Génie logiciel

Philippe Dugerdil

19.09.2019

Objectifs du cours

1. Acquérir les principes du processus de développement - rôle des spécifications fonctionnelles et non fonctionnelles
2. Connaitre les processus SCRUM et DAD, et les fondements de RUP
3. Comprendre les principes de l'architecture logicielle et du test
    1. Acquérir les bases de l'architecture REST et des microservices
    2. Comprendre et mettre en œuvre les design patterns

Support: PDF sur Moodle

Evaluation

- Examen: 100%

- Obligatoire pour se présenter à l’examen: rendre 80% des TPs

Software Engineering

« The disciplined application of engineering, scientific and mathematical principles, methods and tools to the economical production of quality software »

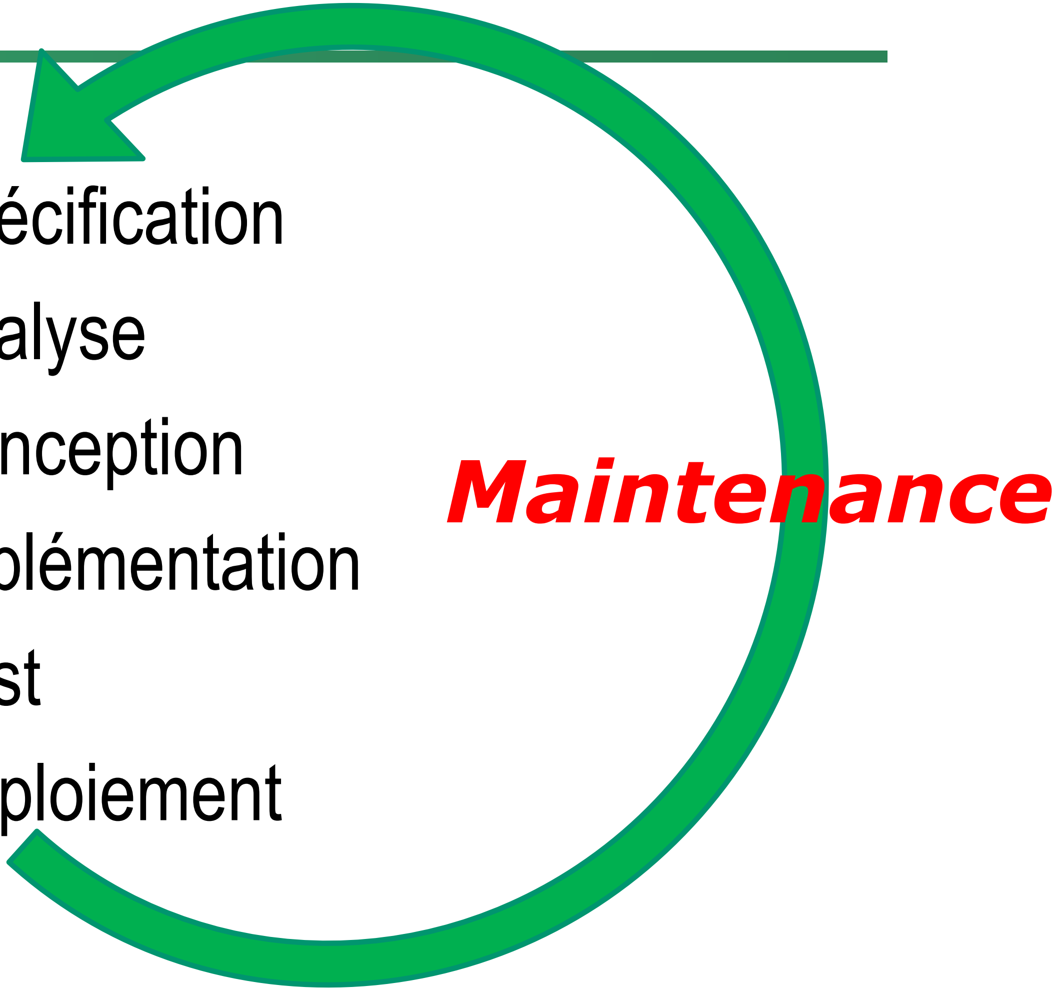
Watts S. Humphrey – Managing the Software Process, Addison-Wesley, 1989

Grossièrement, le GL doit répondre a trois questions…

- Quoi faire
- Comment faire (étapes, livrables)
- Comment le communiquer

En respectant des standards et principes de qualité

… sur de multiples itérations



- Spécification
- Analyse
- Conception
- Implémentation
- Test
- Déploiement

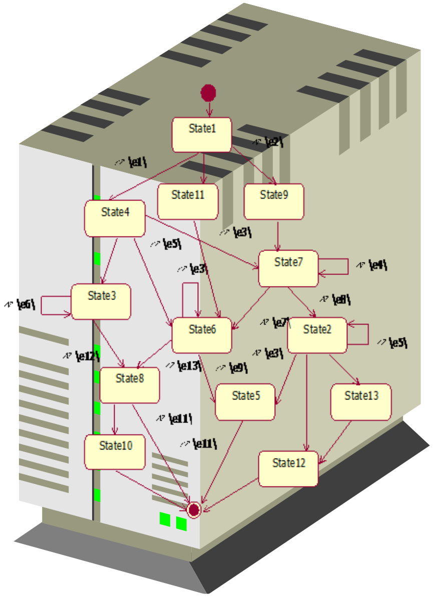
Cependant:

L’activité la plus coûteuse sur le cycle de vie du développement logiciel est:

La comprehension du logiciel

Estimation

Common estimation:
~50% of the maintenance cost



Ce n’est pas nouveau…

“ We all recognize that "understanding" a program is important, but most often it goes unmentioned as an explicit task in most programmer job or task descriptions. Why?

The process of understanding a piece of code is not an explicit deliverable in a programming project.”

[Corbi T.A. - Program understanding: Challenge for the 90s. IBM Systems Journal, 28(2):294-306, 1989]

Rappel : notre théorie naïve

“Understanding means being able to map what we observe to what we already know”

Then, in this context:

Understanding source code means being able to map the structure of the program, at each granularity level, to its purpose in the business domain (supposed to be known)

