

Rétro-ingénierie & Réingénierie

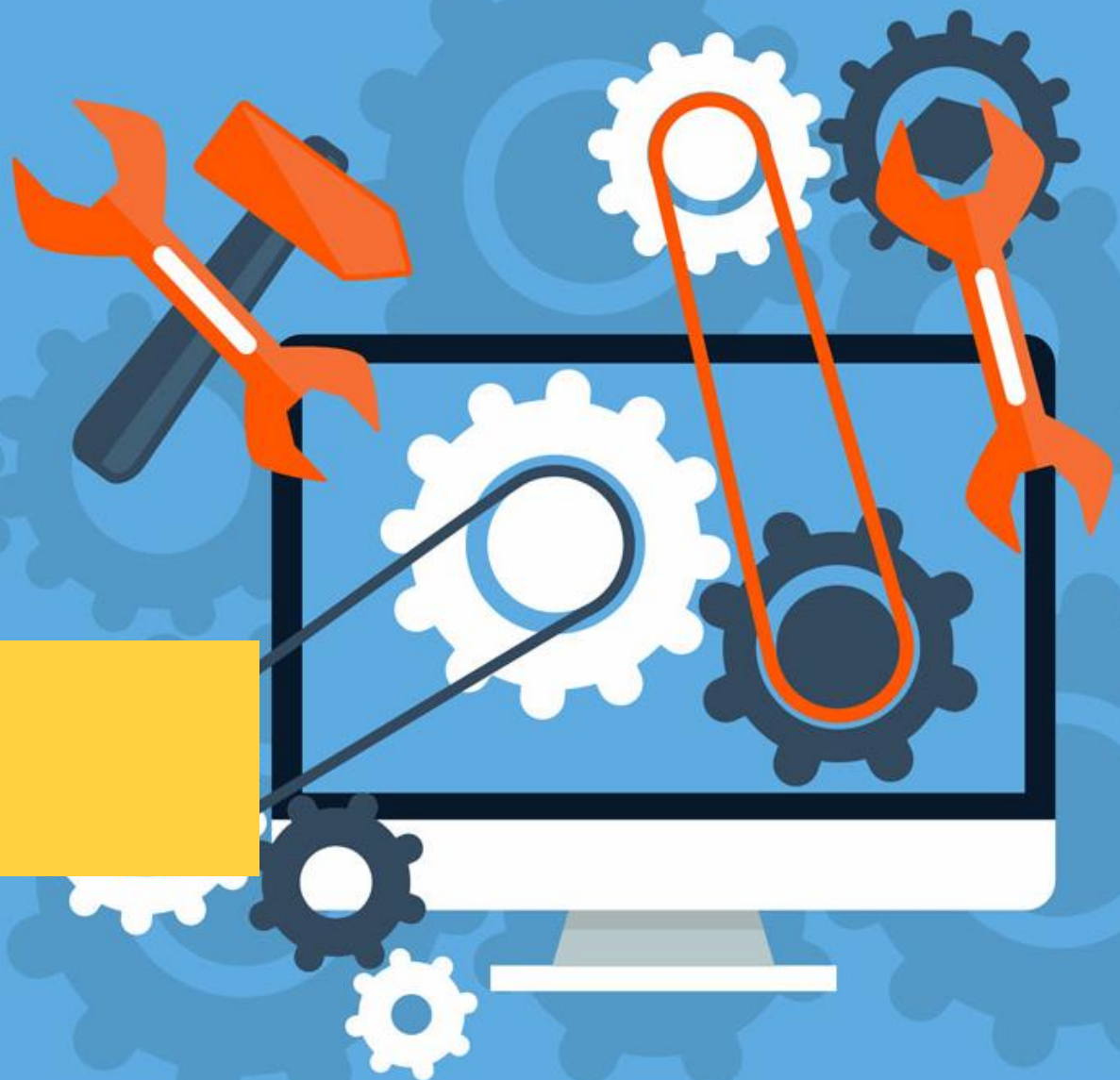
Pascal Zaragoza
MAREL - LIRMM

pascal.zaragoza
@berger-levrault.com
@lirmm.fr



Introduction

Mise-en-scène



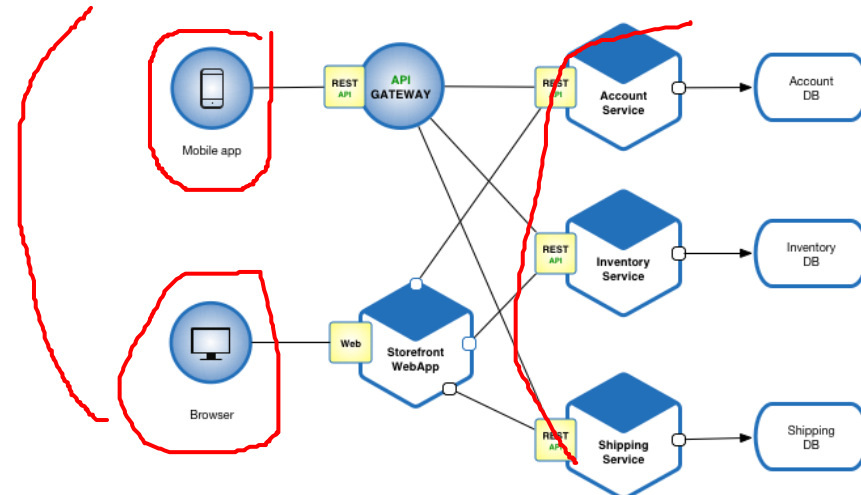
Introduction : L'architecture à base de microservices

- Approche à base de microservices : « **Componentization via Services** »

Approche facilitant l'évolution et la maintenance de logiciels selon un idée de **composabilité et interactions entre services indépendants**.



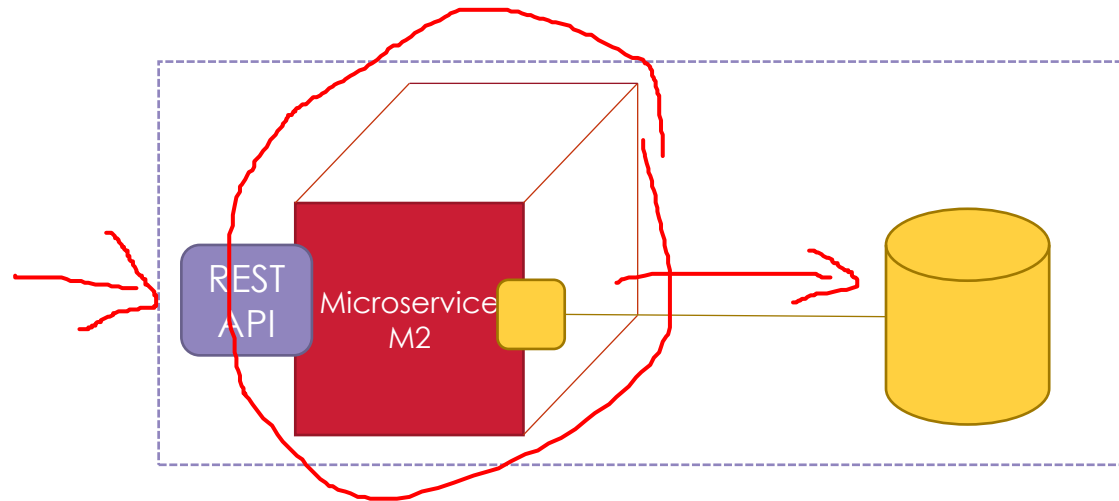
JPetClinic : application web à base de microservices



Application mobile + web interagissant avec un backend.

- C'est quoi un microservice ?

Une petite application (ou service) indépendante qui tourne sur son propre processus, utilisant des mécanismes légers, avec un déploiement automatisé et une autonomie de données. – Lewis and Fowler



Introduction : Les Avantages de l'architecture



01

Développement Décentralisé / Agile

Chaque microservice est son propre projet avec sa propre équipe. Cela facilite le développe indépendant.



02

Optimization de ressources

Les applications sont indépendantes ce qui permet de les lancer en fonction de la charge sur le serveur.



03

Résilience

Un couplage faible entre les microservices augmente la résilience contre les pannes de serveur.



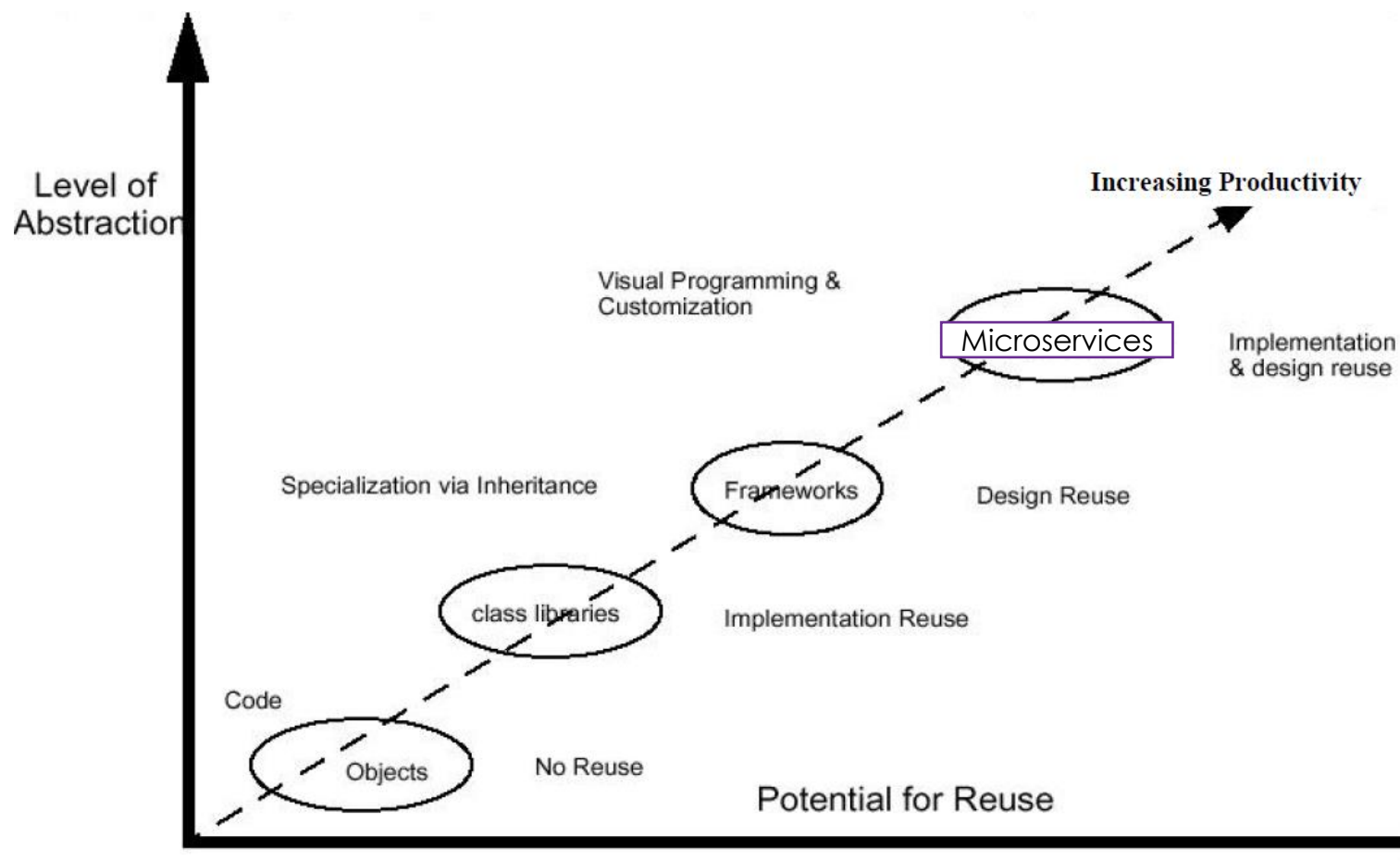
04

Autonomie de données

Chaque microservice est associé à sa propre base de données. Cela rend le service plus rapide.



