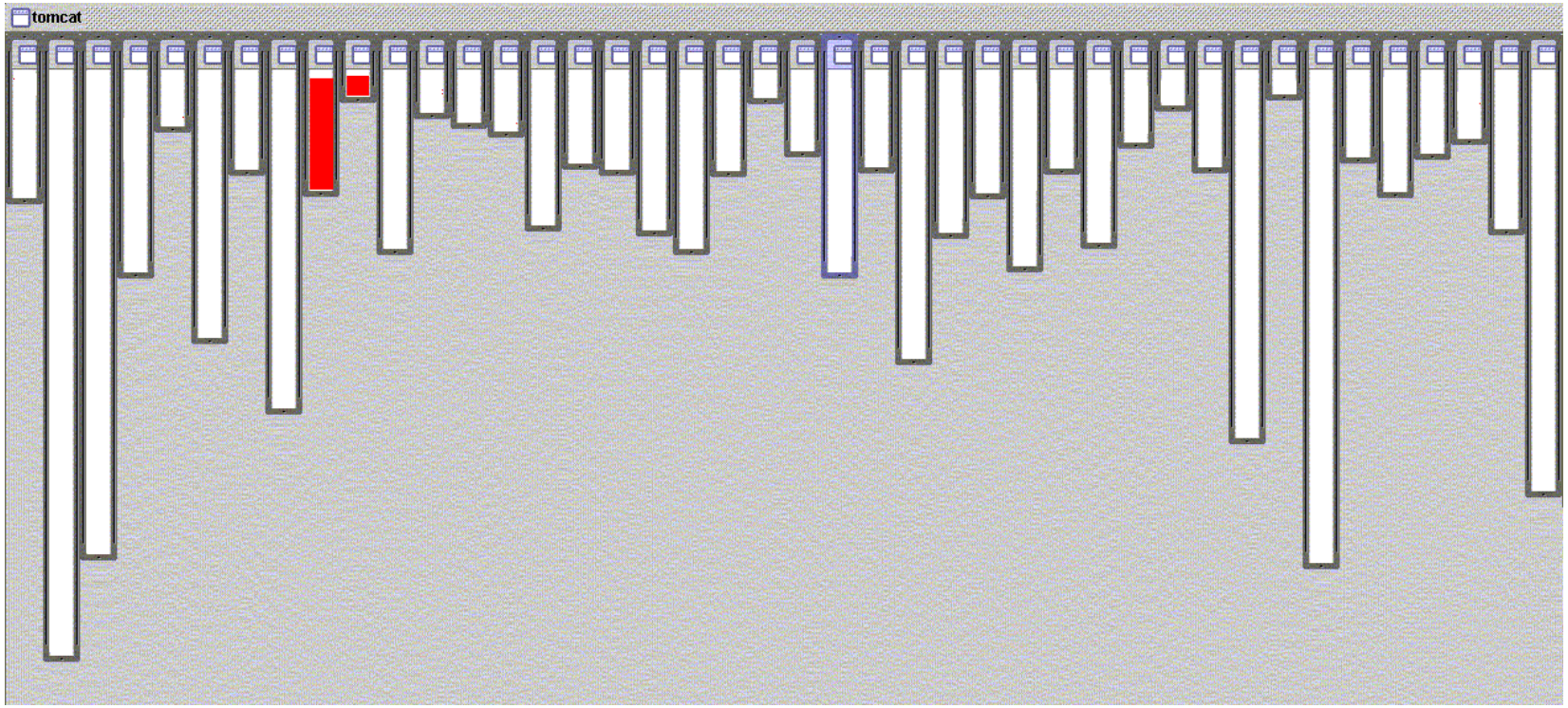
XML parsing



- XML parsing in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in one box

Good modularity

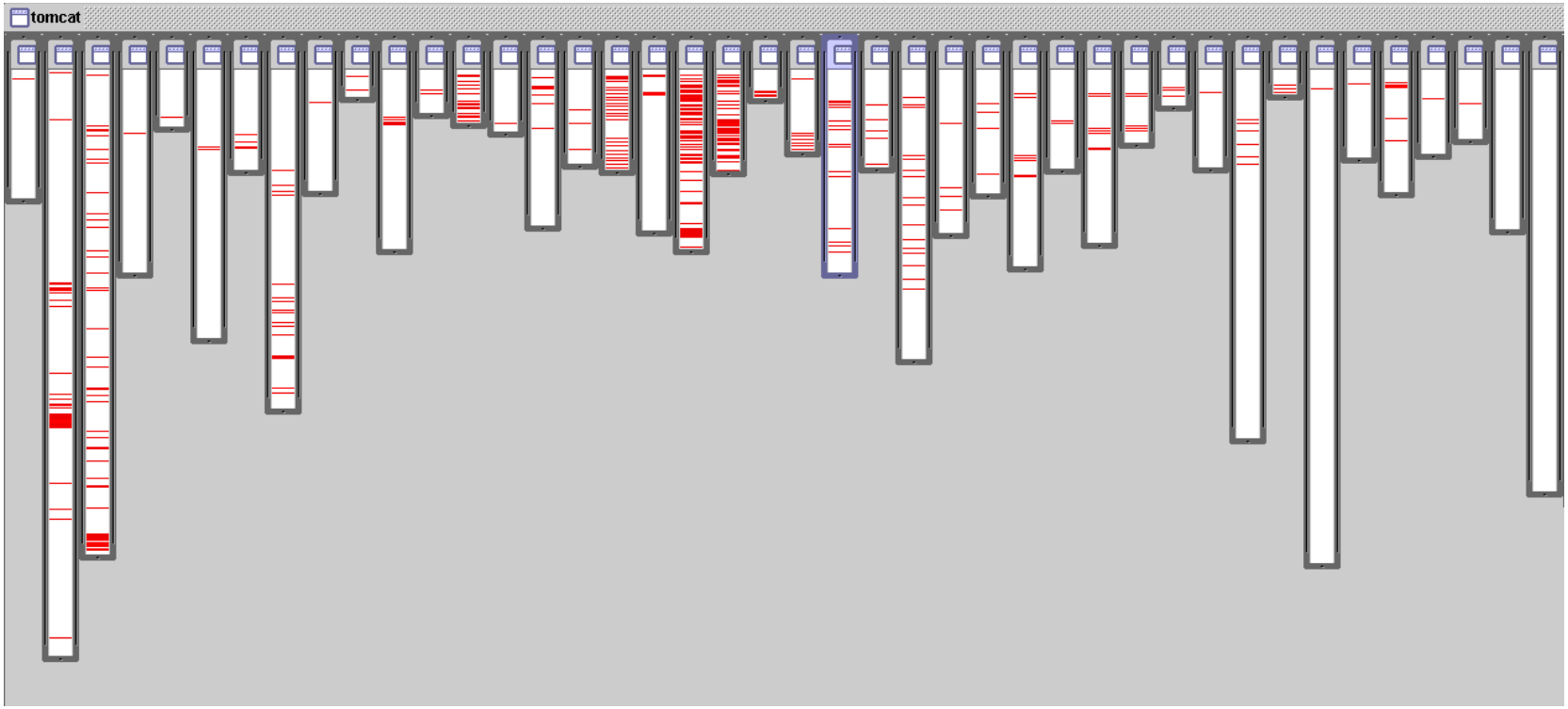
URL pattern matching



- URL pattern matching in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

Problems like…

logging is not modularized



- where is logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

AspectJ™ is…

- a small and well-integrated extension to Java
- a general-purpose AO language
  - just as Java is a general-purpose OO language
- freely available implementation
  - compiler is Open Source
- includes IDE support
  - Eclipse, VS Code, IntelliJ…
- user feedback is driving language design
  - users@aspectj.org
  - support@aspectj.org
- currently at 1.9.7 release

Expected benefits of using AOP

- good modularity,
  even for crosscutting concerns
  - less tangled code
  - more natural code
  - shorter code
  - easier maintenance and evolution
    - easier to reason about, debug, change
  - more reusable
    - library aspects
    - plug and play aspects when appropriate

Outline

① Issues in Software Engineering

② Evolution in Software Engineering

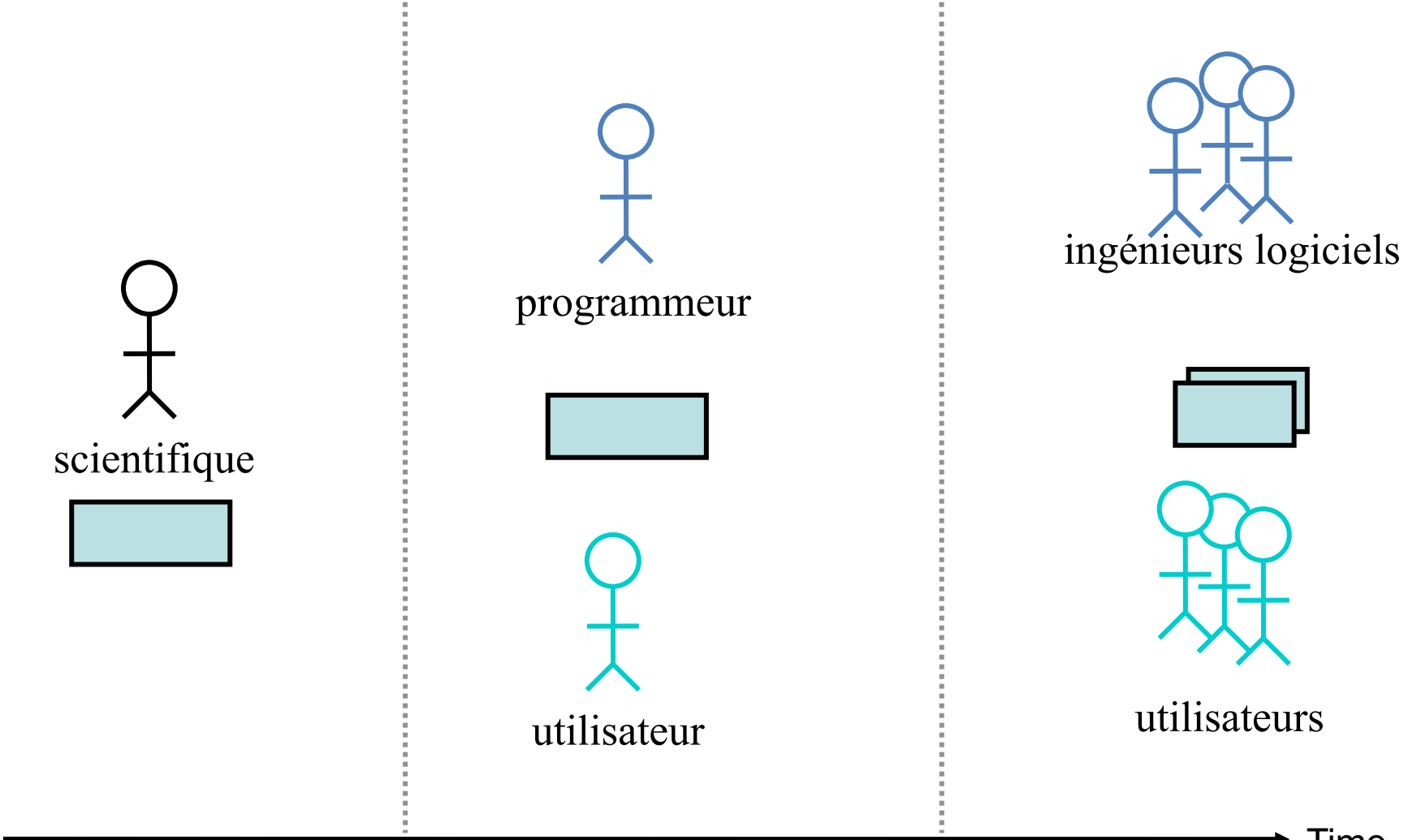
③ State of the Practice

④ Modeling in Software Engineering

Des logiciels complexes...