Why is software getting harder to grasp? Lehman's laws

1. Law of **continuing change**

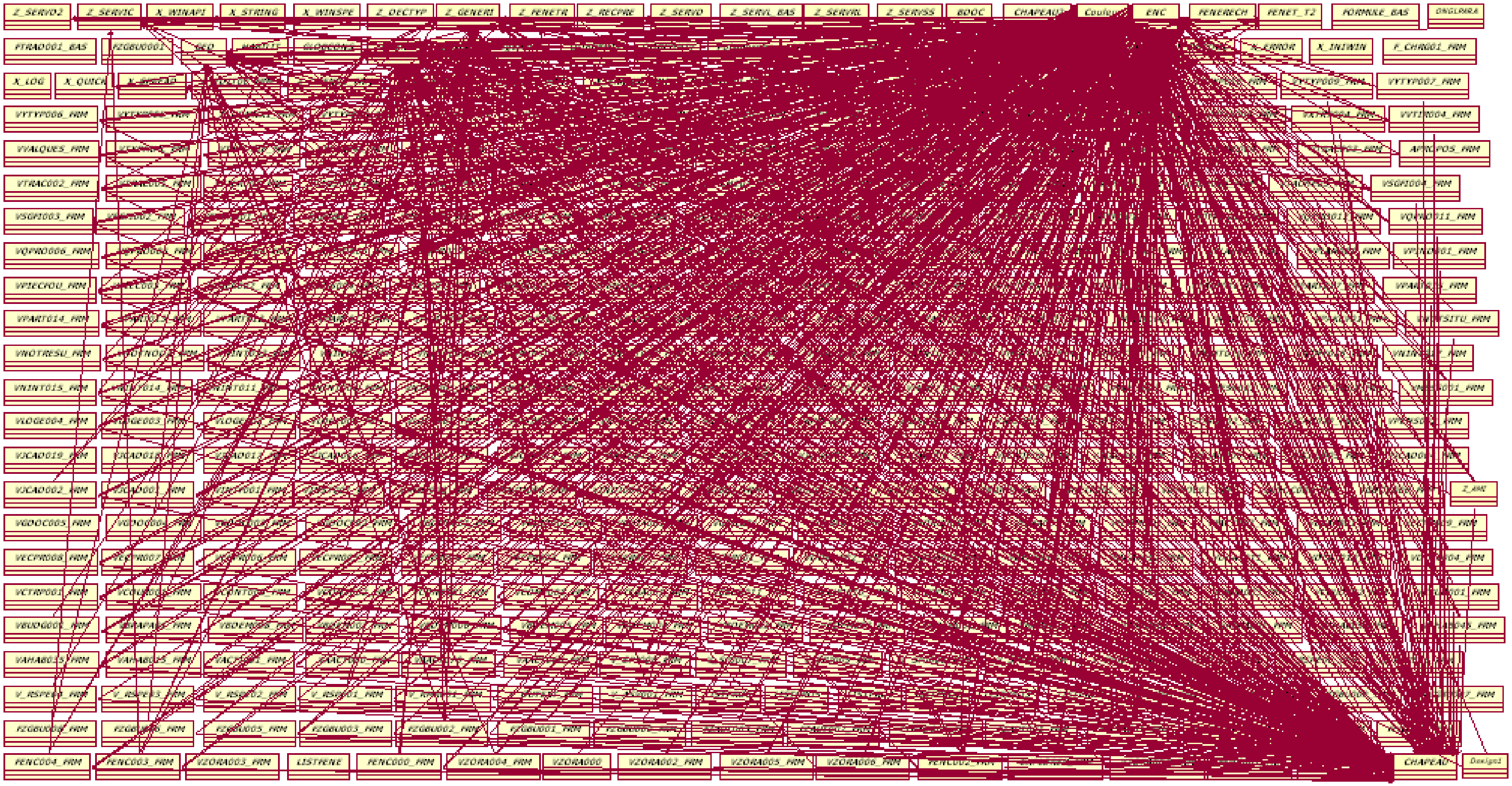
   A program that is used in a real-world environment must change or become progressively less useful in that environment.

2. Law of **increasing complexity**

   As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.

[Lehman M. – Programs, Life Cycles and Laws of Software Evolution. Proceedings of the IEEE 68(9), Sept 1980]

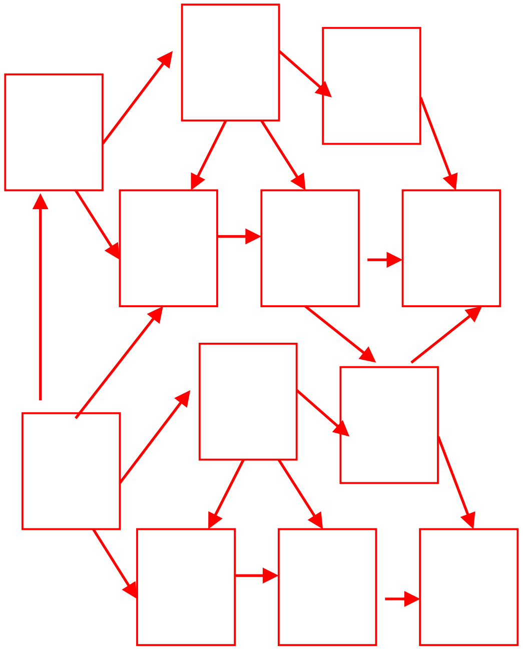
Si on ne fait rien…



Complexity ?
What are we speaking about ?

What is a complex system?

An informal definition



1. A system made up of a large number of parts that interact in a non simple way.
2. Given the properties of the parts of this system and the law of their interaction, it is not a trivial matter to infer the properties of the whole.

[Source: Simon H.A. - The architecture of complexity. In: The Sciences of the Artificial, MIT Press, 1969. (Reprinted in 1981)]

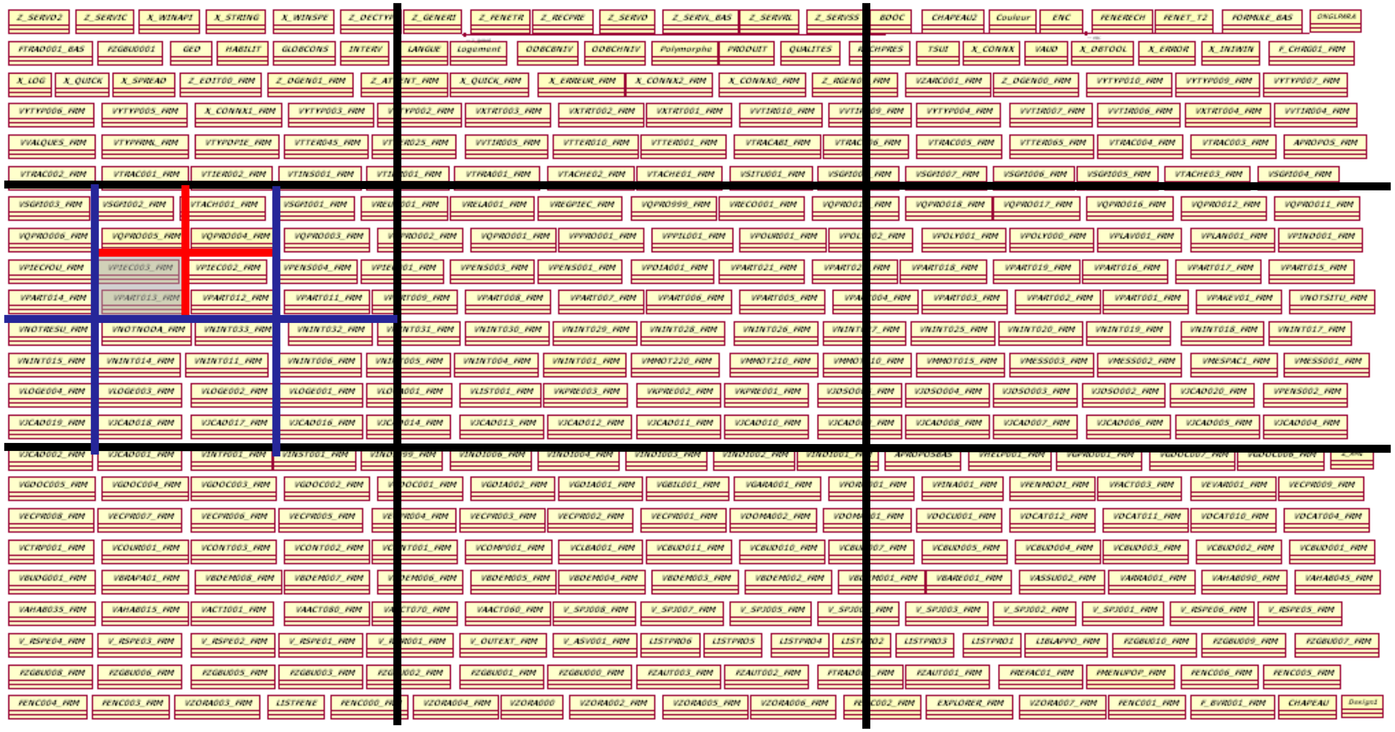
Théorie de H. Simon sur les systèmes complexes

The systems one could possibly understand must exhibit the "**near-decomposable treelike hierarchical**" structure:

- **Treelike hierarchical** : system that, at each level, is made up of interacting subsystems. Each subsystem is, in turn, composed of interacting sub-subsystem an so on.
- **Near decomposable**: the short run behavior of each of the component subsystems is approximately independent of the short run behavior of the other components.

[Source: Simon H.A. - The architecture of complexity. In: The Sciences of the Artificial, MIT Press, 1969. (Reprinted in 1981)]

Est-ce que n’importe quell decoupage hiérarchique est pertinent?