Introduction : Réutilisation



- Les systèmes existants
 1. Besoin de réutilisation
 - Faciliter et diminuer les coûts des nouveaux projets par la capitalisation de l'existant.
 2. Besoin de faciliter la maintenance
 - Absence de vues abstraites, donc difficulté de compréhension
 - Conception « petits grains » (granularité aux classes) donc difficulté de séparation de préoccupations.
 - Phénomène d'érosion
 - Est concernée la documentation de la réalisation.
 - Est moins concernée

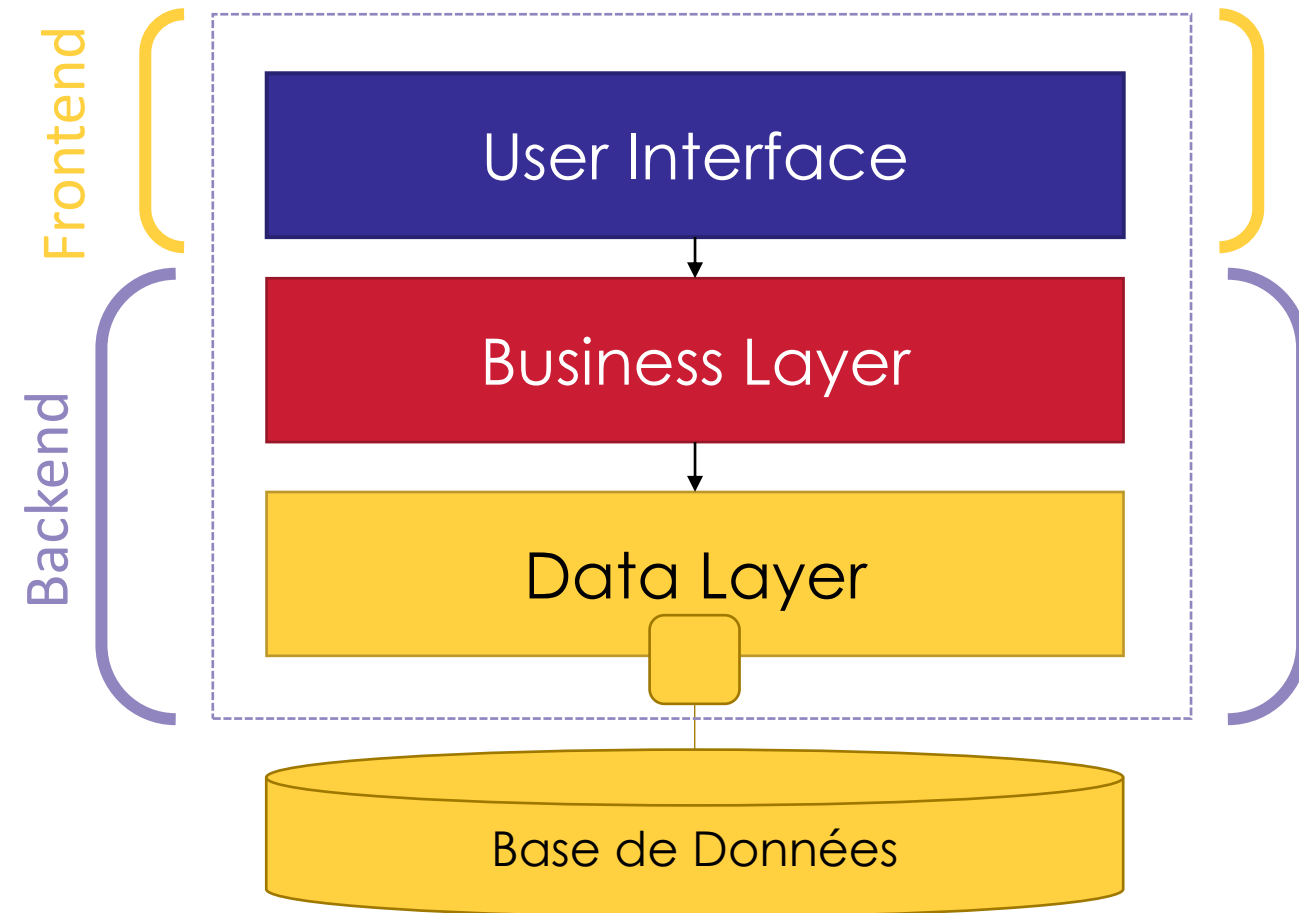
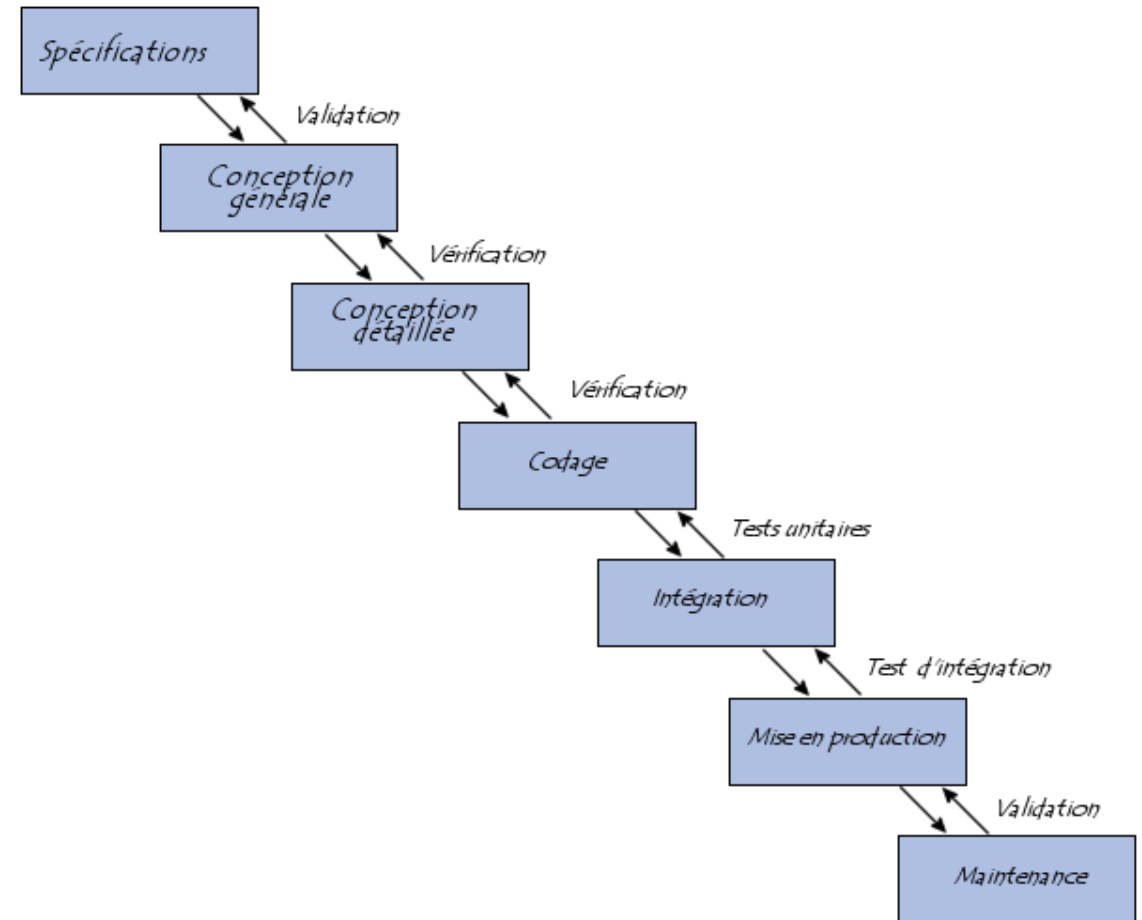


Figure. Architecture monolithique typique

Cycle de vie d'un logiciel

- Définition des objectifs
- Analyse des besoins et faisabilité
- Conception générale : élaboration des spécifications de l'architecture générale
- Conception détaillée : définir chaque sous-ensemble du logiciel
- Codage
- Tests unitaires
- Intégration : s'assurer de l'interfaçage des différents éléments
- Qualification : vérification de la conformité
- ...