- Logiciels de grande taille
  - des millions de lignes de code
  - des équipes nombreuses
  - durée de vie importante
  - des lignes de produits
  - plateformes technologiques complexes
  - évolution continue
- Logiciels complexes
- Logiciels critiques
- …

Évolution des acteurs



Oui, Oui, Oui, mais ...

- La mentalité de "l'informaticien moyen" a t elle radicalement évoluée ?
- Du "Codeur", au "Programmeur", à l' "Ingénieur Logiciel", ...
- L' "Ingénieur logiciel"
  - prouve-t-il ses programmes ?
  - maintient-il à jour la documentation, les spécifications ?

État de la pratique

50 ans après les débuts de l'informatique,

pour "l'informaticien moyen"

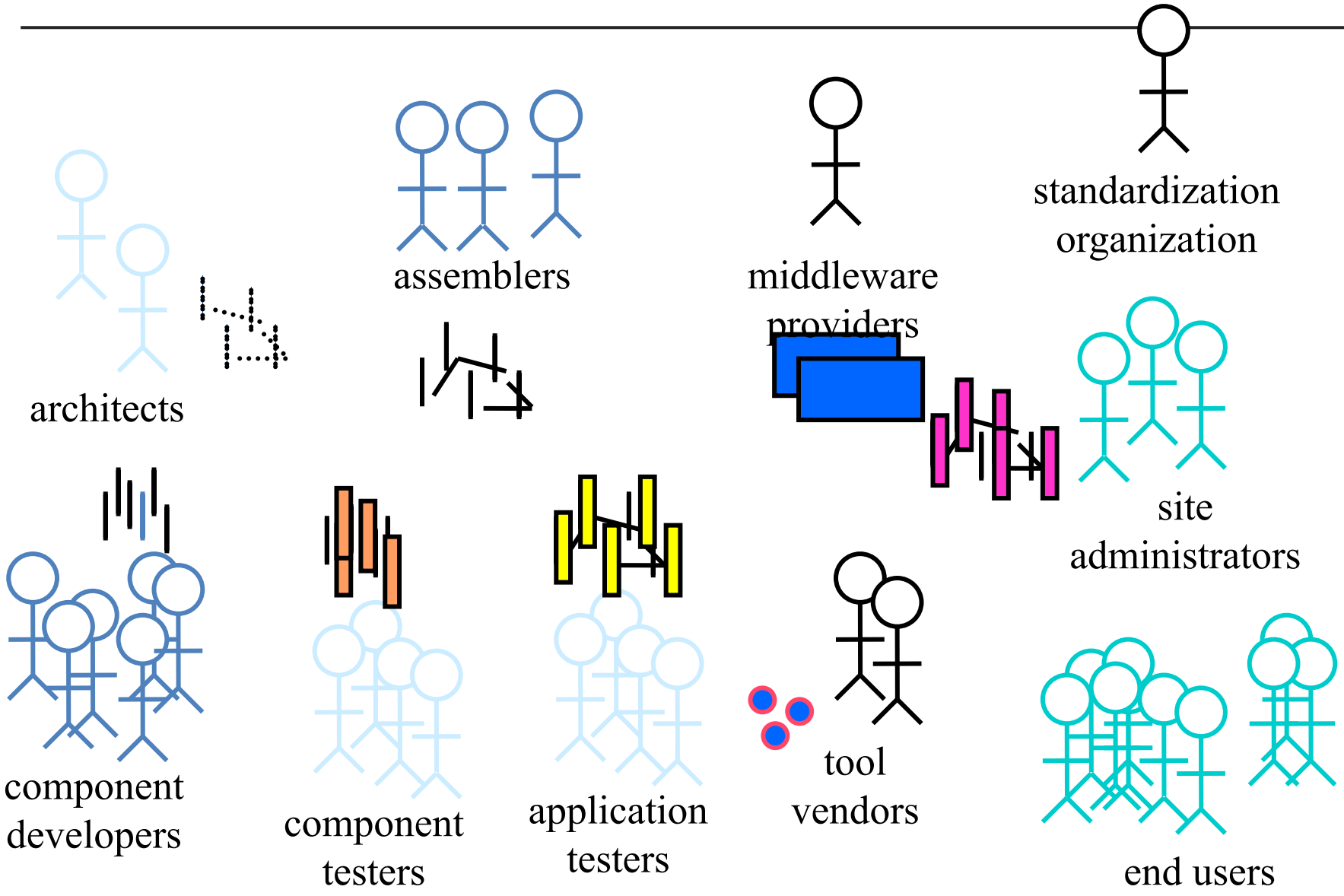
la seule chose importante dans le logiciel c'est

...

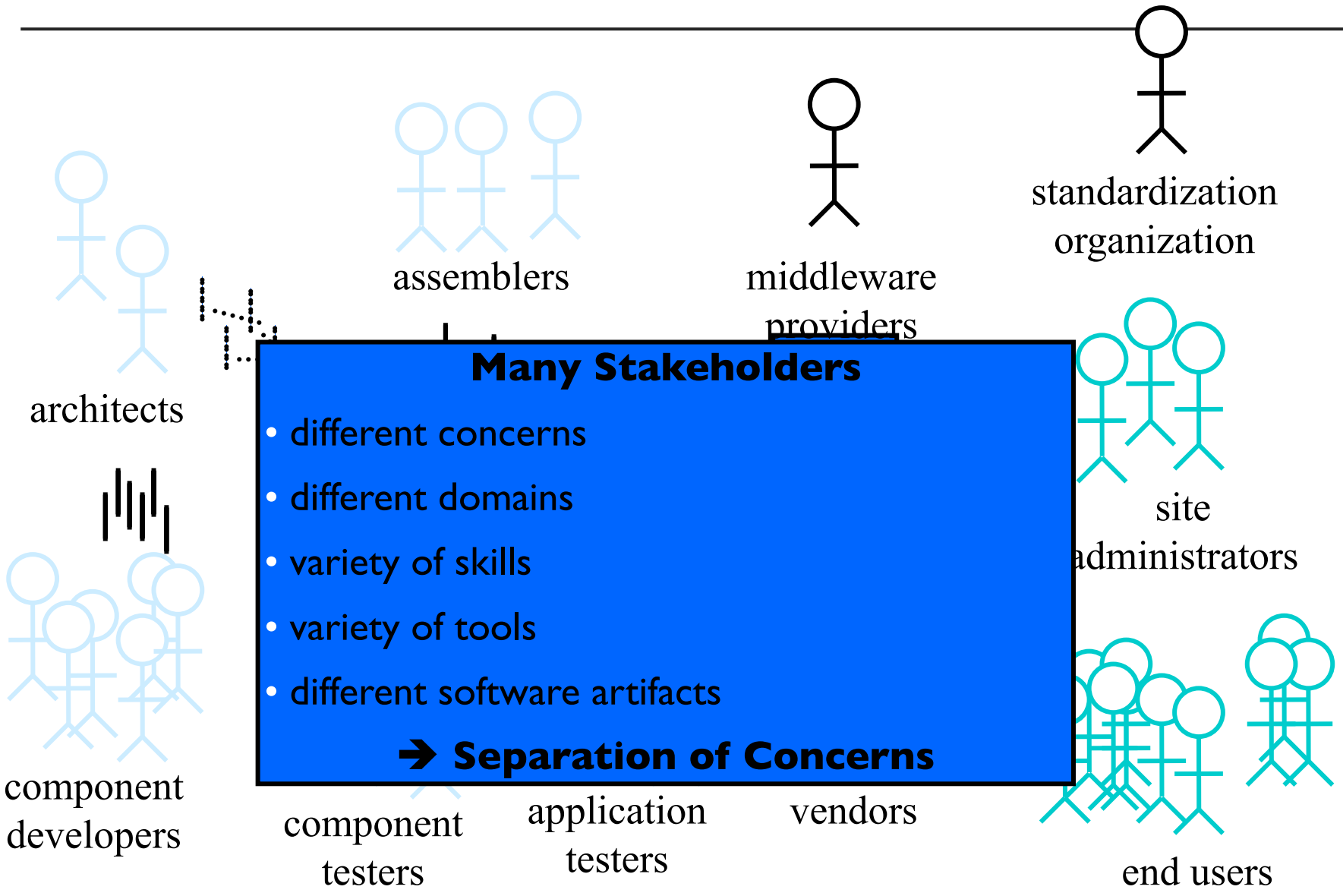
le Code !