Why not ?

Near decomposable?

Building treelike near-decomposable software systems

- For the near-decomposability property to hold, better to build systems having such property in mind
- But…this meets some fundamental principles of Software Engineering that are well known:
  - Weak coupling
  - High cohesion
  - Information Hiding (Encapsulation)

Coupling

Coupling: the degree to which components depend on one another. It is increased when the data exchanged between components becomes larger or more complex. [RUP / IBM Rational]

- This metric can be considered a measurement of the **external** functional dependency of the component
  - If coupling is low, the component is approximately independent from the other components at the same level to deliver its functionality.
  - The system is then near-decomposable !

Cohesion

Cohesion is the measure of the strength of functional relatedness of elements within a module.

[Meilir Page-Jones - Practical Guide to Structured System Design. Prentice Hall, 1980]

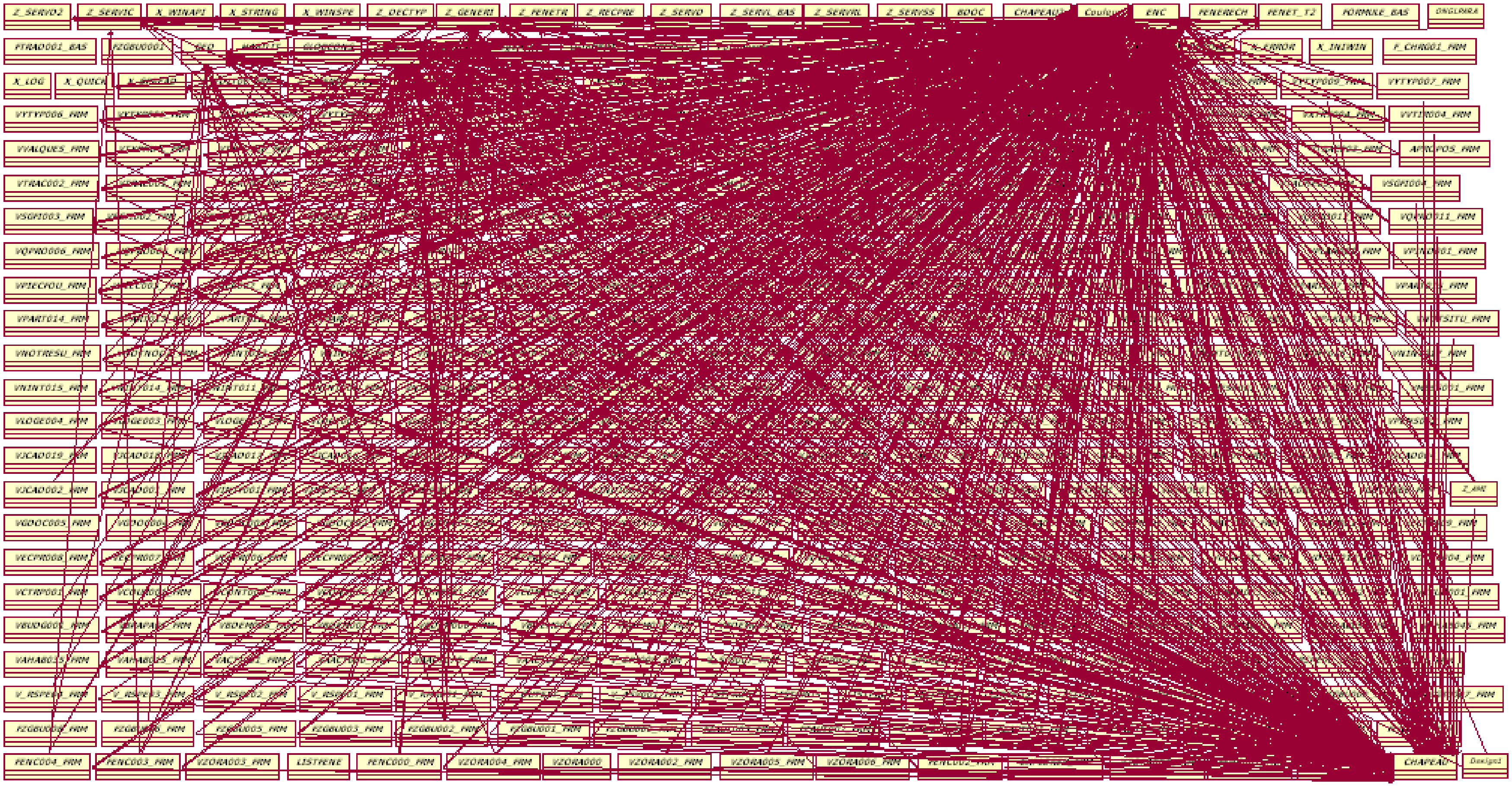
- This metric measures the **internal** functional dependency of the parts of the component.
  - If cohesion is high: its subparts are all involved in the delivery of the functionality of the component.
  - There is no “noise” in the structure of the component: all its subparts have a role in the delivery of the functionality.

Information hiding

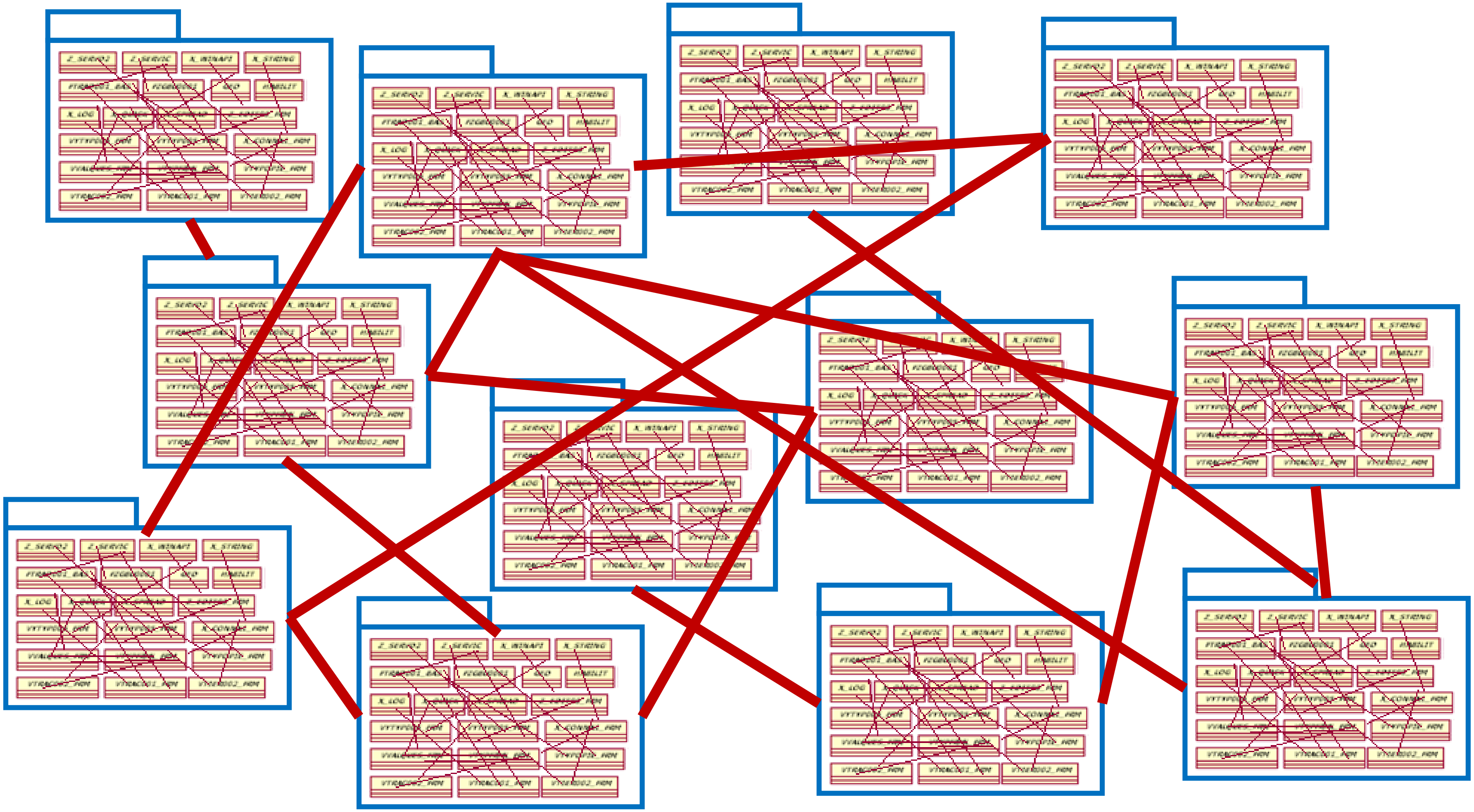
This means: “Hiding the knowledge about design decisions in a module and choosing the interface so that it reveals as little as possible about its inner working”.