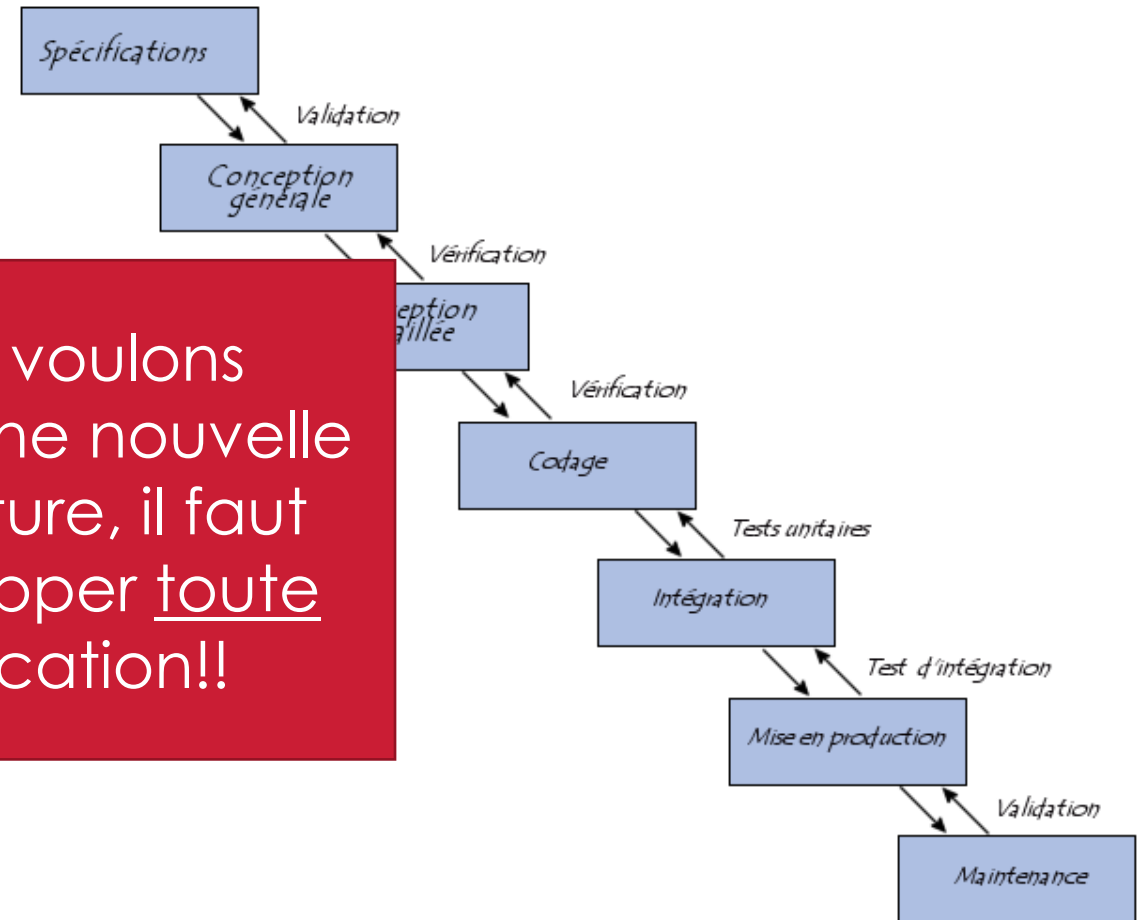


Introduction: cycle de vie d'un logiciel

Cycle de vie d'un logiciel

- Définition des objectifs
- Analyse des besoins et faisabilité
- Conception générale : élaboration des spécifications de l'architecture
- Conception détaillée : définir sous-ensemble du logiciel
- Codage
- Tests unitaires
- Intégration : s'assurer de l'interfonctionnement des différents éléments
- Qualification : vérification de la conformité
- ...

Si nous voulons adopter une nouvelle architecture, il faut redévelopper toute l'application!!



Introduction: cycle de vie d'un logiciel

Cycle de vie d'un logiciel

- Définition des objectifs
- Analyse des besoins et faisabilité
- Conception générale : élaboration des spécifications de l'architecture
- Conception détaillée : définir sous-ensemble du logiciel
- Codage
- Tests unitaires
- Intégration : s'assurer de l'interfonctionnement des différents éléments
- Qualification : vérification de la conformité
- ...

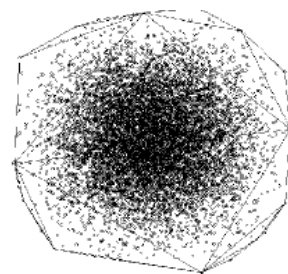
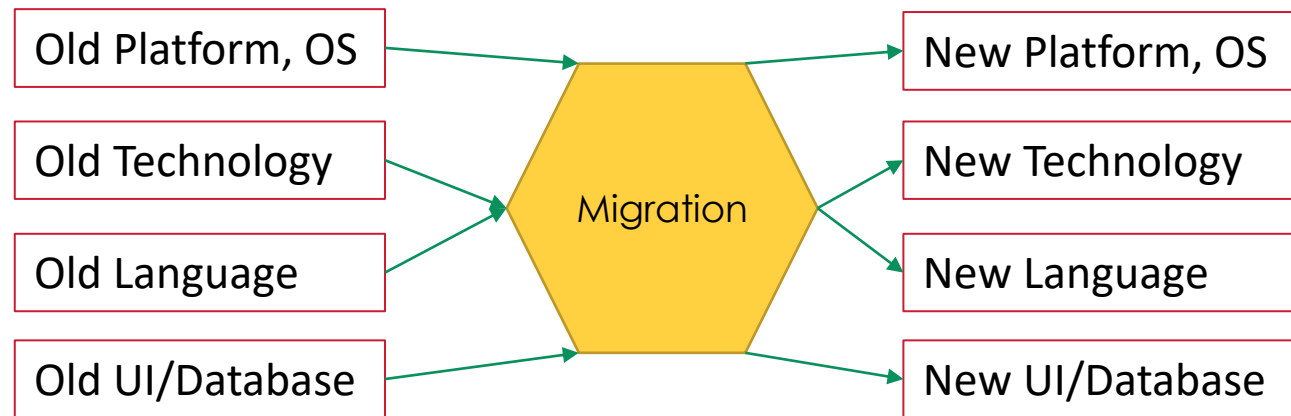
Si nous voulons adopter une nouvelle architecture, il faut redévelopper toute l'application

... ou partir de l'existant?

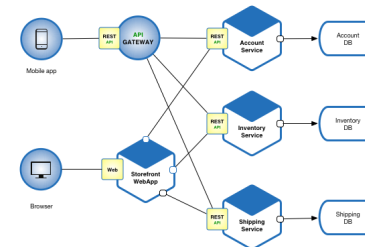


Introduction : Besoin d'évoluer l'architecture

- Migration vers les architectures à base de microservices (MSA)
 - Faire évoluer l'existant



Système orienté objet



Système à base de microservices



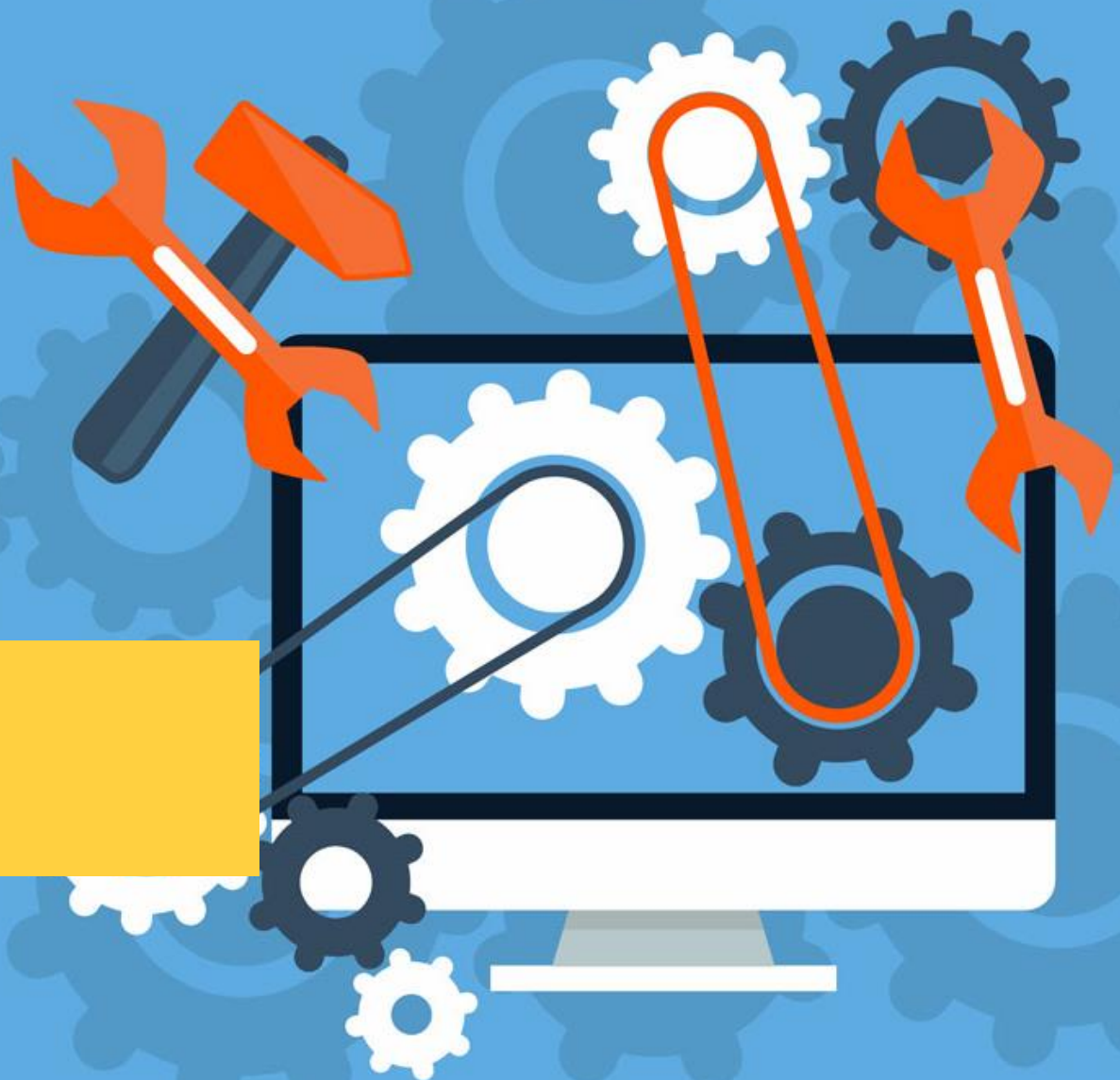
1. Migration manuelle peut-être difficile :

- Coûteuse (en temps et en argent)
- Source d'erreurs
- Grand risque de ne jamais finir