=> Ingénierie Dirigée par le code

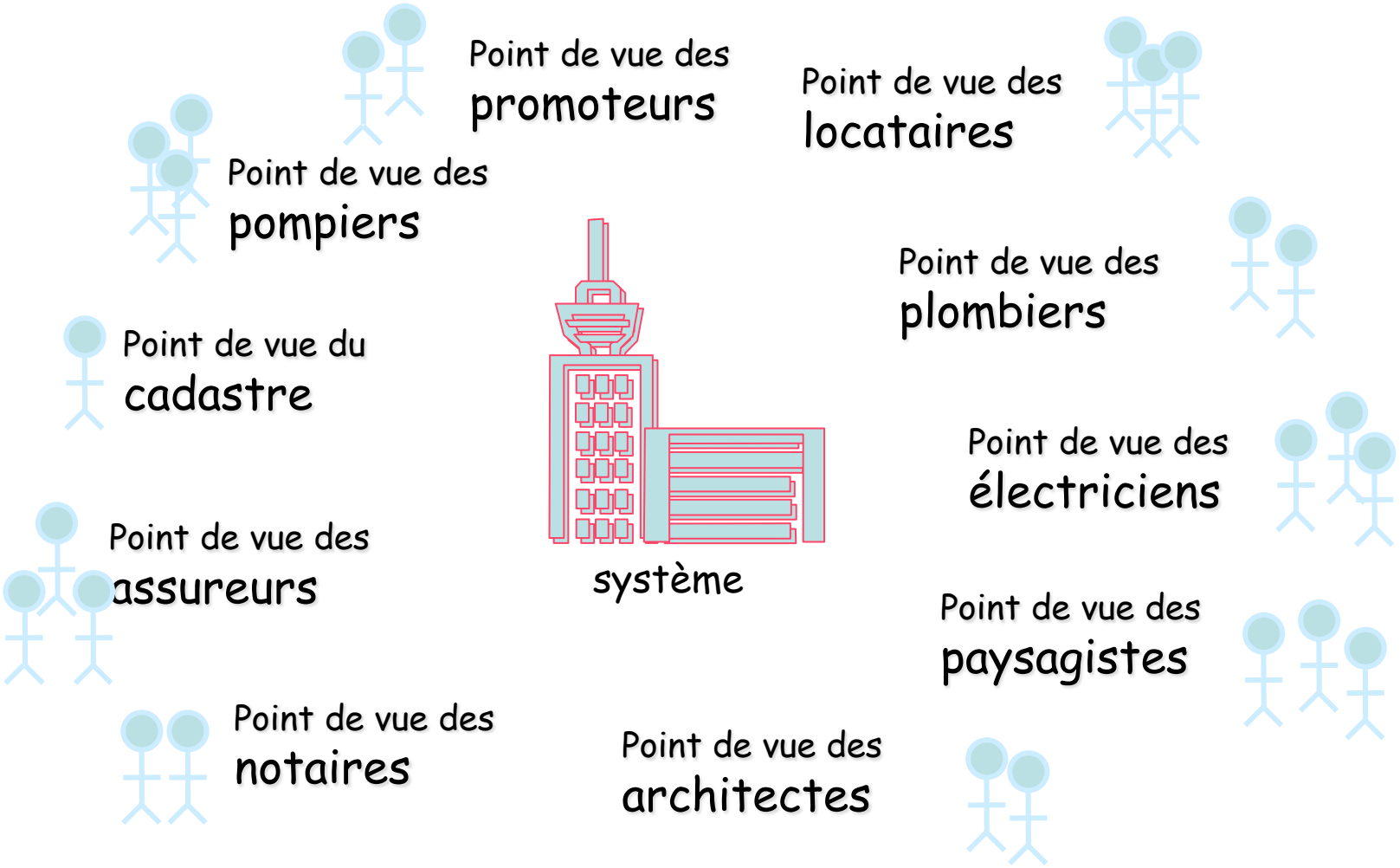
Status in Software Industry



Status in Software Industry



Séparations des préoccupations



Séparations des préoccupations

Utile même pour
des systèmes
"moins" complexes

Point de vue du
propriétaire

Point de vue du
plombier

Point de vue du
cadastre

système

Point de vue de l'
électricien

Point de vue de l'
architecte

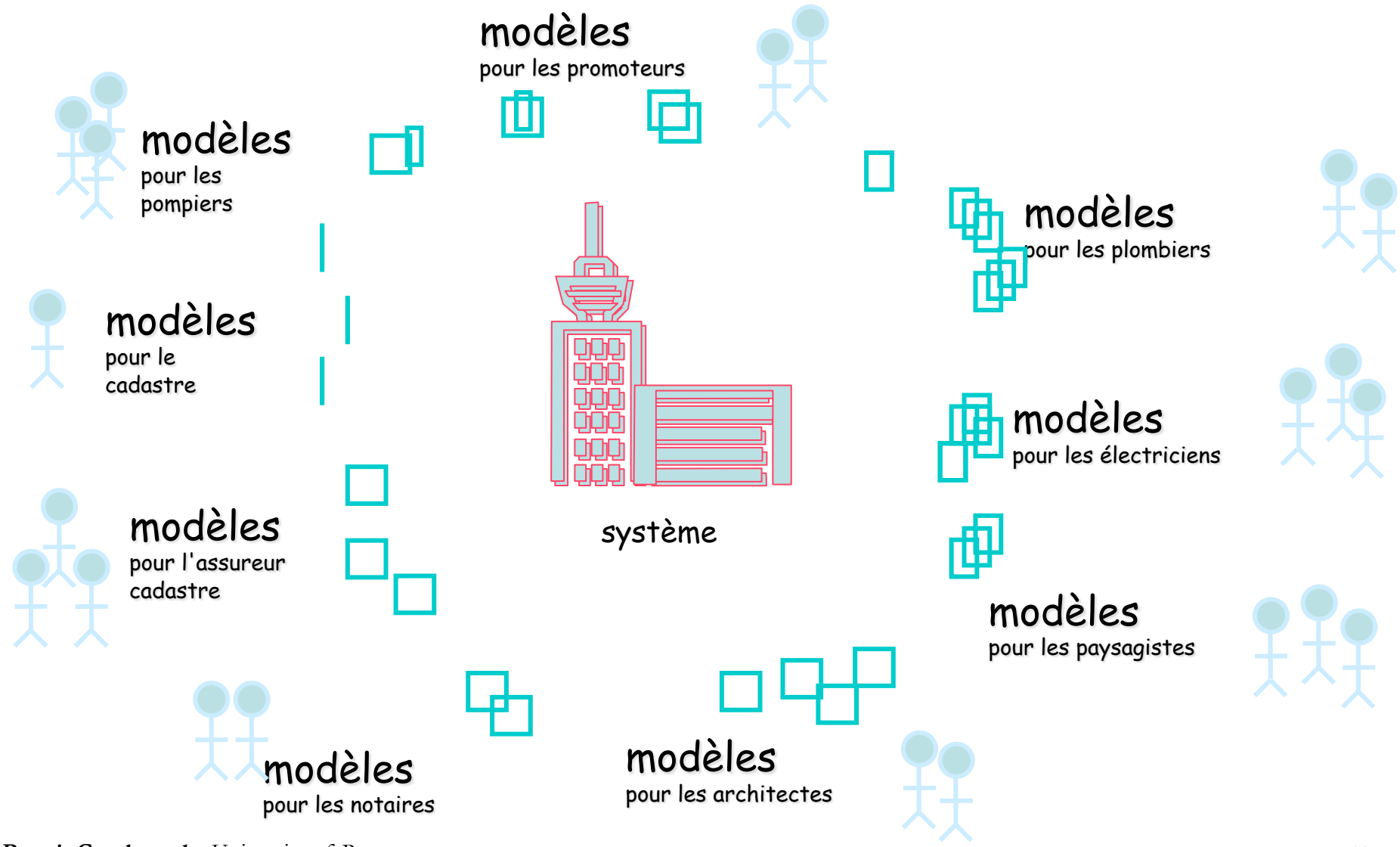
Point de vue du
maçon

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

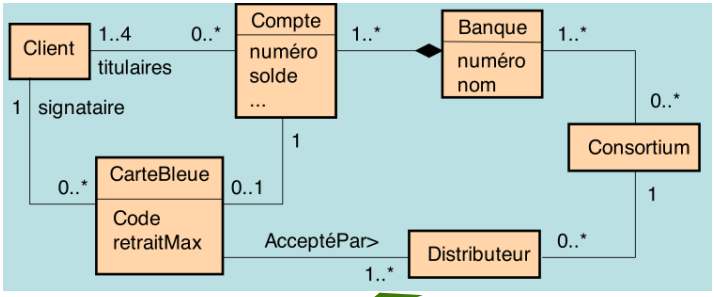


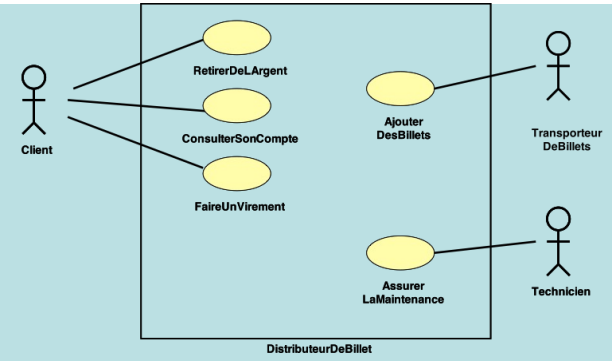
Ingénierie Dirigée par le Code

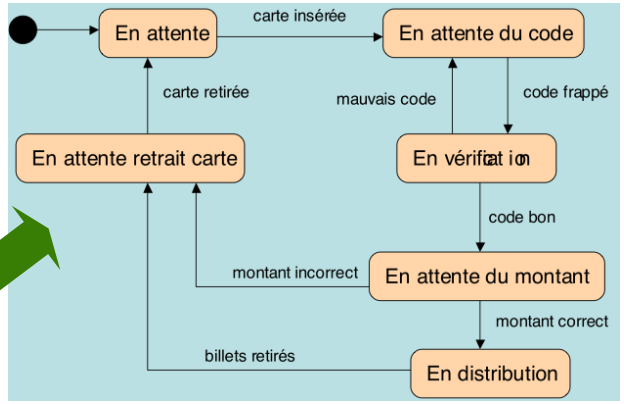
ou

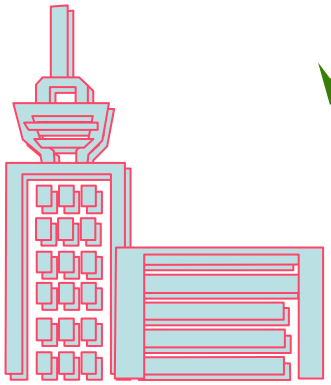
Ingénierie Dirigée par les Modèles ?

Telle est la question...

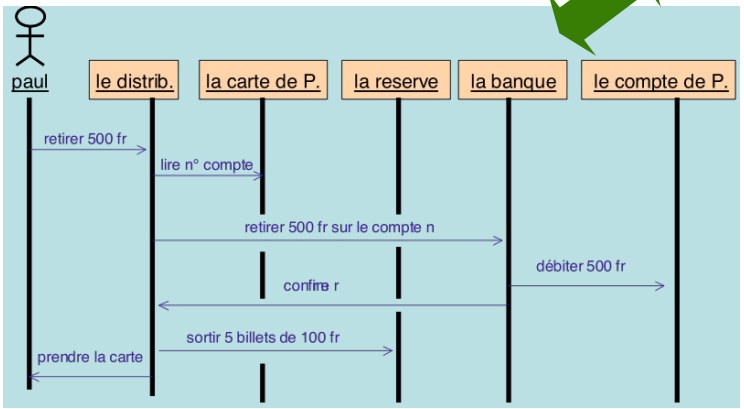

Client
1..4
titulaires
0..*
Compte
numéro
solde
...
1..*
Banque
numéro
nom
1..*
0..*
Consortium
1
signataire
1
0..*
CarteBleue
Code
retraitMax
0..1
AcceptéPar>
Distributeur
0..*
1..*

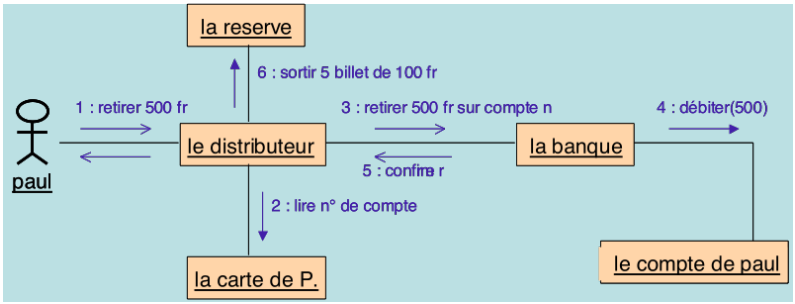

Client
RetirerDeLArgent
ConsulterSonCompte
FaireUnVirement
Ajouter
DesBillets
Assurer
LaMaintenance
Transporteur
DeBillets
Technicien
DistributeurDeBillet


En attente
carte insérée
En attente du code
carte retirée
mauvais code
code frappé
En attente retrait carte
En vérification
code bon
montant incorrect
En attente du montant
montant correct
billets retirés
En distribution