Adapted from [Parnas D.L. - On the Criteria To Be Used in Decomposing Systems into Modules. CACM 15(12), 1972]

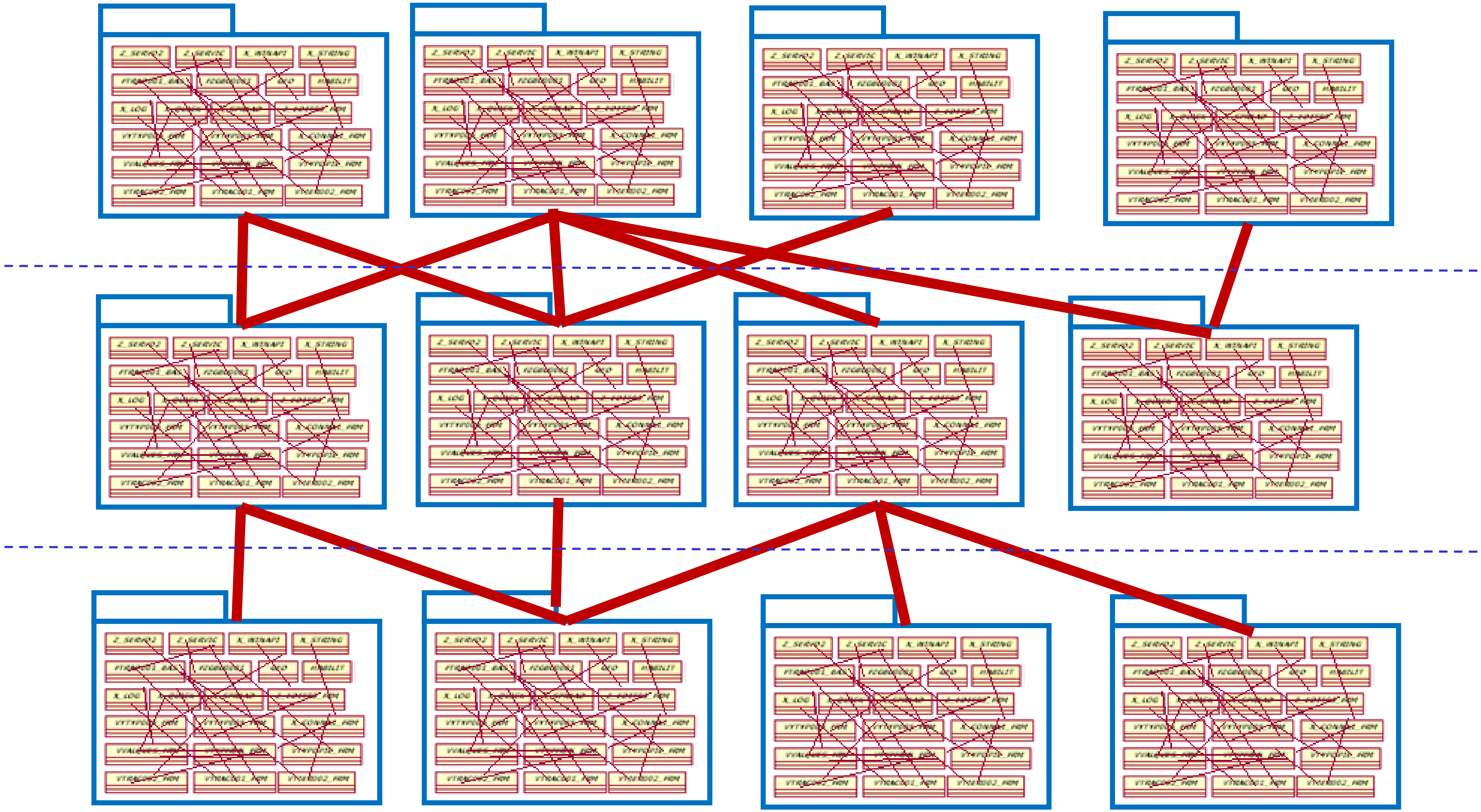
Improving the structure…and understanding



First step…



Second step. Is this enough?

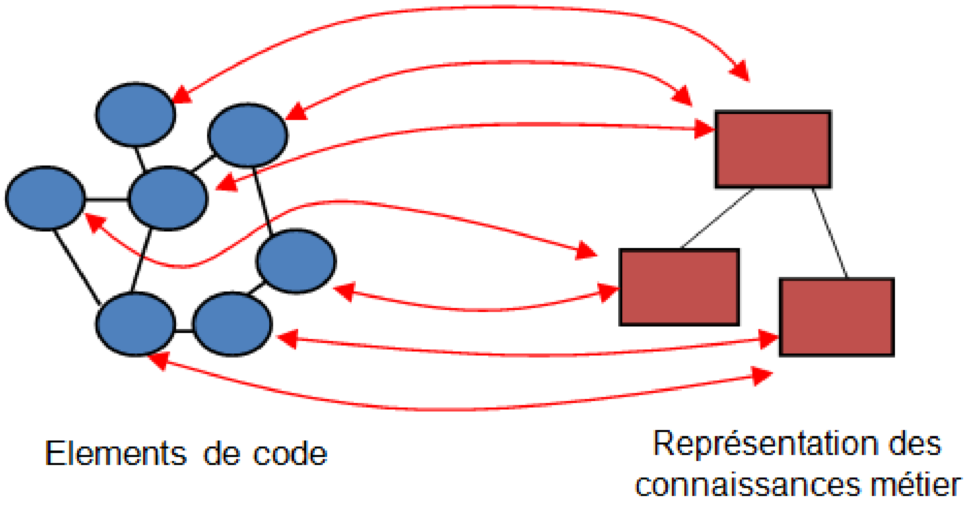


Why bother about components?