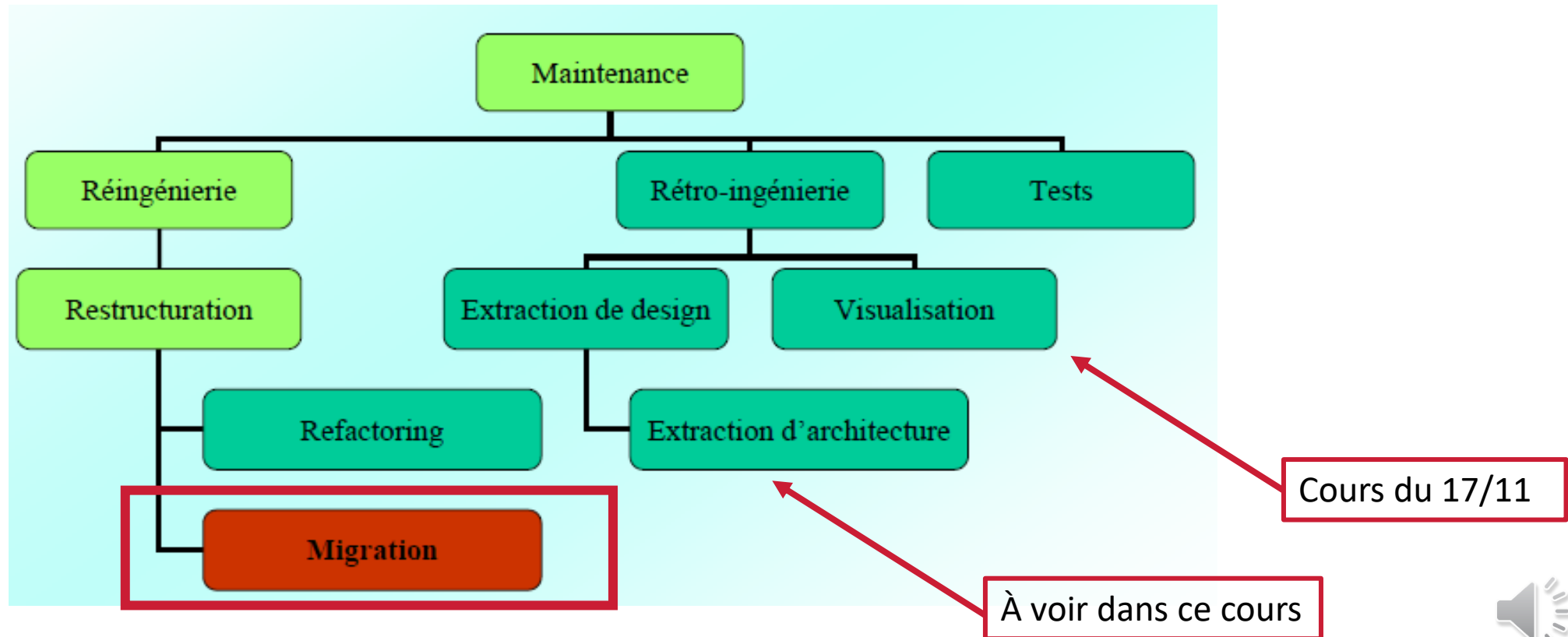
... d'où le besoin
d'automatiser la
migration.



Réingénierie

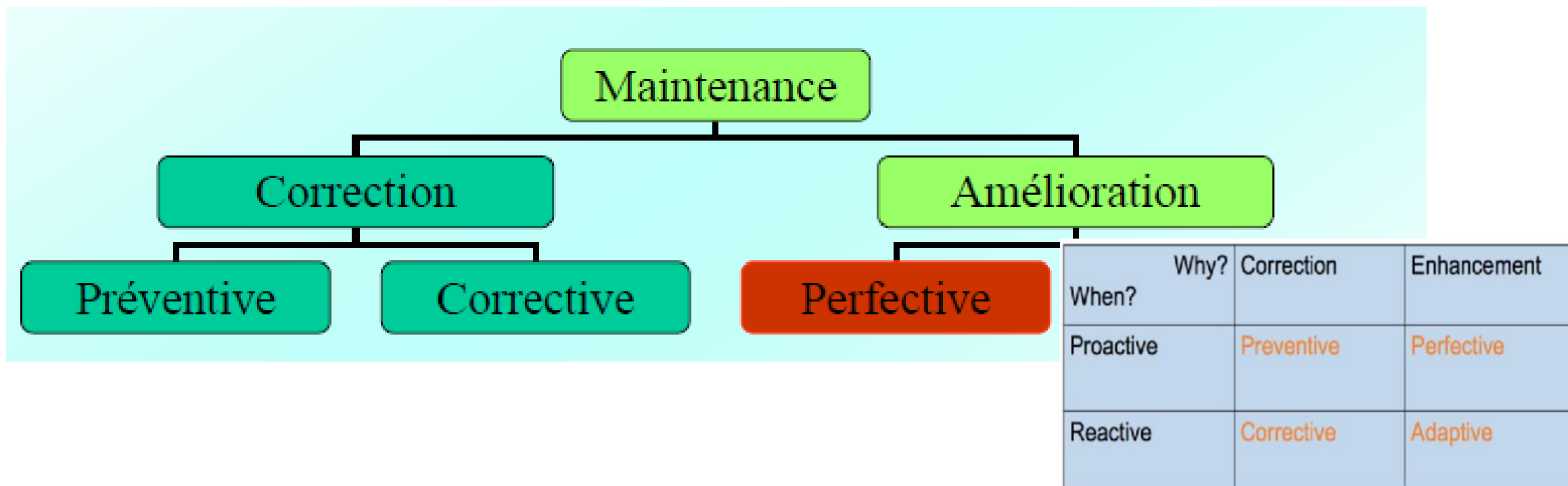


- La migration et les différents types d'évolutions et de maintenance
 - Rappel : l'évolution est un terme qui inclut la maintenance logiciel à grande échelle.



Rappels sur la maintenance

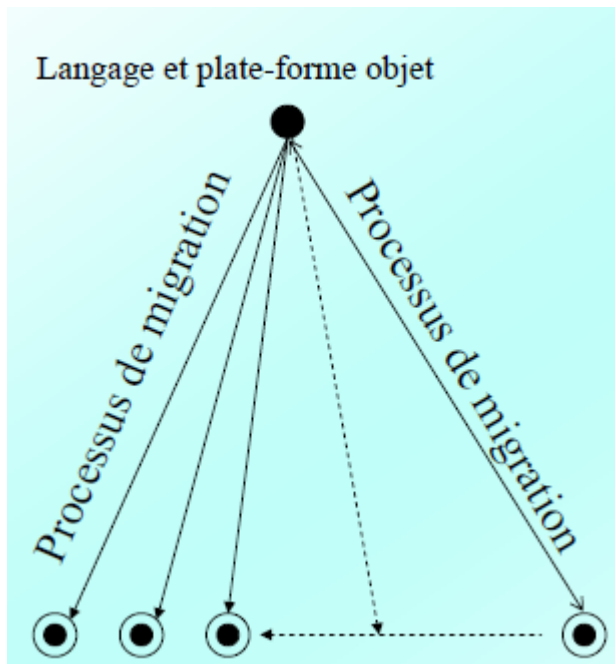
- La migration / les différents objectifs de l'évolution et de la maintenance



Classification de ISO/IEC 14674
(Rappel 1^{er} cours)

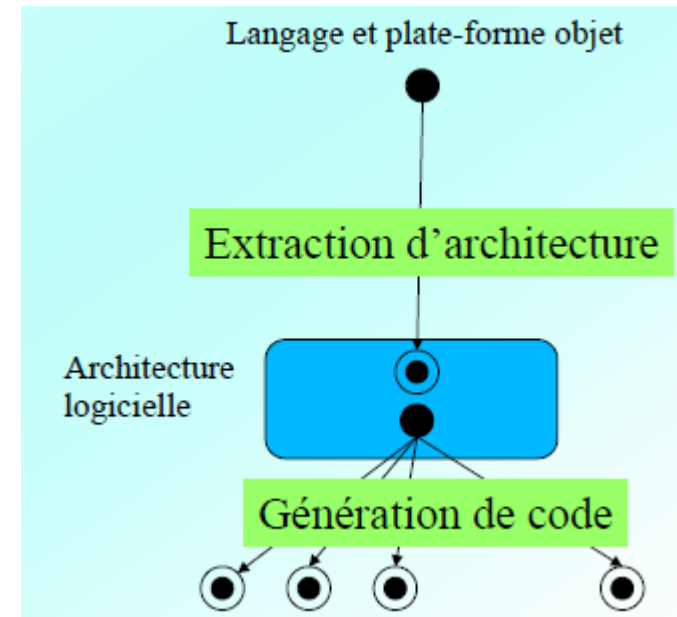
La migration centrée architecture

- Migration point-à-point versus migration centrée architecture



Langages et plateformes à base de microservices

VS



Langages et plateformes à base de microservices

Architecture (ex : architecture à base de microservices)

- Abstraction d'un système logiciel
 - *Microservices*
 - *Encapsule une fonctionnalité*
 - *Interfaces fournies / requises*
 - *Connecteurs*
 - *Communications réseaux entre les microservices (e.g. REST API, message-oriented, event-based communication)*
 - *Configuration*
 - *Topologie des connexions entre les microservices à travers un réseau*



1. Intérêts

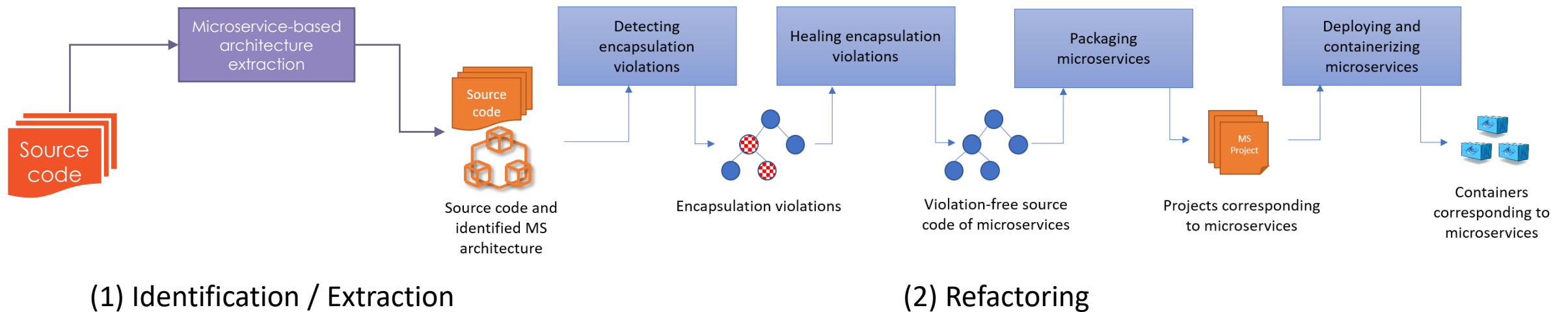
- Echanges entre les différents acteurs du cycle de vie
- Représentation du système :
 - *Meilleure compréhension*
 - *Vérification des propriétés*
 - *Localisation des défauts*
 - *Réduction des risques dans les modifications*
- Réutilisation

2. Etat des lieux

- Systèmes conçus sans représentation de l'architecture
- Systèmes avec une représentation incorrecte
 - *Phénomène d'érosion*
 - *Écart entre l'architecture d'origine et implémentée.*
 - *Manque de synchronisation pendant les phases de maintenance.*