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

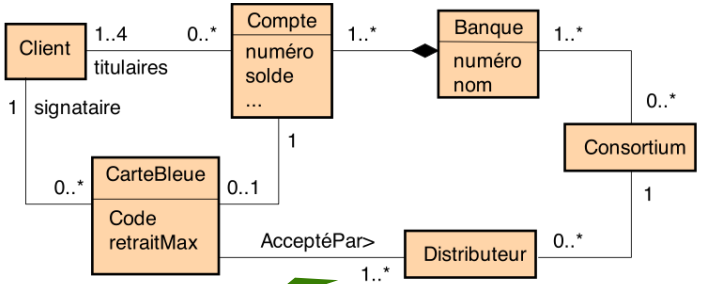

la reserve
6 : sortir 5 billet de 100 fr
1 : retirer 500 fr
3 : retirer 500 fr sur compte n
4 : débiter(500)
le distributeur
la banque
5 : confirmer
paul
2 : lire n° de compte
la carte de P.
le compte de paul

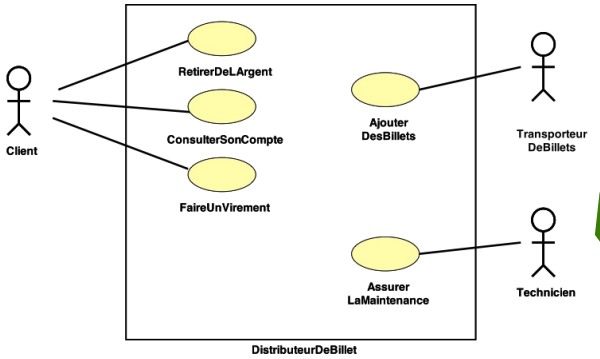
Oui, Oui, Oui, mais ...

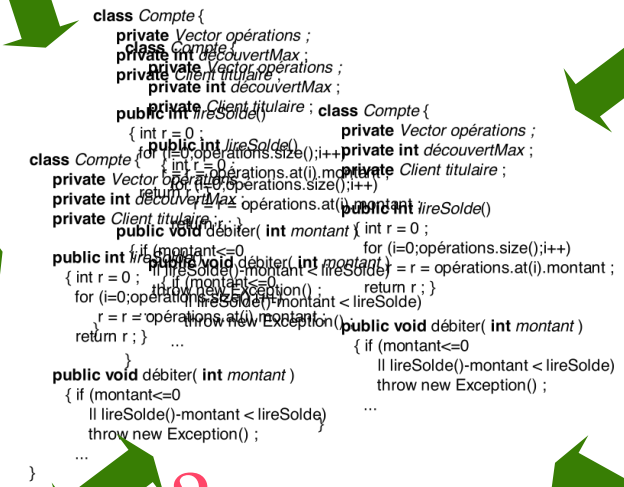
pour "l'informaticien moyen"

la seule chose importante dans le logiciel
c'est ...

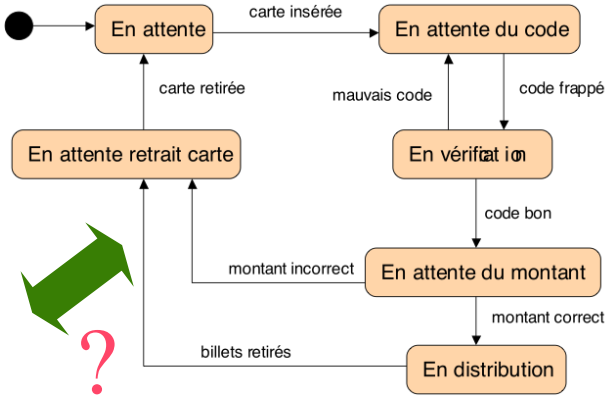
le Code !

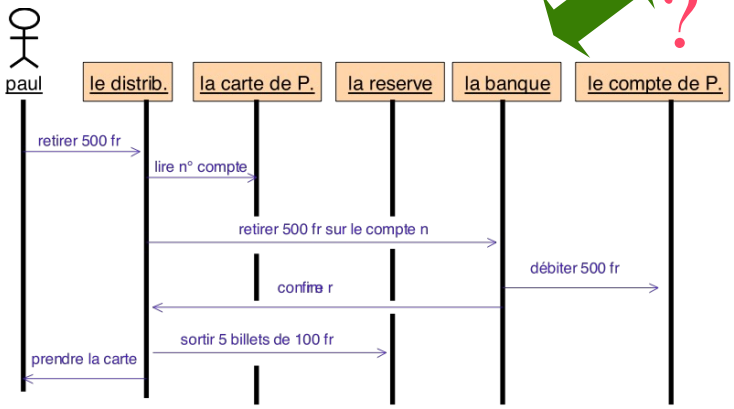


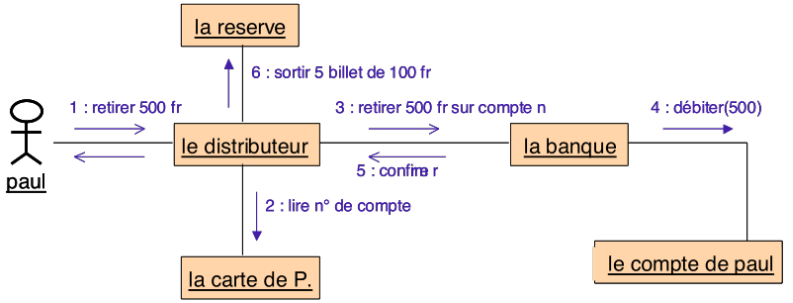




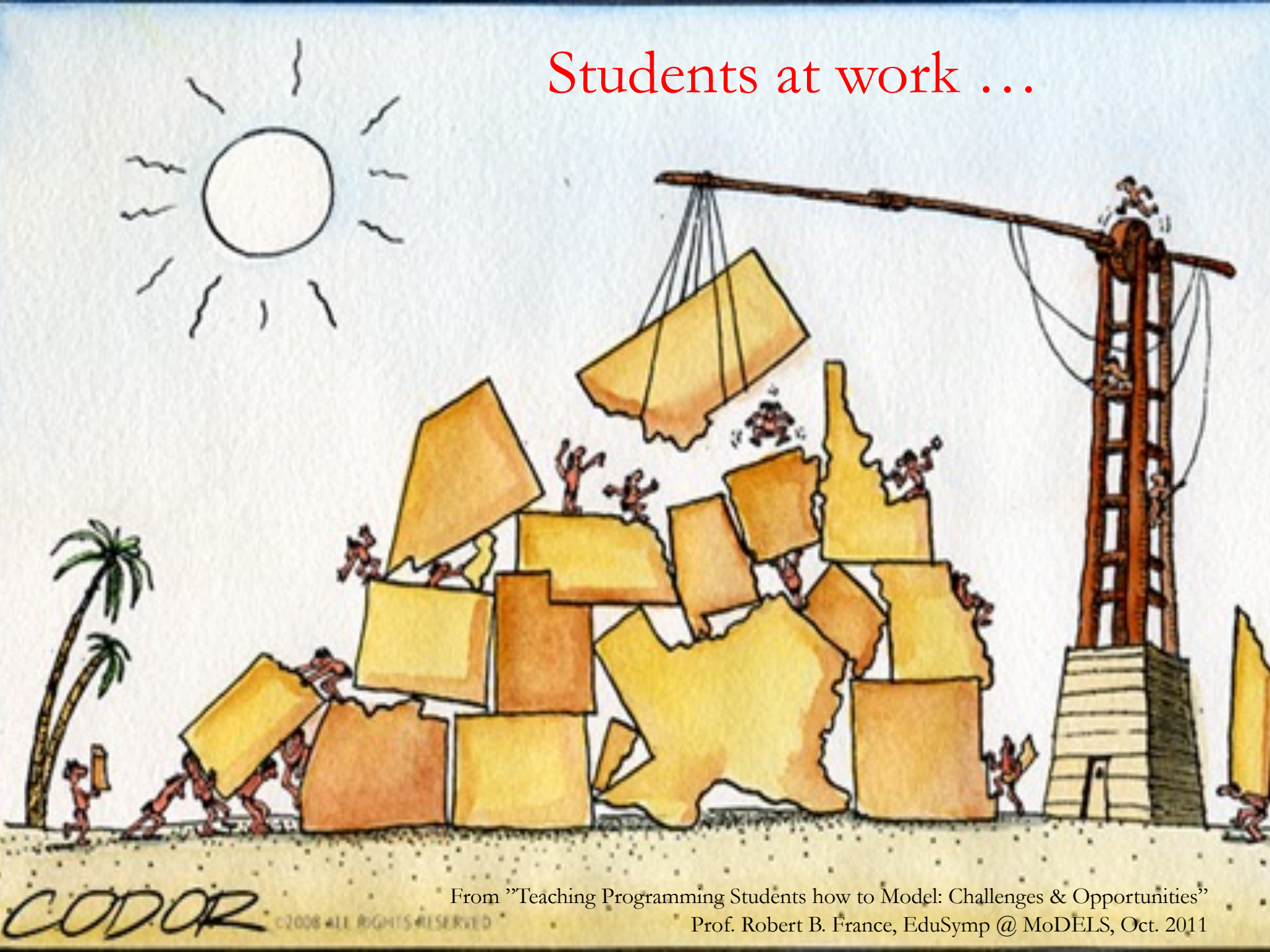
```
class Compte {
   private Vector opérations ;
   private int découvertMax ;
   private Client titulaire ;
   public int lireSolde()
     { int r = 0 ;
       for (i=0;opérations.size();i++)
         r = r = opérations.at(i).montant ;
       return r ; }
   public void débiter( int montant )
     { if (montant<=0
         ll lireSolde()-montant < lireSolde)
         throw new Exception() ;
       ...
     }
}
```







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

EduSymp MODELS 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/

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

Outline

① Issues in Software Engineering

② Evolution in Software Engineering

③ State of the Practice

④ Modeling in Software Engineering

Model to 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."

Jeff Rothenberg

The Nature of Modeling

John Wiley & Sons, Inc., August 1989

Model

"*A model represents reality for a 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

John Wiley & Sons, Inc., August 1989

Exemples de modèles

- **Modèle météorologique –** à partir de données d'observation (satellite…), il permet de prévoir les conditions climatiques pour les jours à venir.
- **Modèle économique –** peut par exemple permettre de simuler l'évolution de la bourse en fonction d'hypothèses macro-économiques (évolution du chômage, taux de croissance…).
- **Modèle démographique –** définit la composition d'un panel d'une population et son comportement, dans le but de fiabiliser des études statistiques, d'augmenter l'impact de démarches commerciales, etc.