- To improve maintainability
- To make software intelligible

Lien avec notre théorie de la compréhension

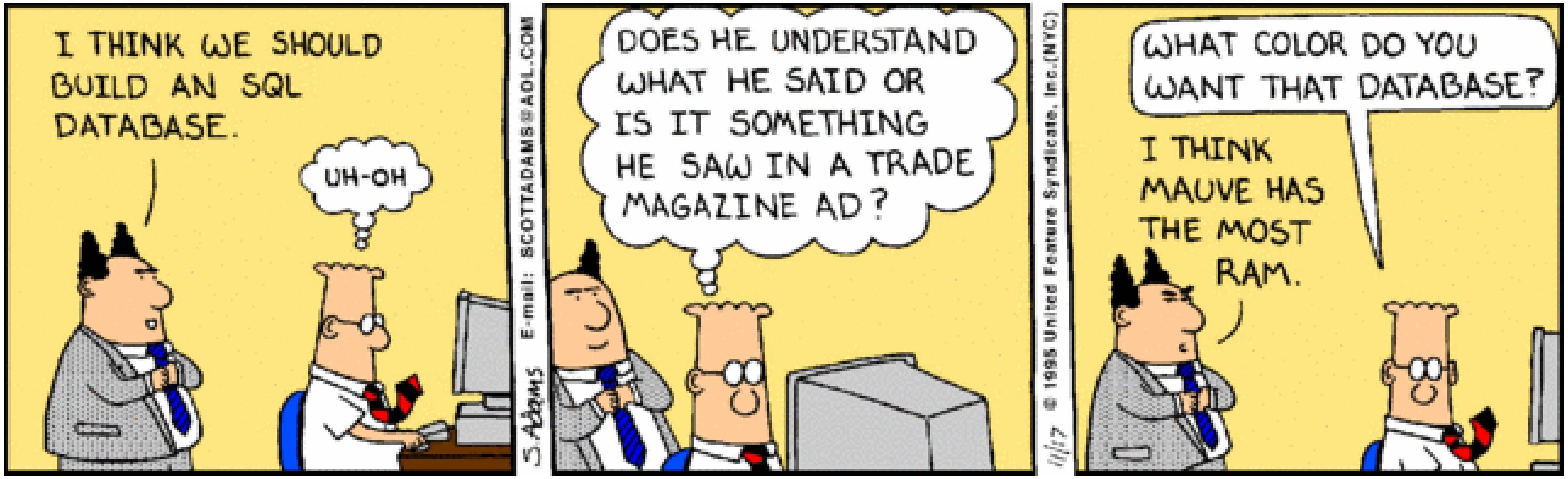
Permettre une mise en correspondance des concepts métier (connu) avec le code (inconnu)



Question

Pourquoi faut-il que le logiciel soit quasi décomposable hiérarchiquement?

Avant tout, le mandant doit exprimer ses besoins en termes métier


I THINK WE SHOULD BUILD AN SQL DATABASE.
UH-OH
E-mail: SCOTTADAMS@AOL.COM
S. Adams
DOES HE UNDERSTAND WHAT HE SAID OR IS IT SOMETHING HE SAW IN A TRADE MAGAZINE AD?
© 1998 United Feature Syndicate, Inc.(NYC)
11/17
WHAT COLOR DO YOU WANT THAT DATABASE?
I THINK MAUVE HAS THE MOST RAM.

Rappel: facteurs de risque sur les projets (1995)

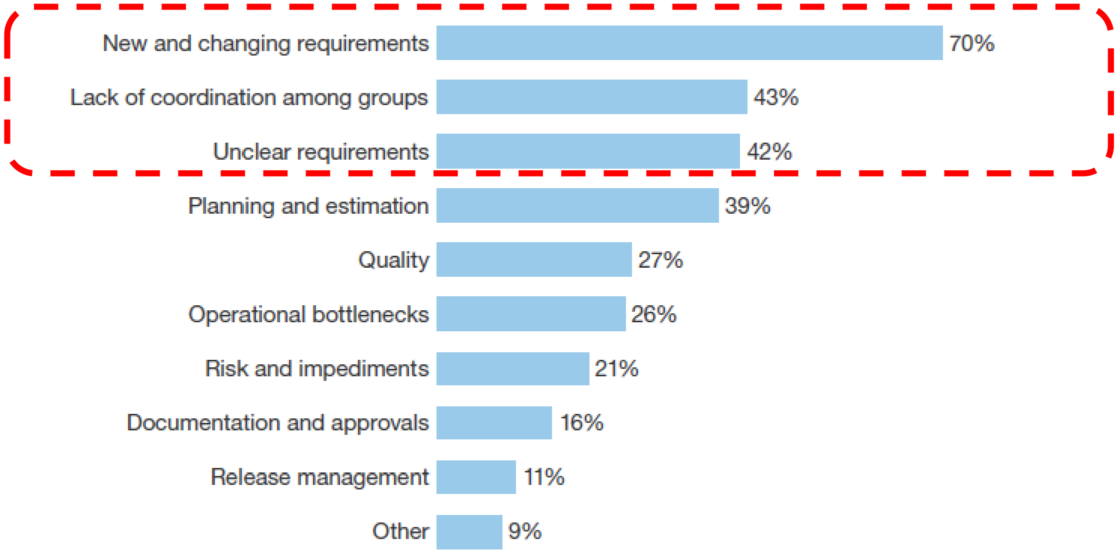
% of Responses		
1. Lack of User Input	12.8%	36.9 %
2. Incomplete Requirements & Specifications	12.3%	
3. Changing Requirements & Specifications	11.8%	
4. Lack of Executive Support	7.5%	
5. Technology Incompetence	7.0%	
6. Lack of Resources	6.4%	
7. Unrealistic Expectations	5.9%	
8. Unclear Objectives	5.3%	
9. Unrealistic Time Frames	4.3%	
10. New Technology	3.7%	
Other	23.0%	

Sample size : 365 respondents and represented 8,380 applications

Source : The Standish Group, 1995

Et beaucoup plus récemment (2014)

Figure 1 Requirements Are A Consistent Source Of Delivery Problems



"When you cannot deliver on time, what are the usual points of failure?"

Base: 115 software development professionals
(multiple responses accepted)

Source: March 2013 Application Life-Cycle Management Online Survey

115852 Source: Forrester Research, Inc.

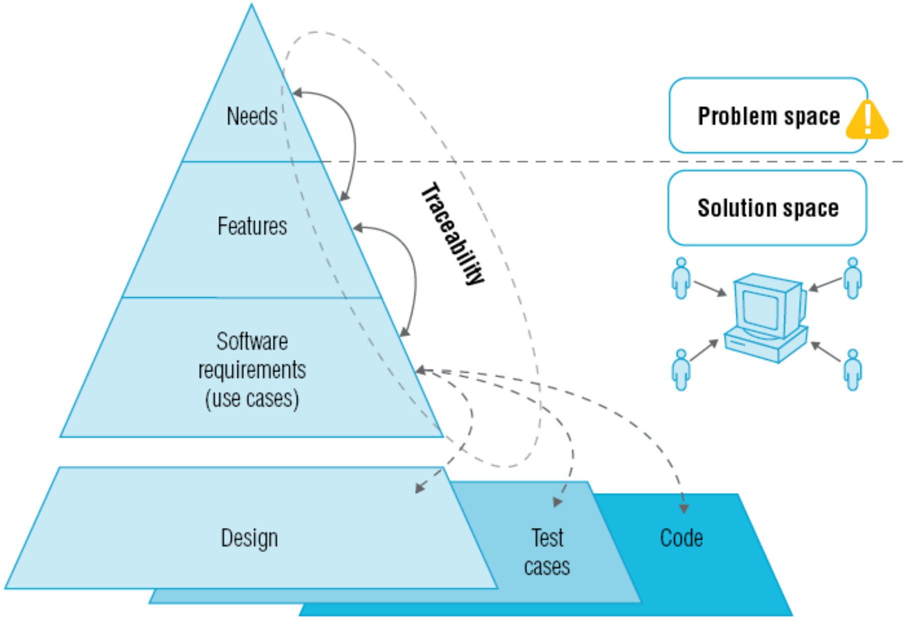
Source: Kurt Bittner - Brief: Software Requirements Practices Are Ripe For Disruption. Forrester research, April 25, 2014

Ne pas se tromper de niveau

- Besoin (**need**) : problème métier, personnel ou opérationnel justifiant la mise en œuvre d'une solution informatique.