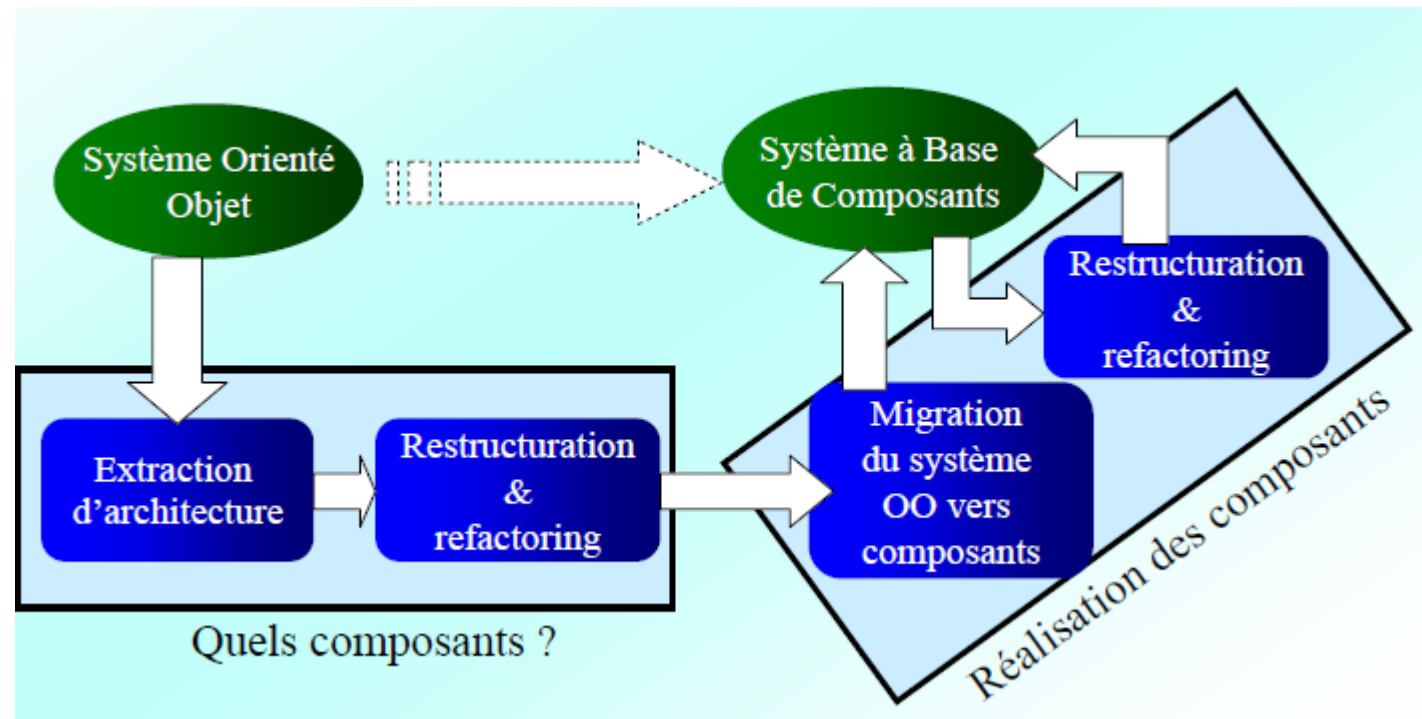


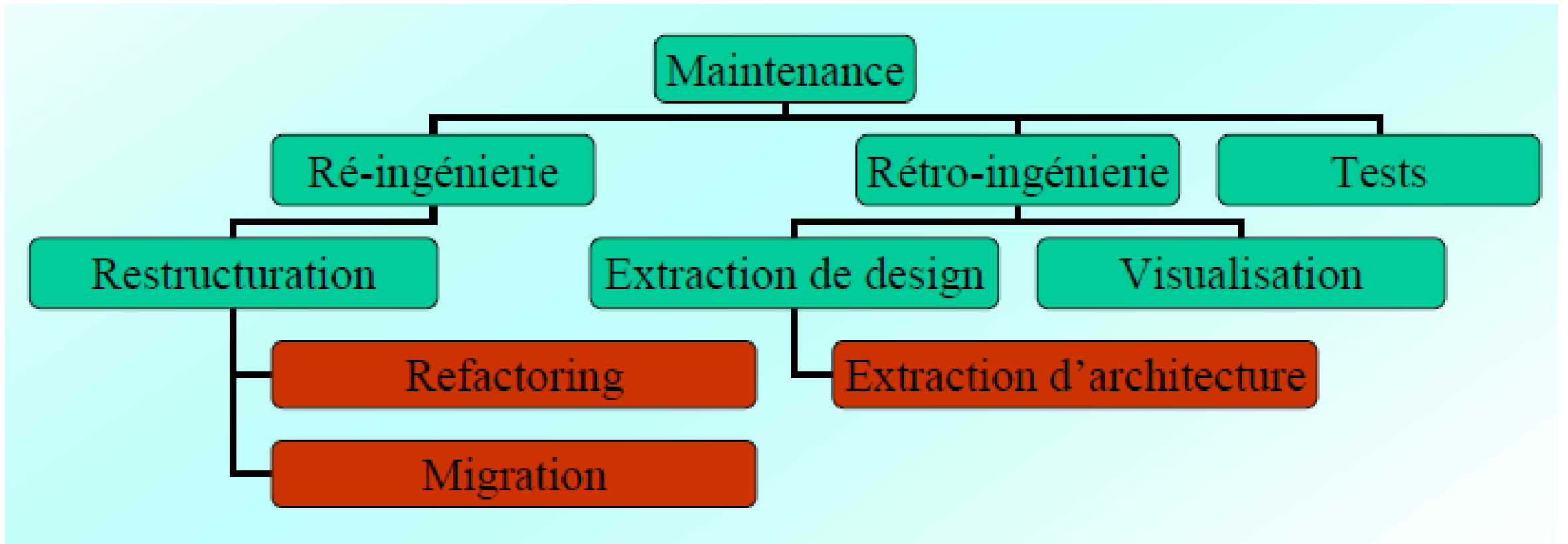
1. **Mono2Micro : Monolith to Microservice Migration Tool**
2. ROMANTIC : **R**e-engineering of **O**bject-oriented **S**ystems by **A**rchitecture extraction and migration to **C**omponent-based ones.



La migration centrée architecture

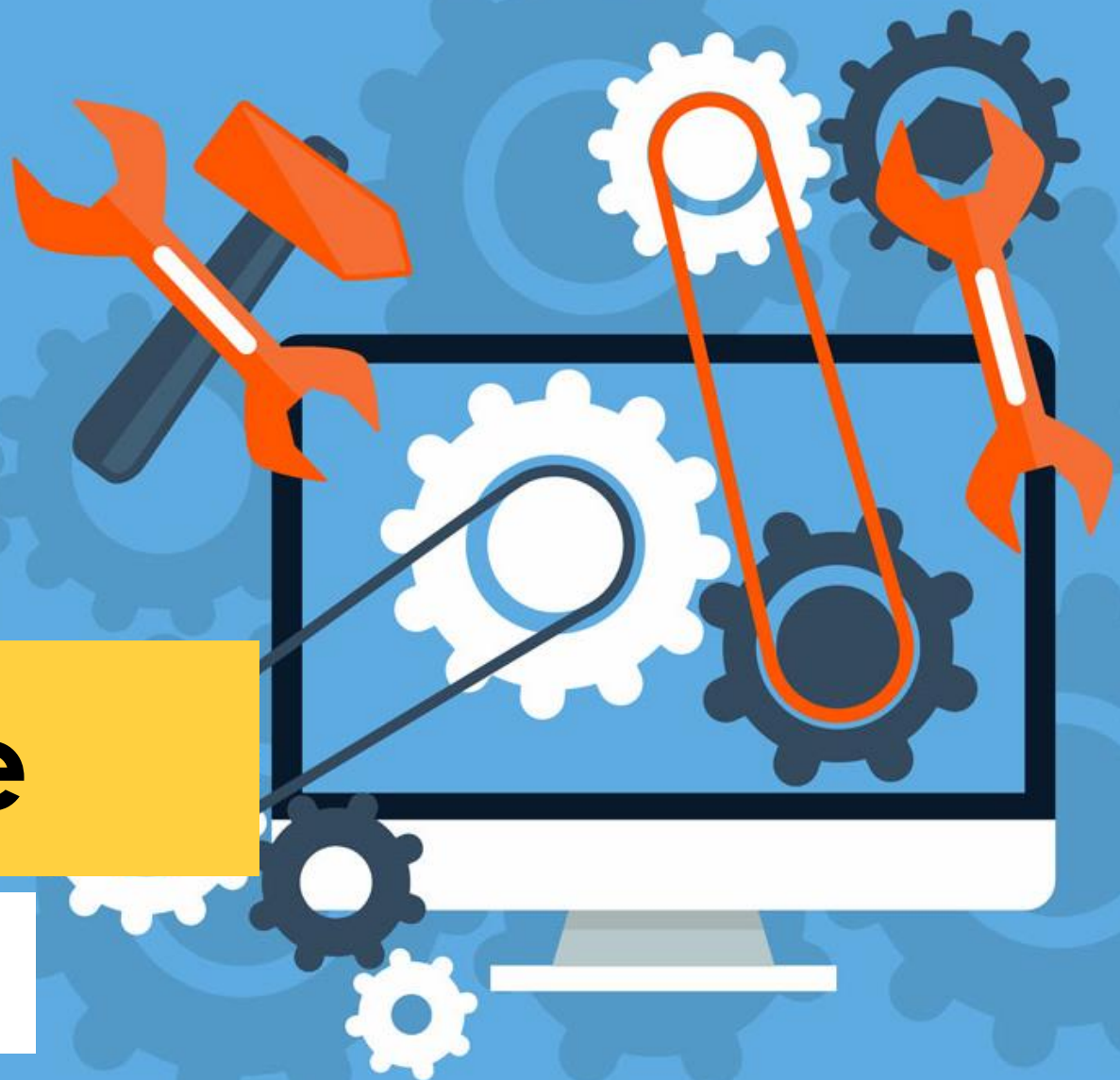
1. Mono2Micro : Monolith to Microservice Migration Tool
2. **ROMANTIC : Re-engineering of Object-oriented Systems by Architecture extraction and migration to Component-based ones.**