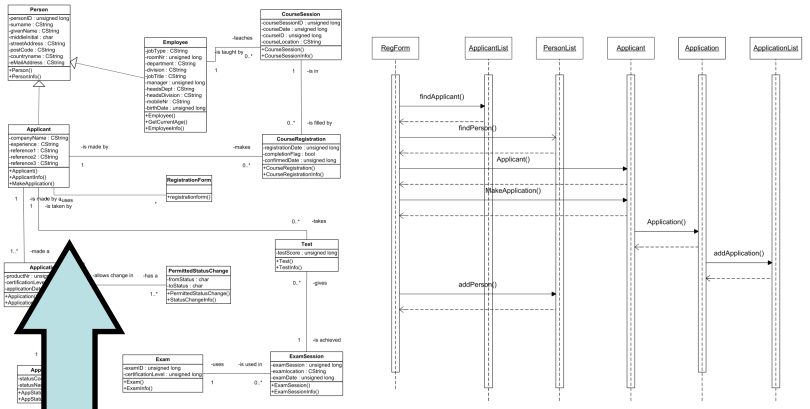
Modeling in Science & Engineering

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

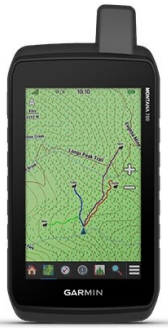
Specificity of Engineering: Model something not yet existing (in order to build it)

M_1
(modeling space)

Is represented by

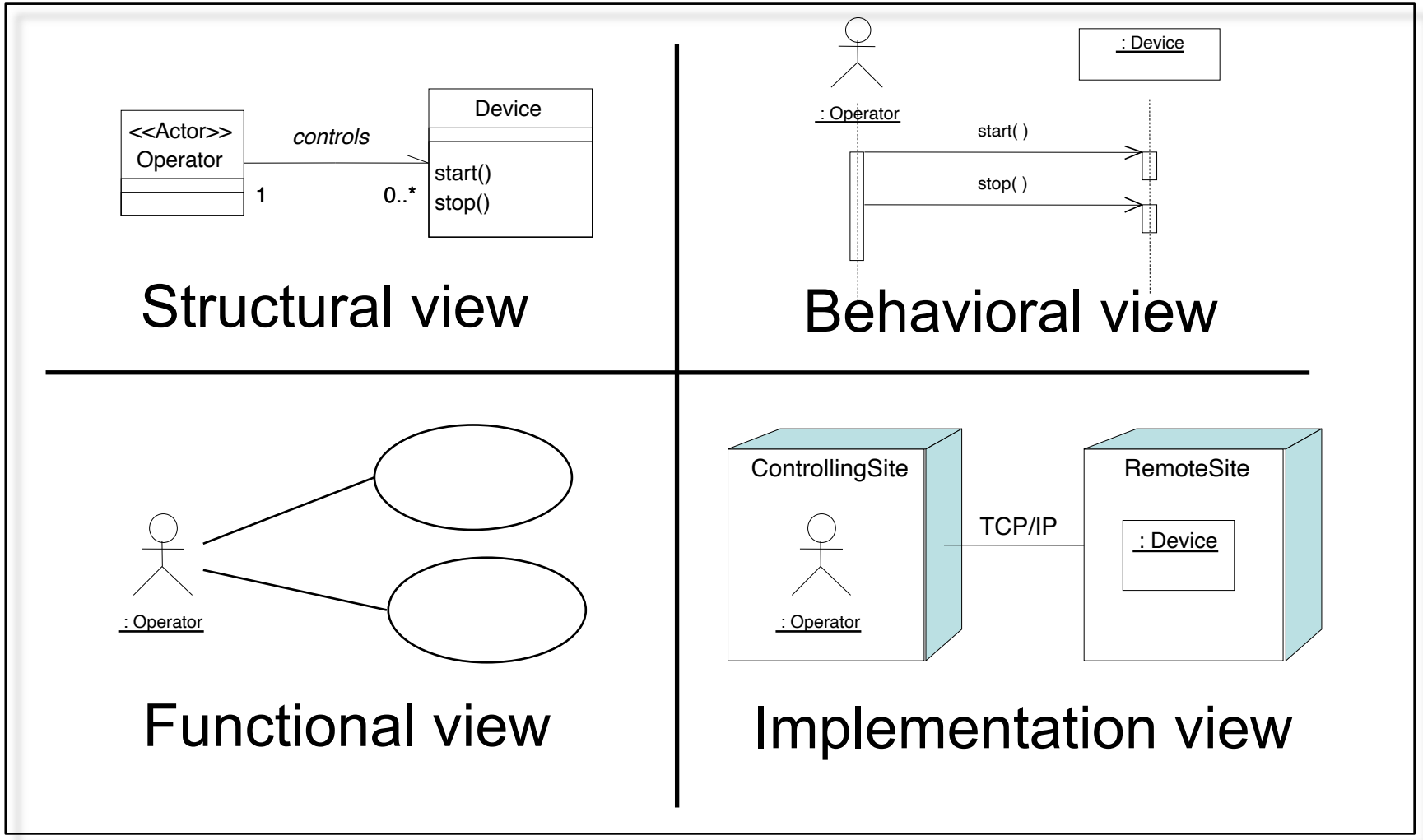


M_0
(the world)



UML: one model,

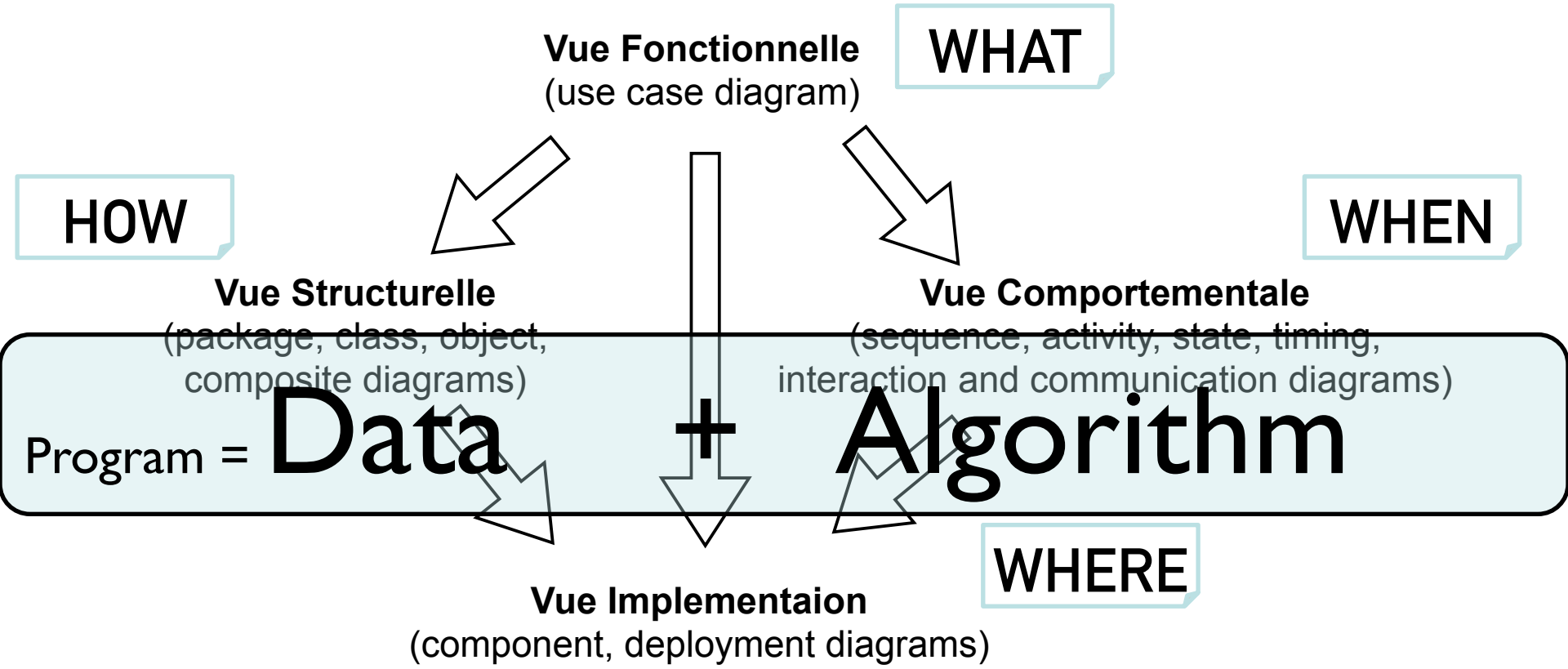
4 main dimensions, multiple views



The X diagrams of UML

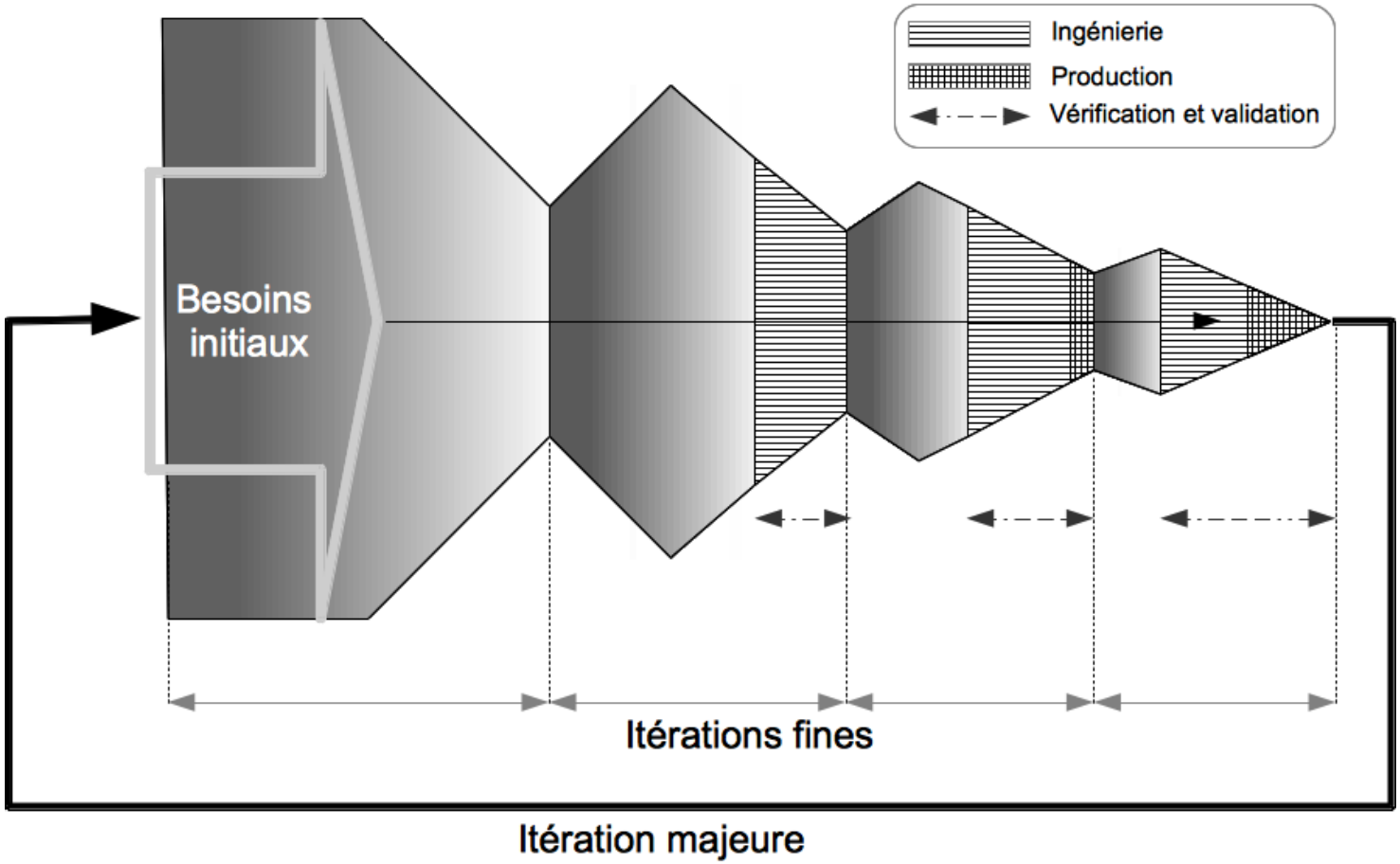
- Modeling along 4 main viewpoints:
  - User view *(What?)*
    - Use cases Diagram
  - Static Aspect *(Who?)*
    - Describes objects and their relationships (Class & Object Diagrams)
    - Structuring with packages
  - Dynamic Aspects *(When?)*
    - Sequence Diagram
    - Collaboration Diagram
  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