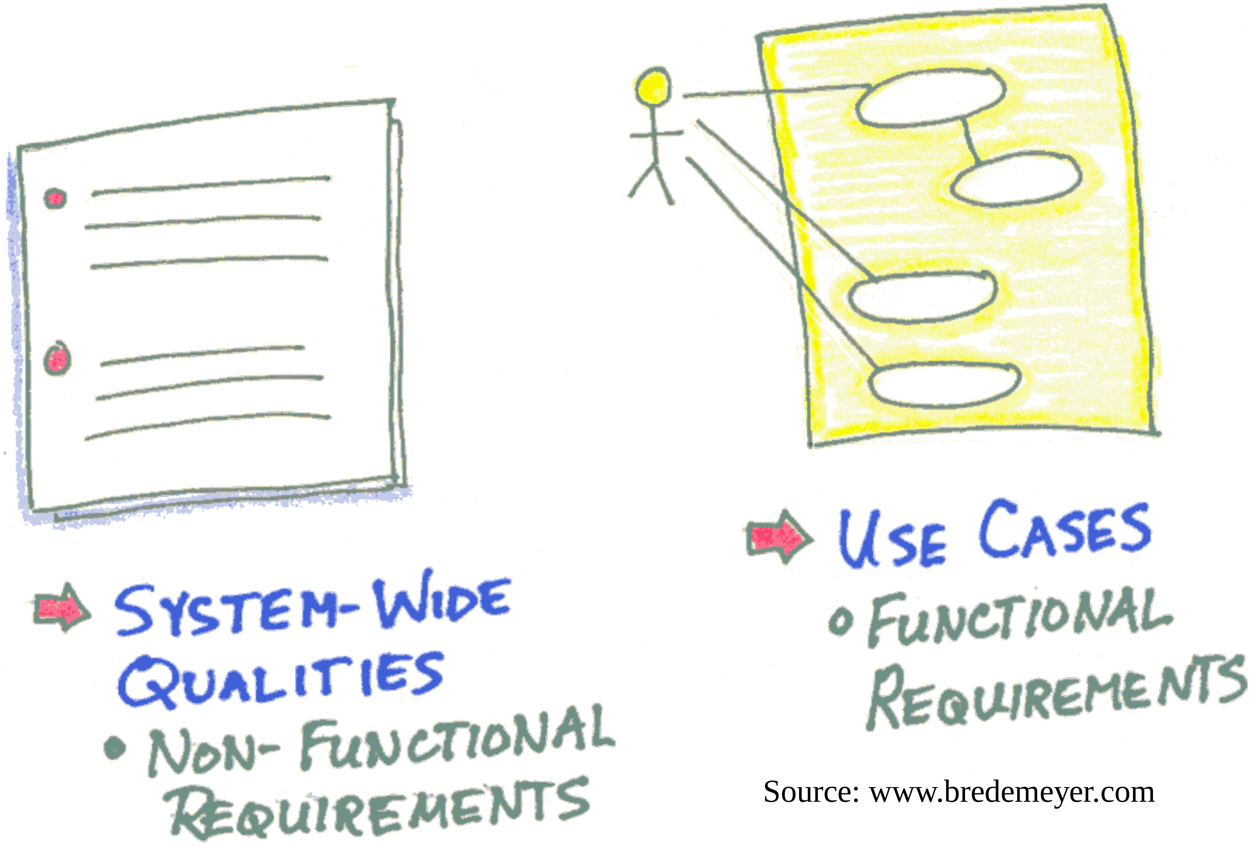
- Caractéristique de la solution (**feature**) : service que le système va offrir pour répondre au besoin.

- Spécification informatique (**requirement**) : spécification du système permettant de résoudre le problème en offrant les services identifiés (features).



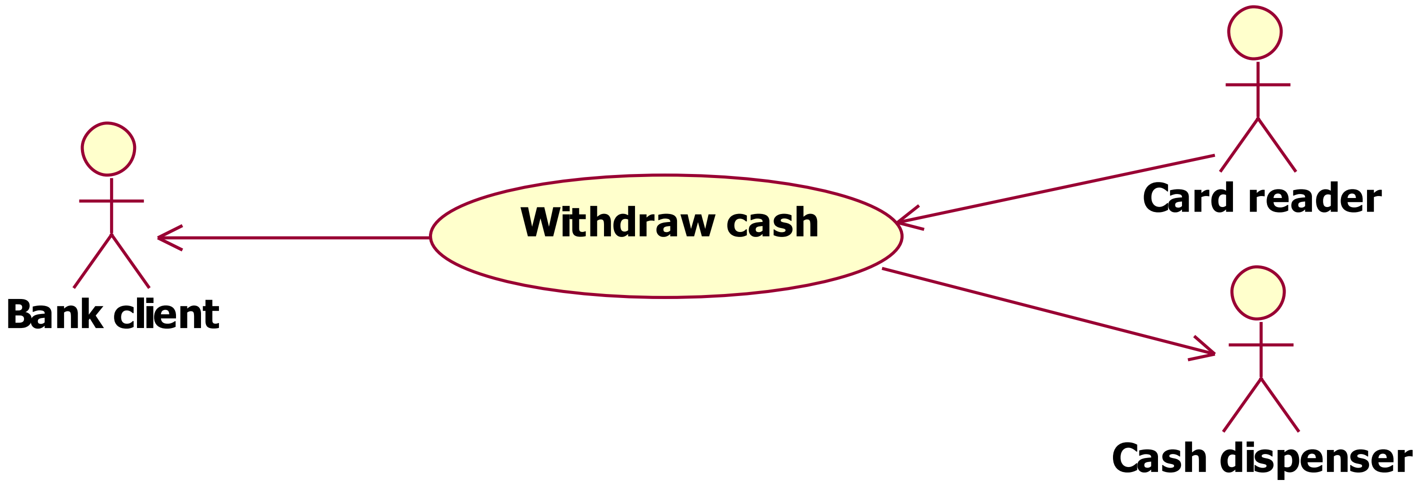
Dean Leffingwell - Features, Use Cases, Requirements, Oh My!
Rational Software 2001

Functional and non functional specs



Source: www.bredemeyer.com

Functional specs: use-cases (rappel)



Use-case : withdraw cash

Name: Withdraw cash
Actors: primary: bank client, secondary: card reader, cash dispenser
Trigger: the client insert a card in the card reader
Basic flow:

1. The card reader tells the system that a card is inserted
2. The system displays: please enter PIN code
3. The client enters his PIN code
4. The system asks the card reader to check the PIN code
5. The card reader validate the code
6. The system displays the menu of possible operations
7. The client select: withdraw cash without receipt
8. The system displays: please enter the amount of cash to withdraw
9. The client enters the amount
10. The system asks the card reader to eject the card
11. The system asks the cash dispenser to output the bank notes

Non-functional Requirements

(NFR, (Quality attributes)

Any global property of a system that is expressed in non-functional terms. It is « orthogonal » to the functionality. Examples:

- Performance
- Availability
- Modifiability
- Testability
- Reliability
- Usability
- Portability

Often QA are categorized in two sets:
Visible to the end user (availability, performance,…)
Invisible to the end user (modifiability, portability,…)

But… invisible QA can have visible side-effects !

Role of NFR in software design

NFR drive the architecture of the solution

- Whatever the architecture the FR may be met
- However, NFR can only be met through the proper system architecture

Specifying quality attributes

Since the quality attributes are important to design the architecture of the system, they must be expressed early.

However, what do « high maintainability » or « high performance » really mean ?

- what is « maintainability »
- what is « high »

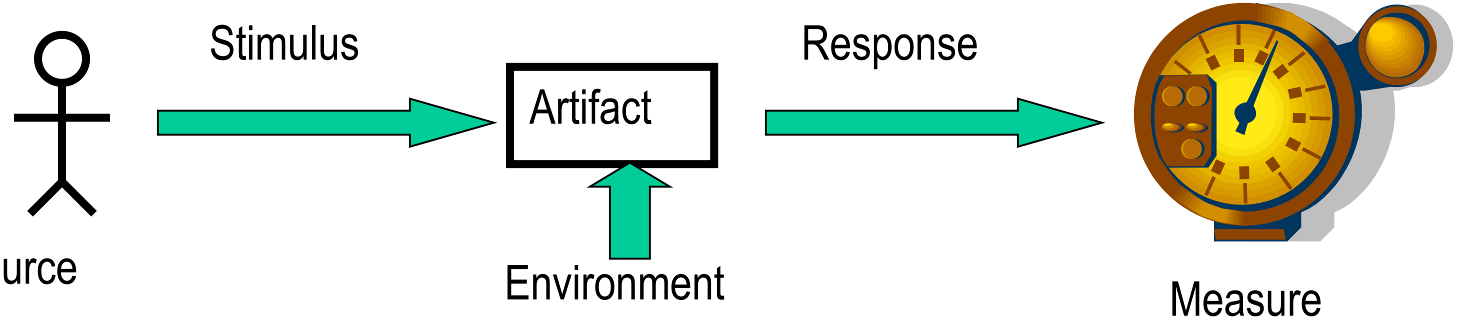
Moreover the QA’s are heterogeneous…now to specify them ?