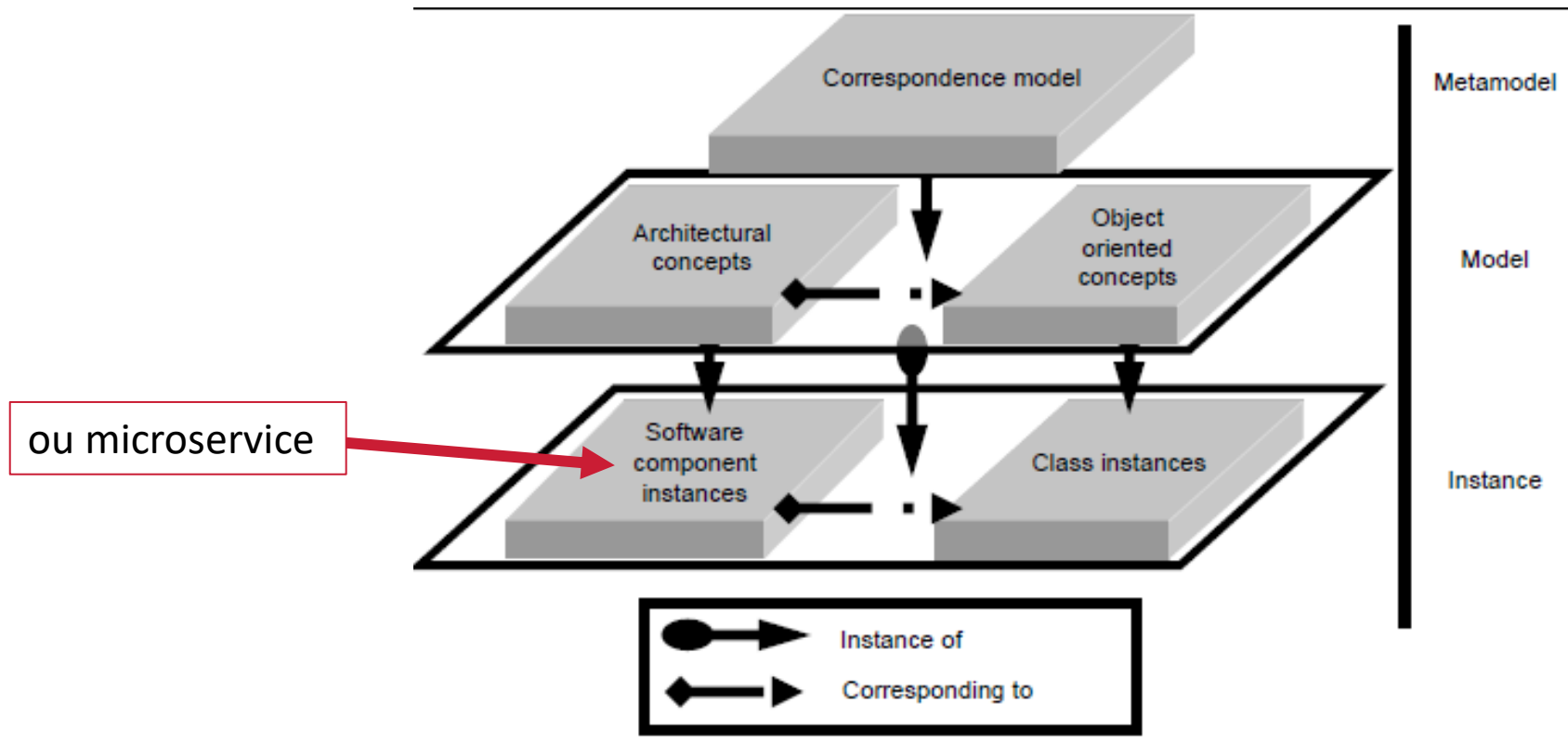
Rétro-Ingénierie

Extraction d'architecture



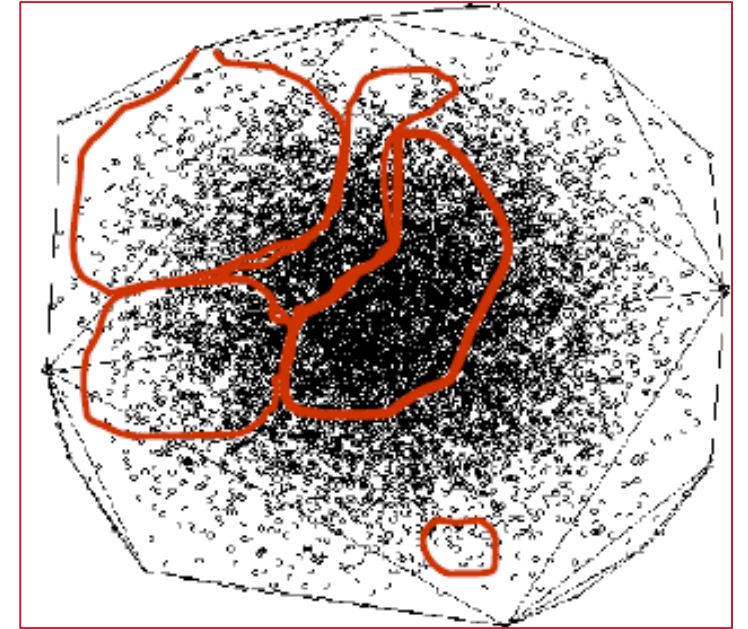
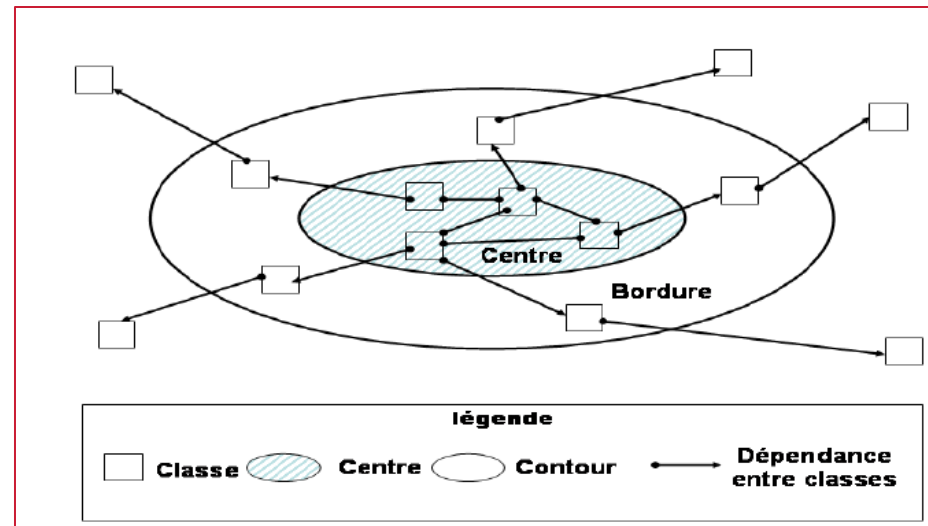
Les principes de l'extraction d'architecture

1. Modèle de correspondance orienté-objet / composant (ou microservice)



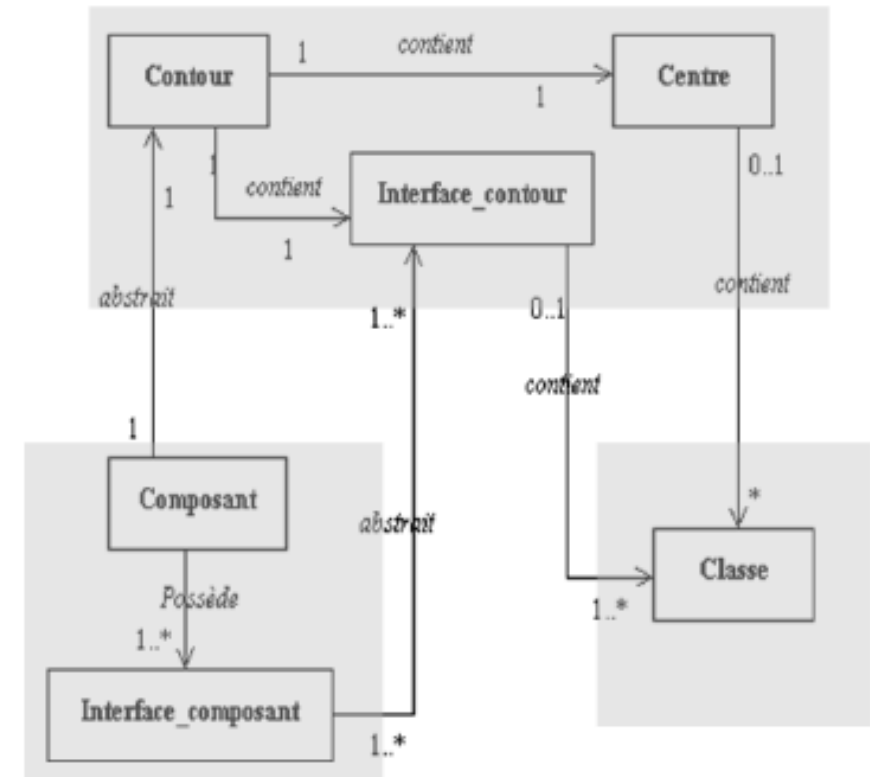
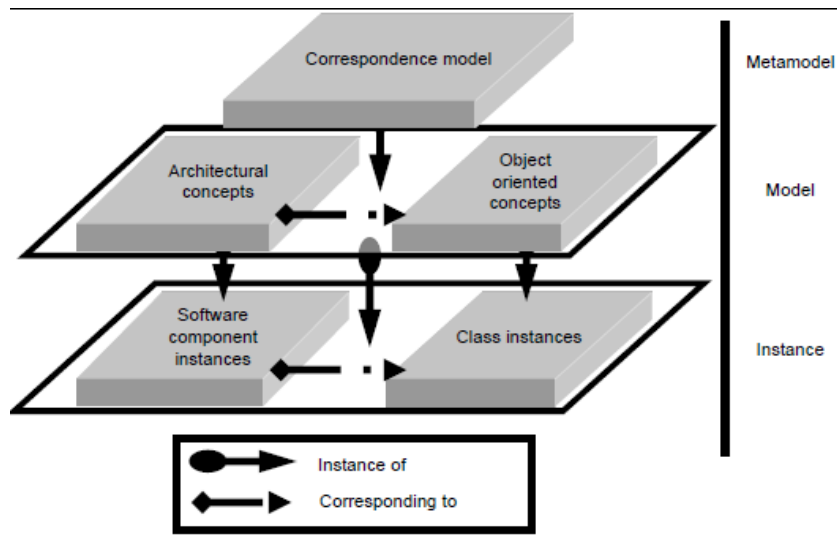
Les principes de l'extraction d'architecture

1. Architecture
 - Une partition des classes du systèmes
2. Composant / Microservice
 - Un ensemble de classes
3. Connecteur
 - Lien de dépendance entre les classes des composants/microservice différents.