Une démarche sous-jacente



- La démarche doit être formalisée au sein d'un processus
- Il existe certains processus standard (e.g., RUP)

Une démarche sous-jacente



« Sketching User Experiences: Getting the Design Right and the Right Design » (Bill Buxton, Morgan Kaufmann, 2007)

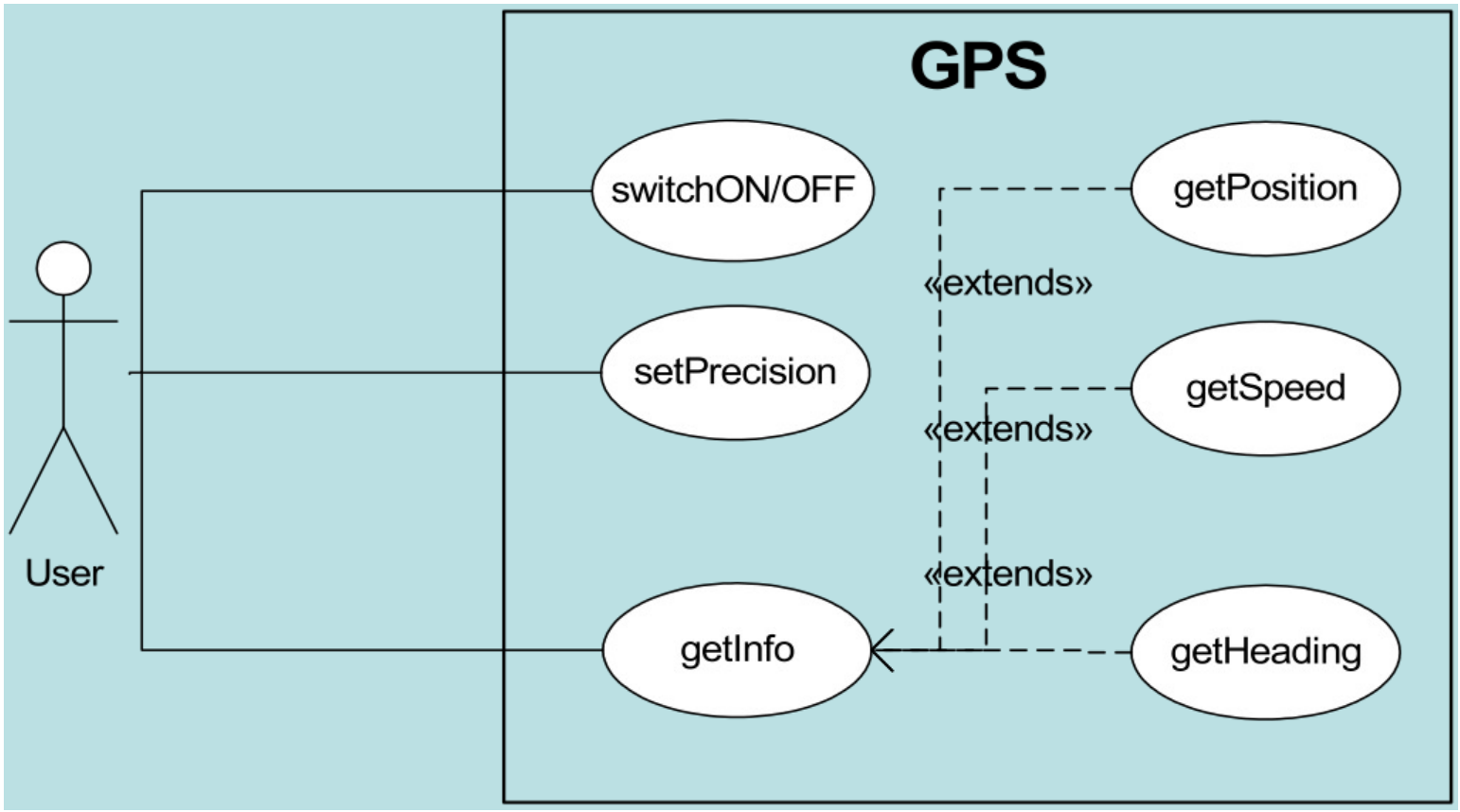
Example

- 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)