Specifying NFR: the SEI framework:
(Scenario approach to QA)

The QA precise semantics and measurement value are highly contextual

=> The scenario approach to quantify QA's

SEI Framework:



Source: Bass L., Clements P., Kazman R. - Software Architecture in Practice. 3rd Edition. Adison-Wesley 2013.

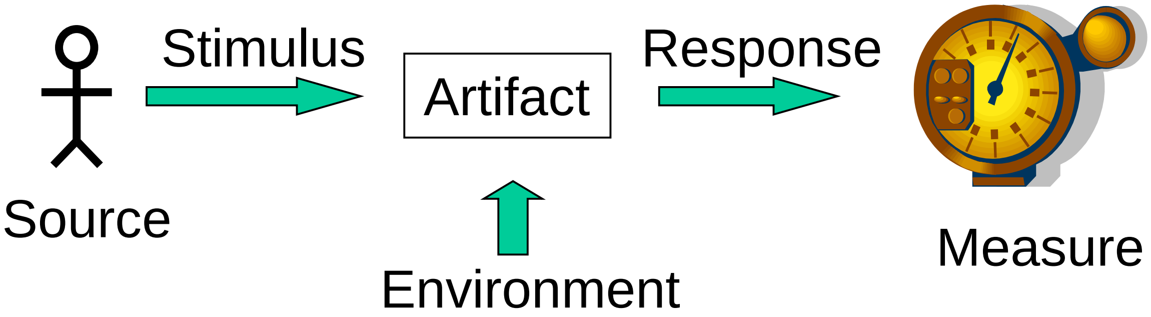
Example QA: modifiability

- What can change?
  - The functions
  - The platforms
  - The environment
  - The qualities of the system (performance)
- When is the change made and who makes it?
- Are the requirements the same for all subsystems ?

Example: modifiability

« The system must be easily modified »

- Rephrase this specification with the framework:



Some clues…

- Source of the change :
  - User at run time
  - System administrator at run time
  - Parameterization engineer at run time
  - Developer at development time
  - Developer at system configuration time
  - System engineer at installation time
  - …

Example: modifiability scenario

QA: modifiability

Source: developer

Stimulus: whishes to add a functionality

Artifact: user interface

Environment: development time

Response: locate places where changes must be made, change made without disturbing the rest of the system, test the system, deploy modification

Response measure: time for the modification to be operational.

Source: Bass L., Clements ., Kazman R. – Software Architecture in Practice. Second Edition Adison-Wesley Inc. 2013.

Example QA: performance

- How long does it take for the system to respond to an event (latency) ?
- Source of performance problems
  - Availability of required resources
- Are the requirements the same for all subsystems ?

Performance scenario example

QA: performance

Source: user

Stimulus: input some information through the screen

Artifact: whole system

Environment: normal operation

Response: any information that is output from the system due to the stimulus

Response measure: time to get the output.

Example QA: availability

The degree to which a system is operable

- Source of availability problems
  - Faults in the system or subsystems (such as bugs)
  - Unknown operation conditions (such as bad data)

- Are the requirements the same for all subsystems ?

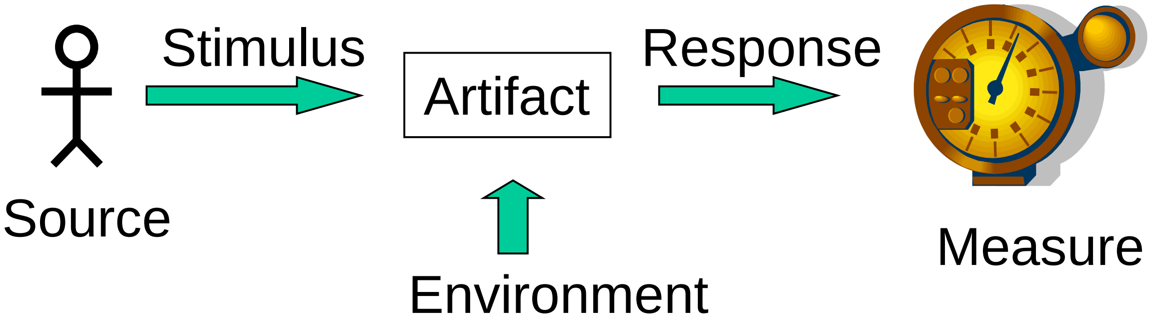
Example: availability

«The system must be available all the time»

– What does «all the time» mean ?

– For which user / stakeholder exaclty (all of them) ?

– How to measure it ?



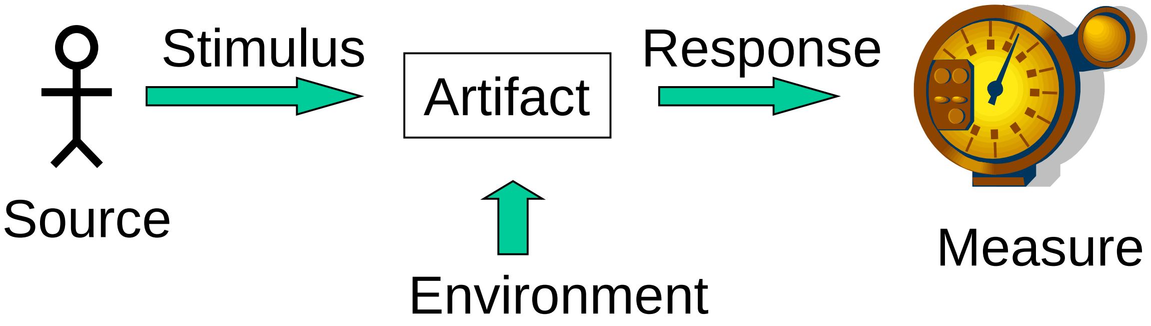
Some clues…

- Available for :
  - Standard user in normal mode
  - Standard user in failure recovery mode
  - System administrator in normal mode or recovery mode
  - Database administrator in system's configuration mode
  - ….

Example : utilisability

«The system must be user-friendly»

- What does it mean?
- How to quantify it?



Source

Stimulus

Artifact

Environment

Response

Measure

Besoin d’un processus

Une vieille recommandation….

“The most significant improvements in quality and productivity come from improvements to the process of systems development, this can occur only when everyone is following the same process in a standardized fashion.”

[Yourdon Ed. - Decline and Fall of the American Programmer, Prentice Hall, 1999]

Comment organiser les taches?