Les principes de l'extraction d'architecture

- Contour
 - Ensemble de classes
- Composant
 - Abstraction d'un contour
- Architecture
 - Partition des classes



1. Approche manuelle

- Même difficultés que la conception
- Fort besoin en expertise du code métier

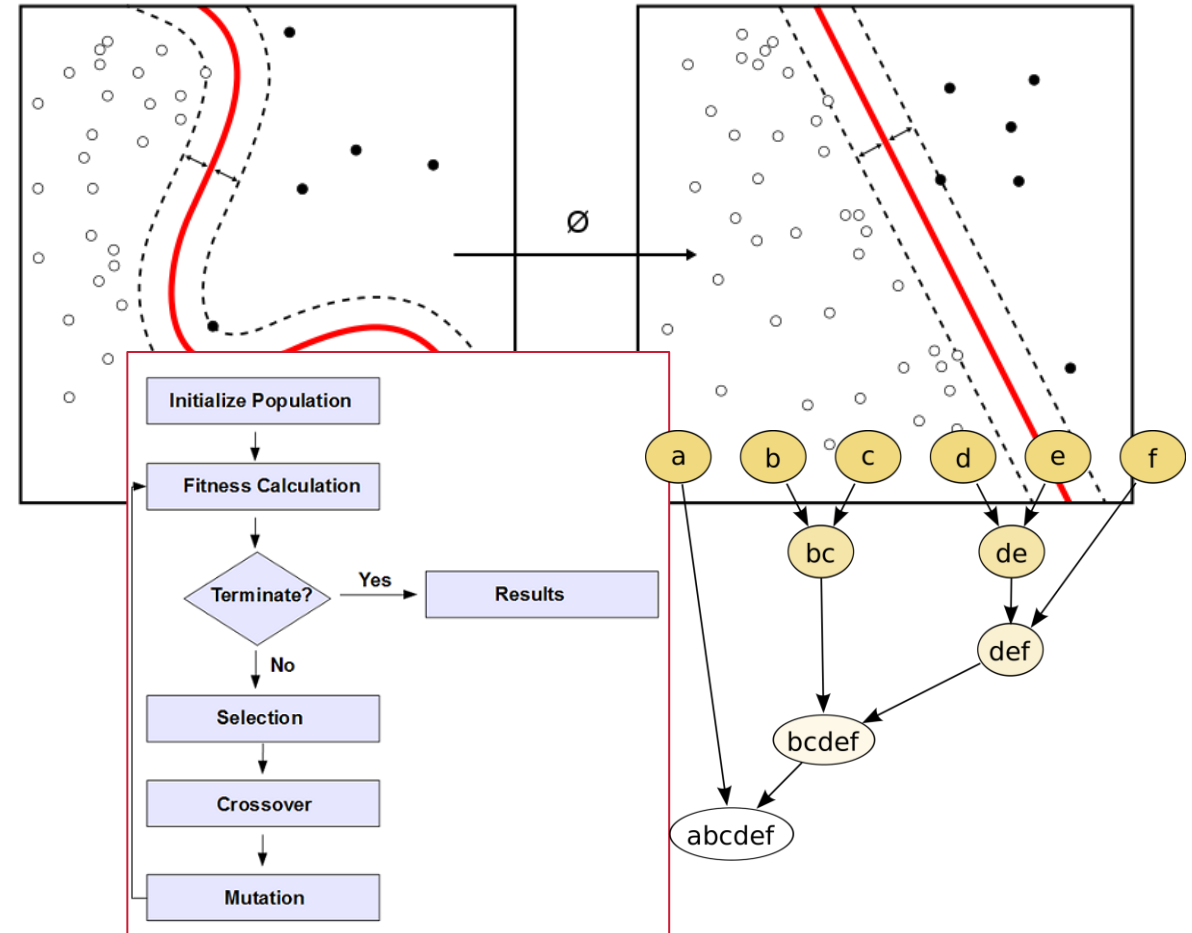
2. Approche automatique

- L'espace des solution
 - Nombre d'architectures possible: $\frac{(2 * n)!}{(n + 1)! * n!}$
 - Choix naïf : sélection aléatoire?
 - La plupart des solution possibles sont mauvaise. Cela nécessite:
 - Sémantique des microservices
 - Granularité des microservices

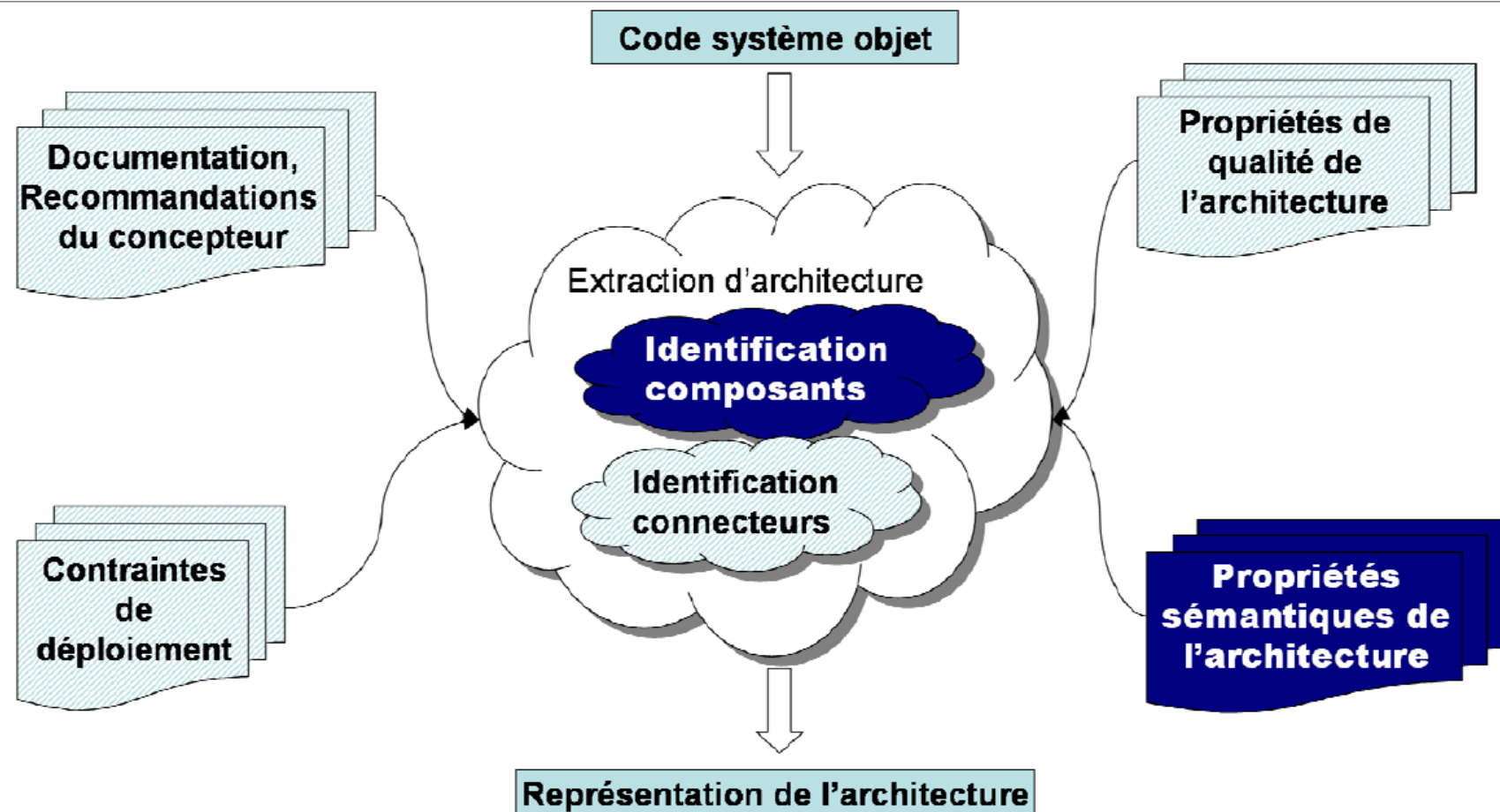


Les principes de l'extraction d'architecture

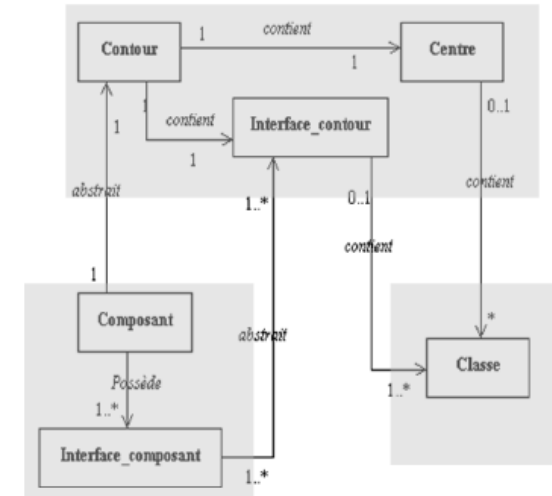
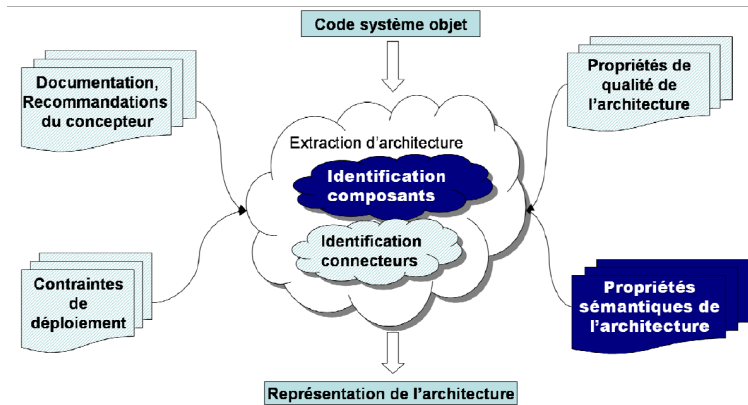
- Processus automatique
 - Limite les besoins en expertise
 - Exploration de l'espace de solution
 - Besoin d'un ensemble de guides
 - Oriente le choix de la meilleure architecture
 - Dirige l'exploration
 - Sélectionne les solutions