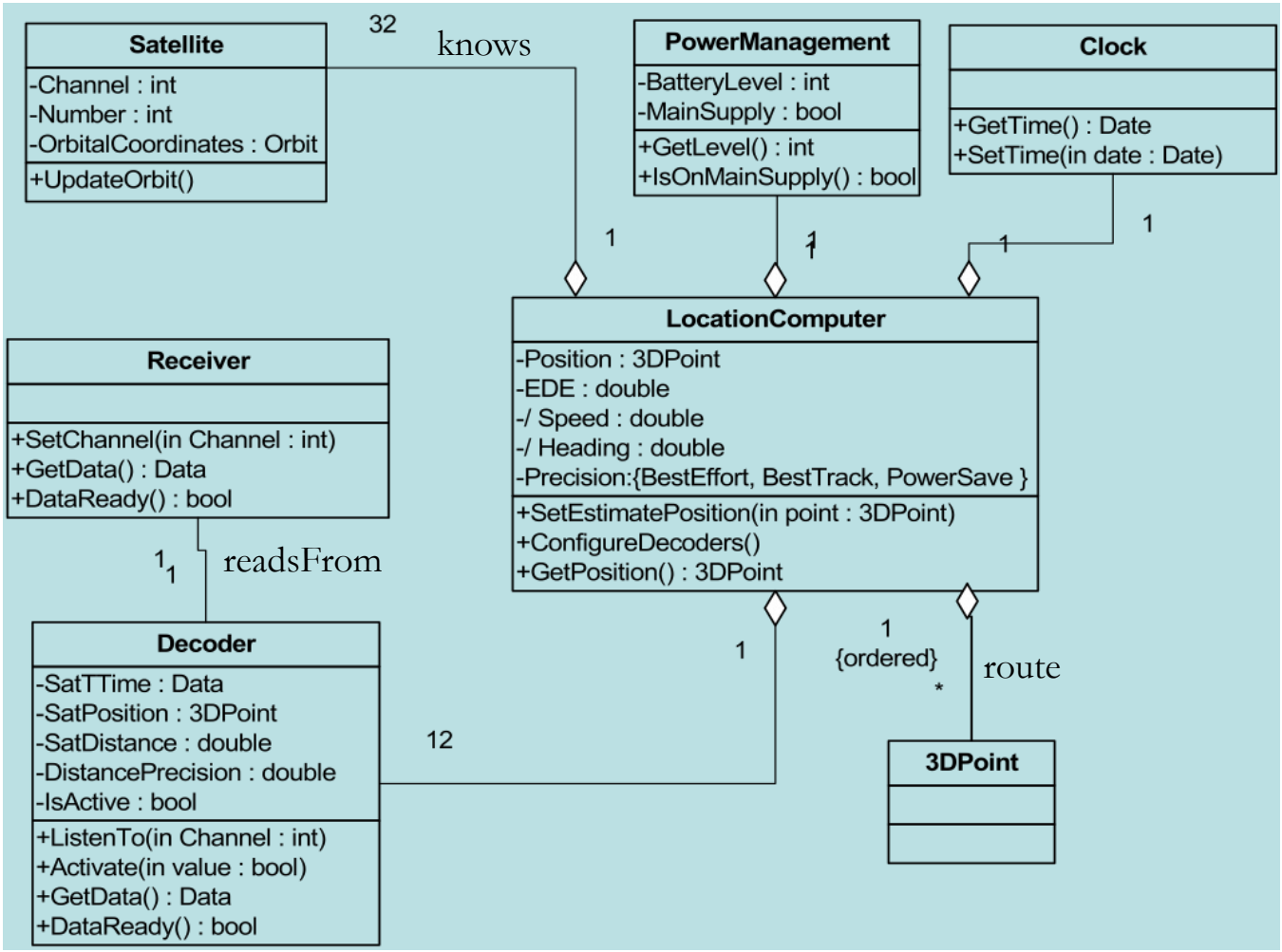
Modeling a (simplified) GPS device

- Use case diagram



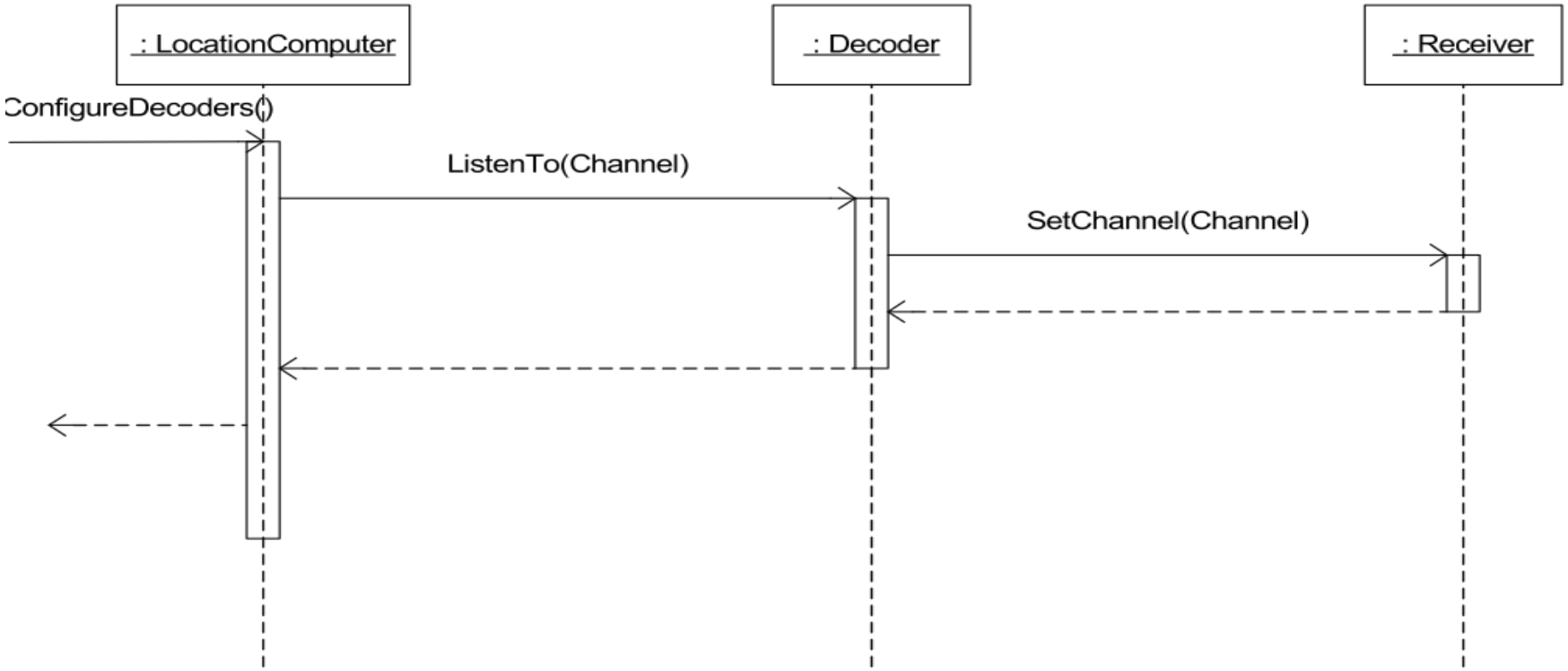
Modeling a (simplified) GPS device

- Class diagram



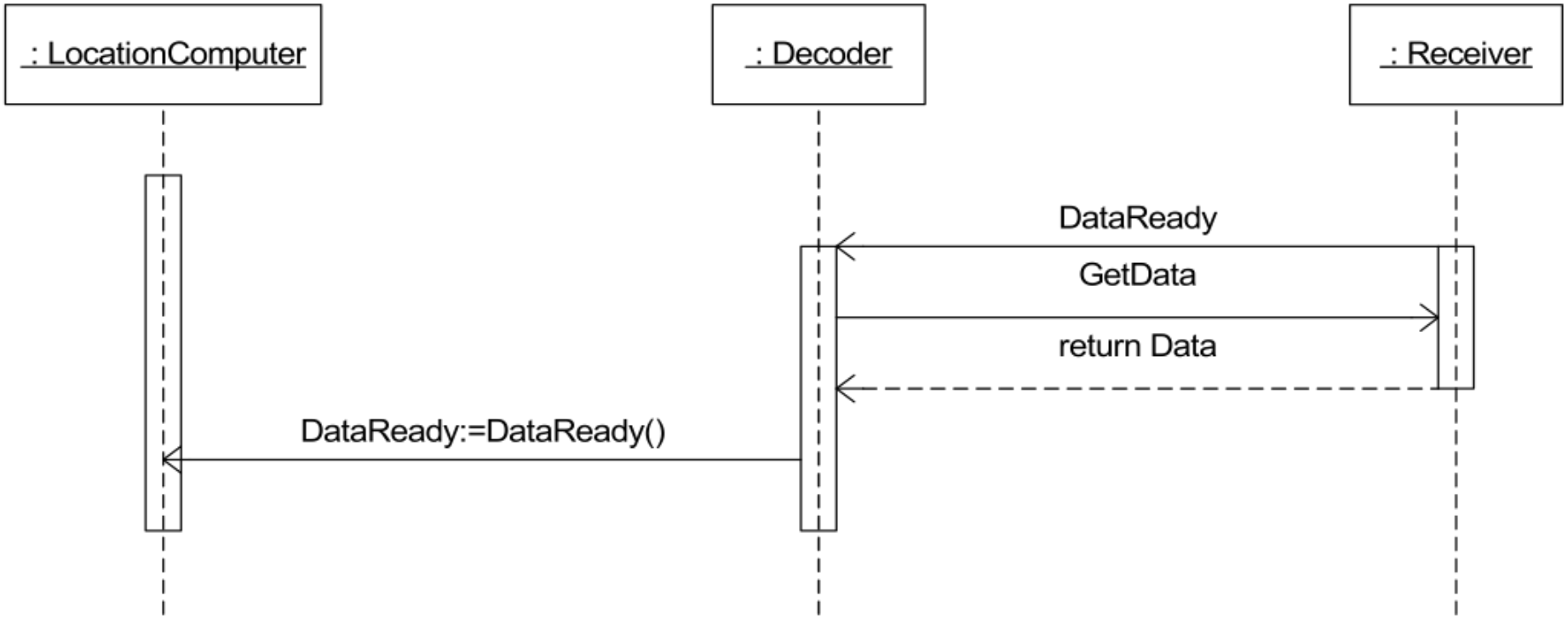
Modeling a (simplified) GPS device

- Sequence diagram: configuring decoders



Modeling a (simplified) GPS device

- Sequence diagram: interrupt driven architecture



- Many more sequence diagrams needed…

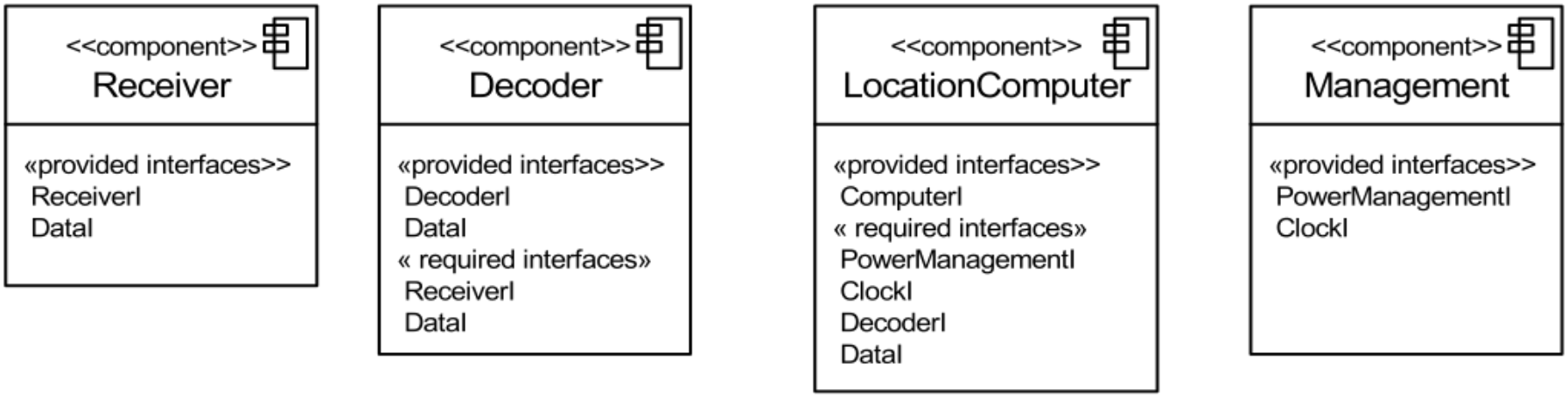
Modeling a (simplified) GPS device

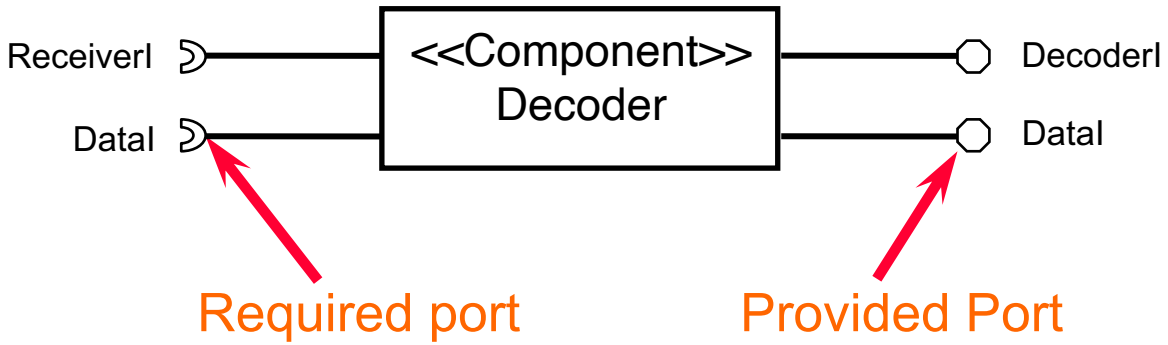
- Targeting multiple products with the same (business) model
  - Hand held autonomous device
  - Plug-in device for Smart Phone
  - Plug-in device for laptop (PCMCIA/USB)
  - May need to change part of the software after deployment