Traditionnellement, on définit

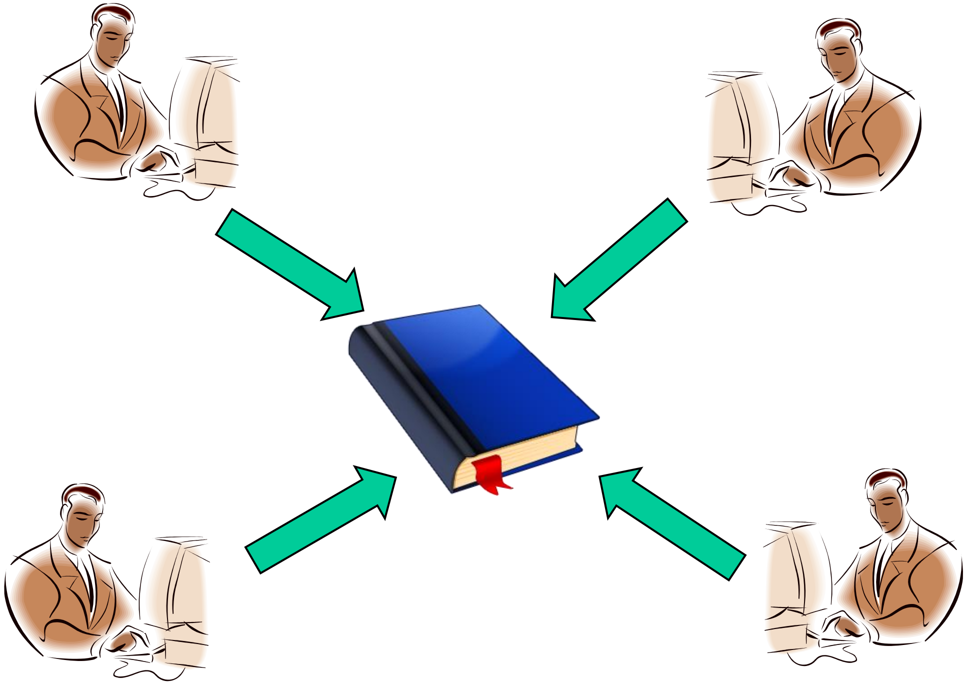
- Les rôles des personnes
- Les outils
- La granularité des tâches et leur dépendances
- Les livrables des taches (modèles, doc. et code)

Un exemple d’organisation de projet

- Métaphore : écriture d’un ouvrage sur commande
  - Sur un thème donné
  - Dans un domaine que vous connaissez mal

- Contrainte
  - Délais et couts fixés à l’avance, demandant le travail de plusieurs personnes à la fois.

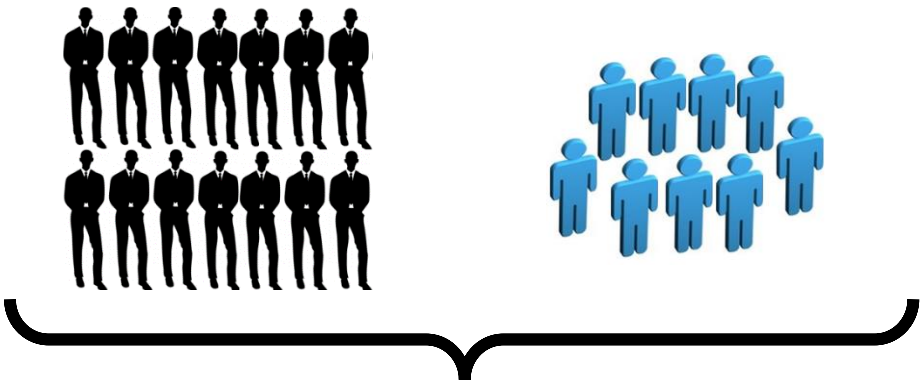
Comment s'y prendre?



Résumé

- Organisation du «développement» de l’ouvrage?
- Specs
  - Fonctionnelles ?
  - Non fonctionnelles ?
- Qui définit le succès du projet ?

Launching a project: need of a Vision



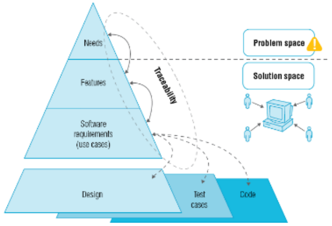
Vision

Common understanding of the issues and the potential solutions

Le document de vision (origine: RUP)

Lien avec le business

- Problème à résoudre
- Produit envisagé
- Features et lien avec les besoins
- Contraintes non fonctionnelles



Intervenants

- Intervenants et responsabilités
- Utilisateurs, nombre et responsabilités
- Besoins essentiels des intervenants

Contraintes

- Segments de marché
- Produits concurrents
- Environnement technique.
- Contraintes à la réalisation
- Marché (coût, prix, politique de licence, installation)

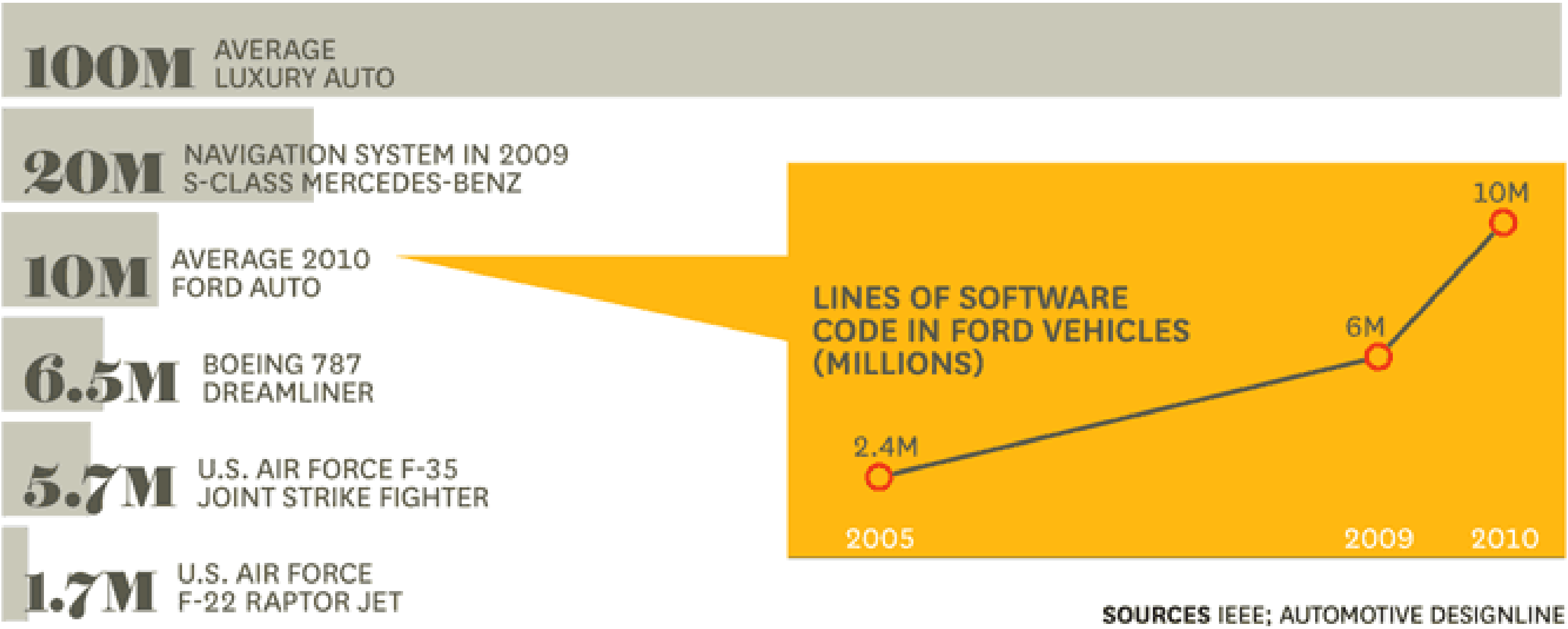
Quick look to the Vision Document

(~Etude d’opportunité)

Taille des logiciels aujourd’hui

Pour fixer les idées…

Exemple : software embarqué dans une voiture (Nb ligne de code)



SOURCES IEEE; AUTOMOTIVE DESIGNLINE

What does 1 million SLOC mean?

- A metaphor: books in a library
  - Average book: 300 pages, 50 lines / pages : 15’000 lines
  - Thickness: ~3 cm
- 1 million lines ~ 65 books , ~2m on bookshelf

Changing the behavior of the software

Software maintenance: changing the behavior of a program by modifying some lines of code.

Challenge :

- What to change to get the intended behavior ?
- How to avoid side effects ?

But…could we change a paragraph without *understanding* the story?

Try to rewrite a paragraph in a medical surgery book !