⇒We choose a component based delivery of the software

Modeling a (simplified) GPS device

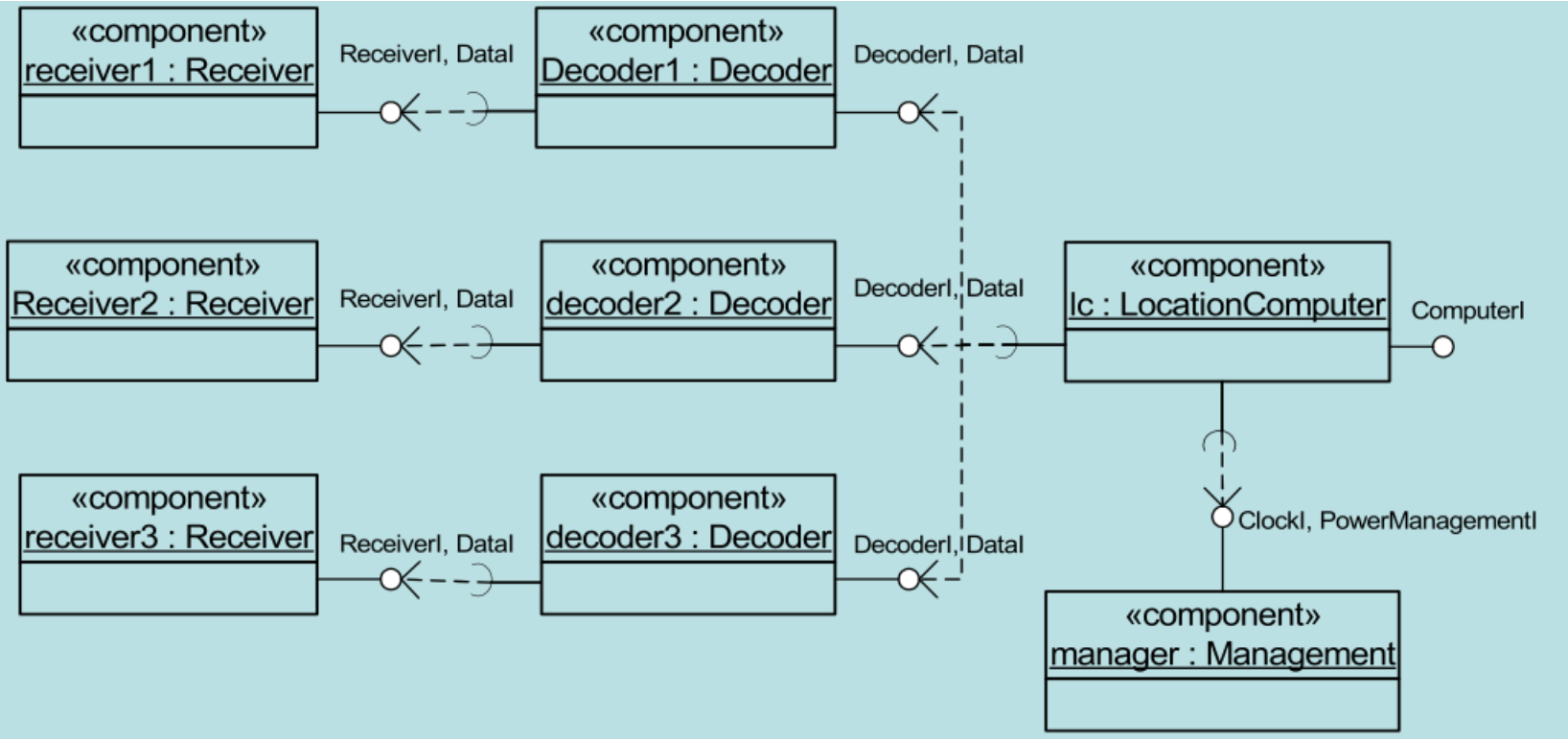
- Component diagram





Modeling a (simplified) GPS device

- Deployment diagram

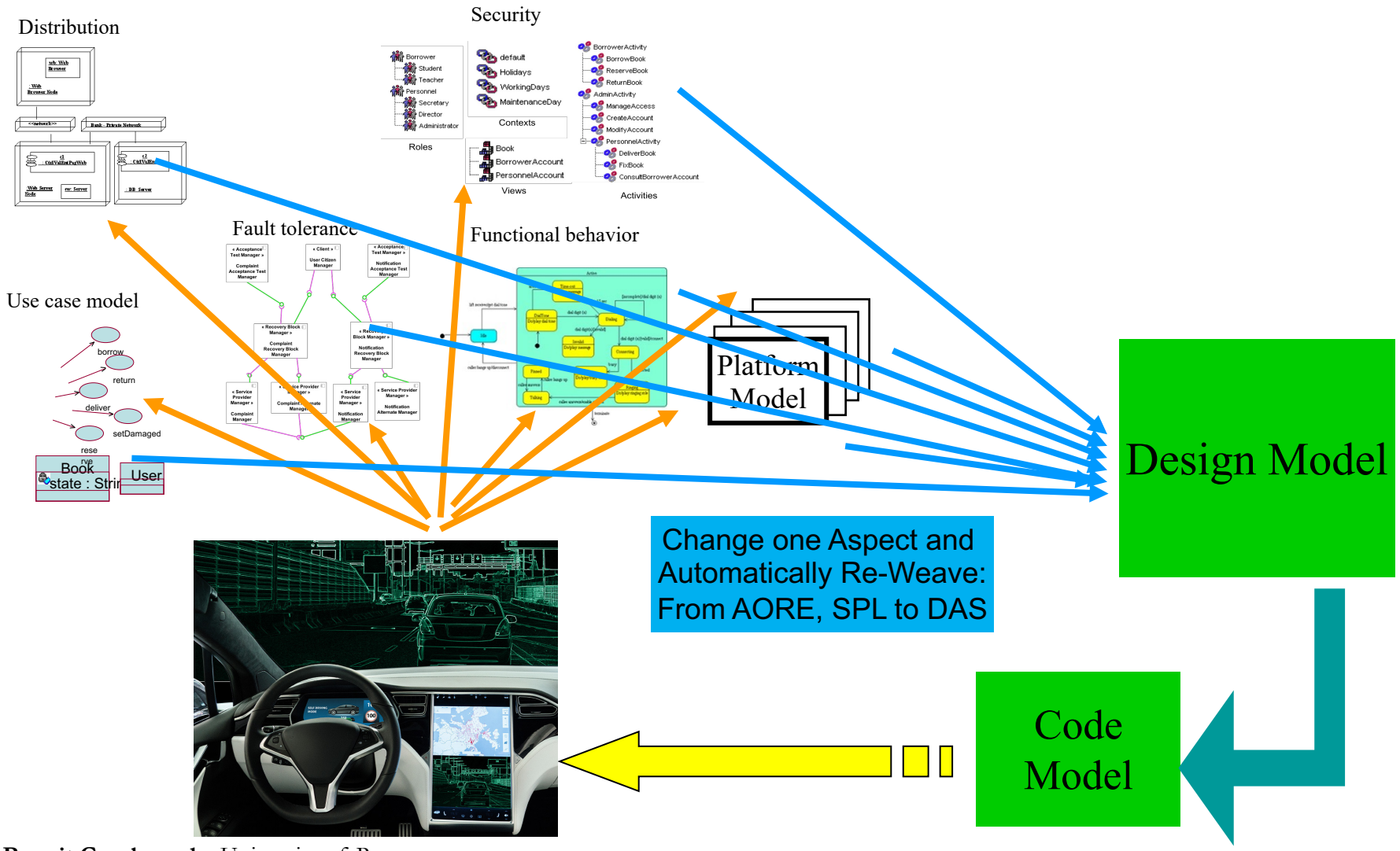


Model and Reality in Software

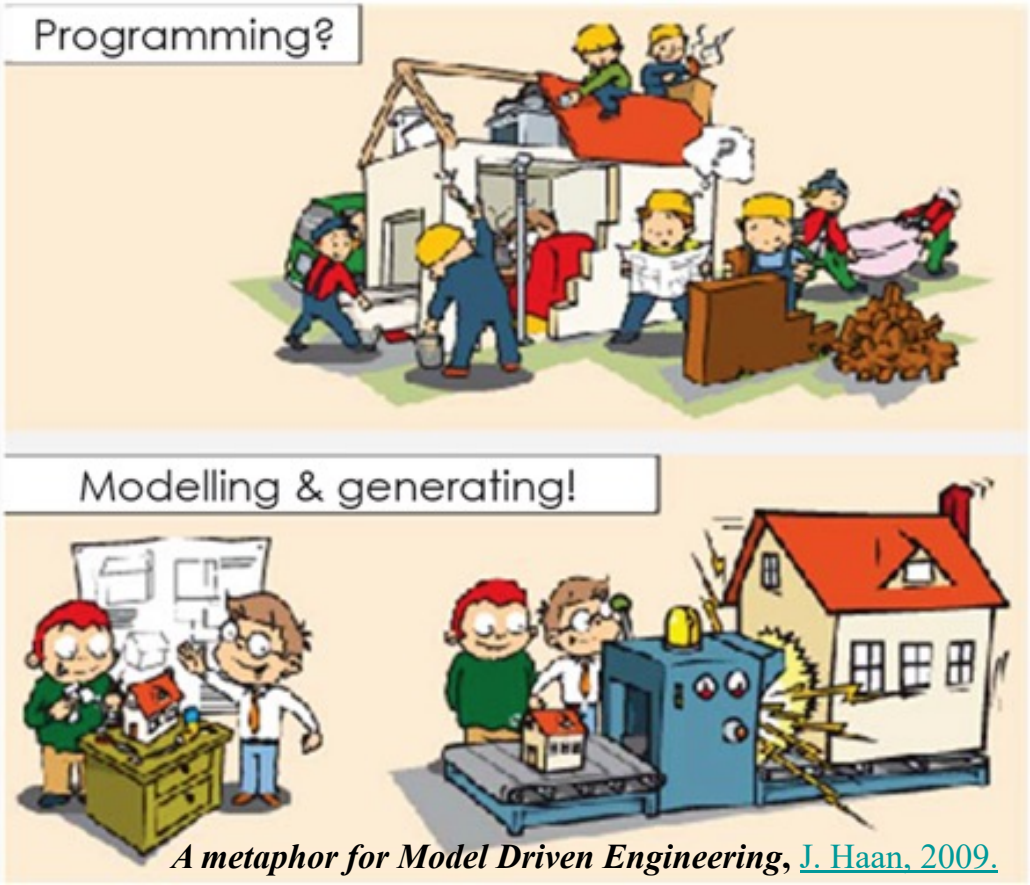
- **Sun Tse:** *"Do not take the map for the reality"*
- **William James:** *"The concept 'dog' does not bite"*
- **Magritte:**

- Software Models: from contemplative to productive

Towards Model Driven Engineering



Towards Model Driven Engineering

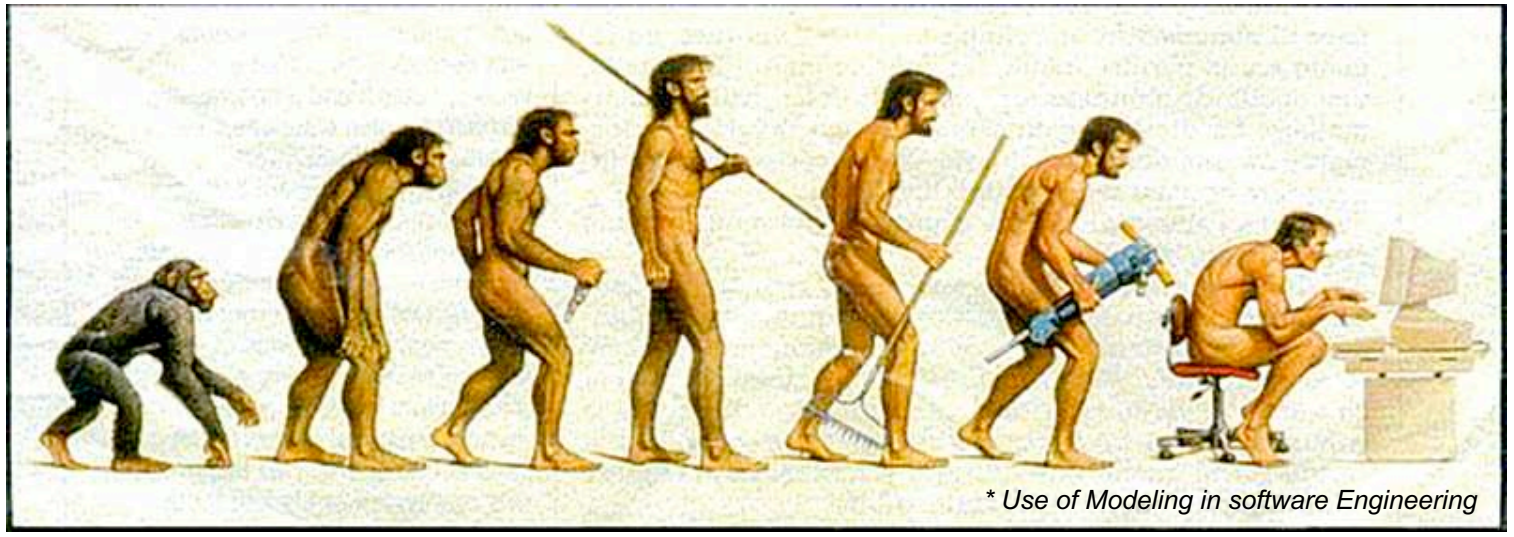


***A metaphor for Model Driven Engineering*, J. Haan, 2009.**

La modélisation: qu'elle utilisation ?

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

Adoption of Software Modeling



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.

Conclusion

- **Le métier d'ingénieur logiciel est complexe**: *principes*, *techniques*, *méthodes*, et *outils* pour décrire, implémenter, vérifier, gérer, et rendre opérationnel un système logiciel

- Réponse de l'ingénierie du logiciel par l'utilisation de la modélisation
  - séparation des préoccupations
  - montée en abstraction
  - agilité des développements

Key takeaways?