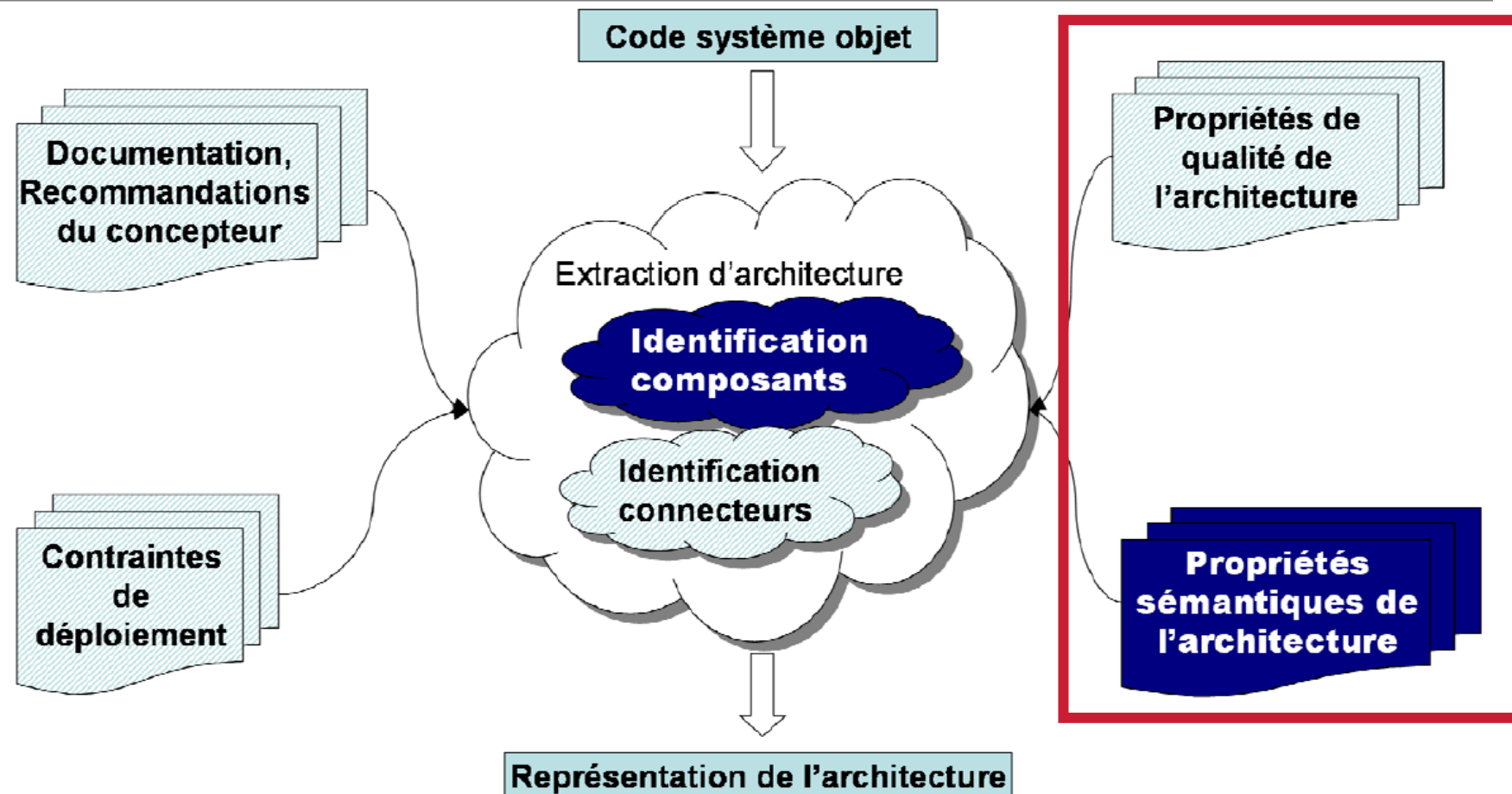
Ressources utiles à l'extraction



Les principes de l'extraction d'architecture



Ressources utiles à l'extraction



Analyse et réification de la sémantique de la notion d'architecture à base de microservices

- Définition d'une MSA :
 - Une application qui consiste d'un ensemble de **petits** services qui sont déployés **indépendamment**. Chaque microservice gère **ses propres données** et communique à travers un ensemble de **protocole de réseau légers** (e.g. HTTP). - Lewis and Fowler
 - Abstraction
 - *Montre les interactions entre les microservices*
 - *Masque les informations purement internes*
- Les éléments architecturaux
 - Les microservices
 - Les connecteurs
 - La configuration



Analyse et réification de la sémantique de la notion d'architecture à base de microservices

1. Objectif

- Mesurer les caractéristiques d'un microservice

2. Problèmes :

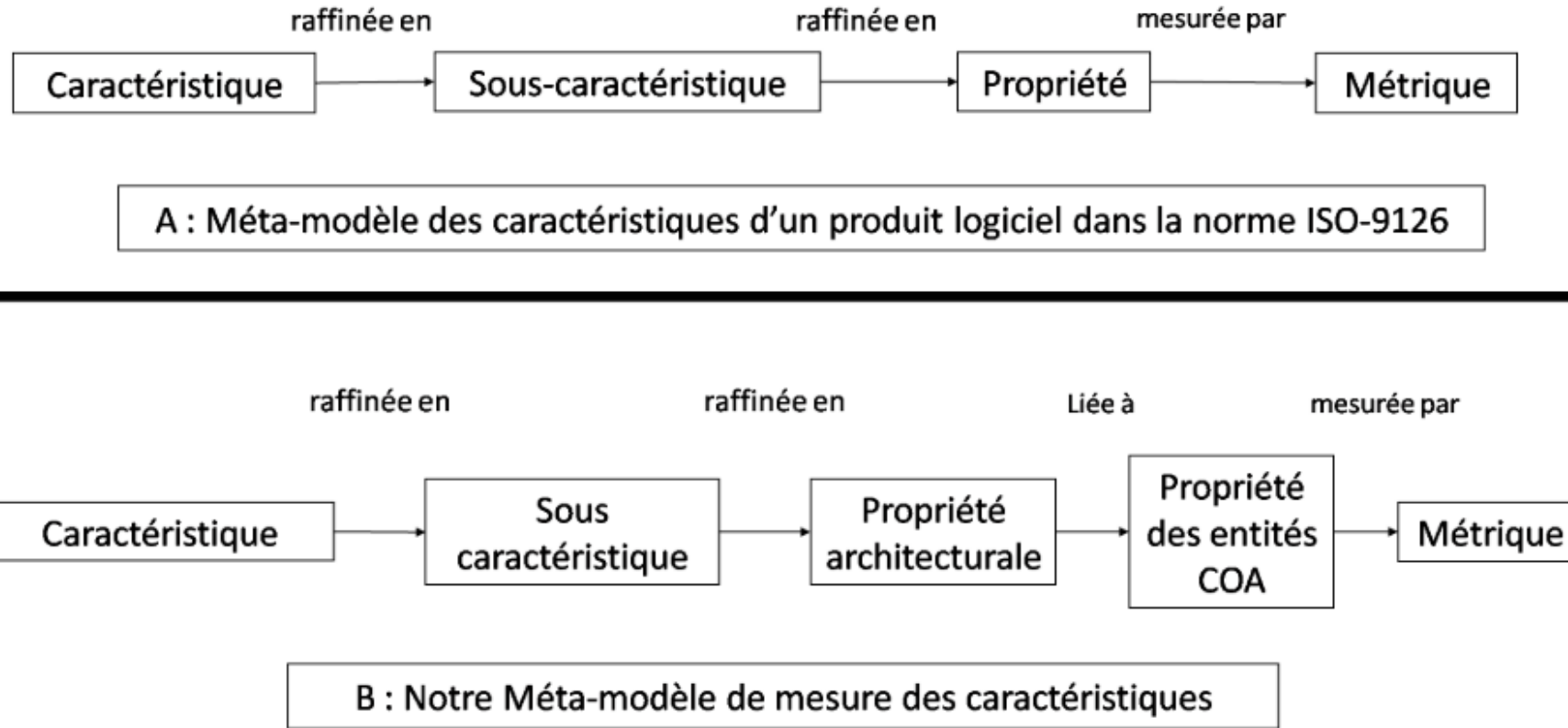
- Absence de microservices (seulement un regroupement de classes)

3. Processus en deux étapes:

- Comment mesurer les caractéristiques sur un microservice?
 - *Identification des propriétés mesurables des microservices*
 - *Lien avec les caractéristiques*
- Comment mesurer ces propriétés mesurables sur un contour?
 - *Identification des propriétés mesurables du contour.*
 - *Liens avec les propriétés mesurables du microservice.*



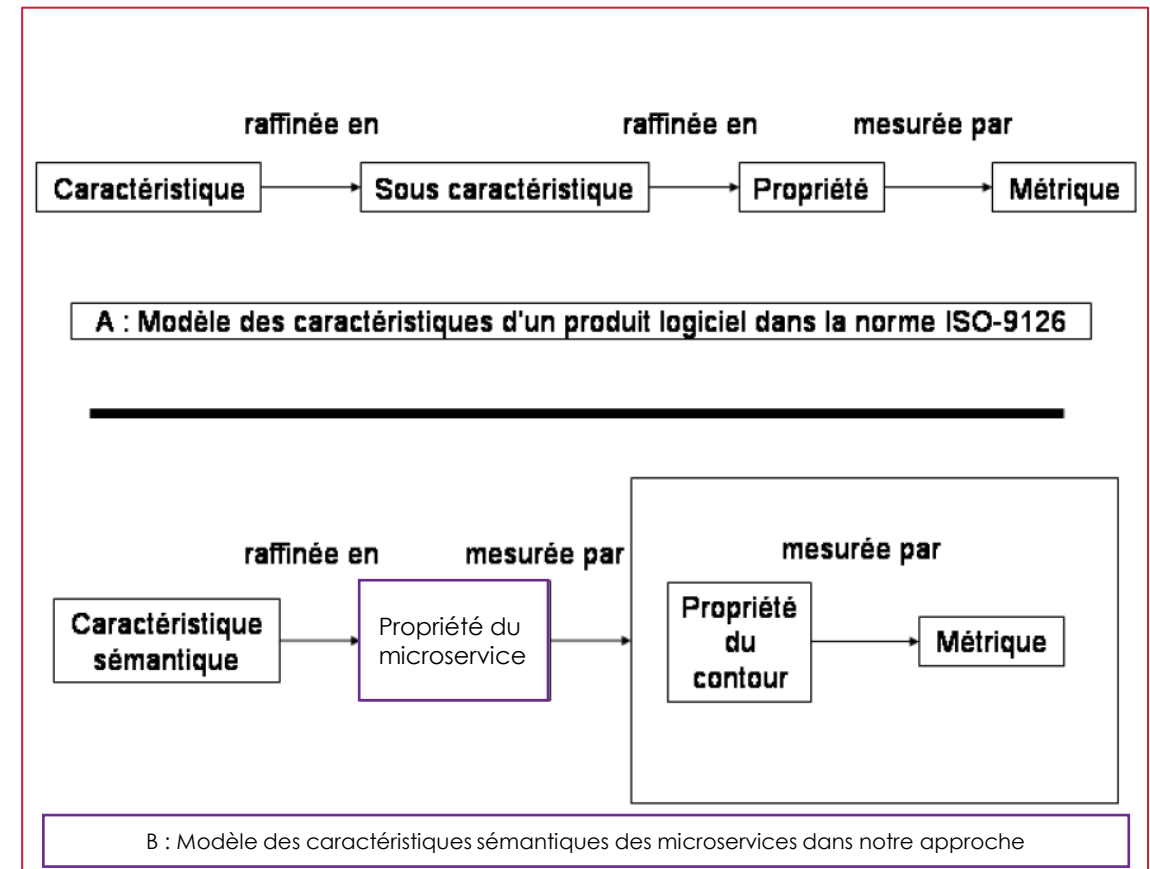
Analyse et réification de la sémantique de la notion d'architecture à base de microservices



(COA = modèle de correspondance objet/architecture)

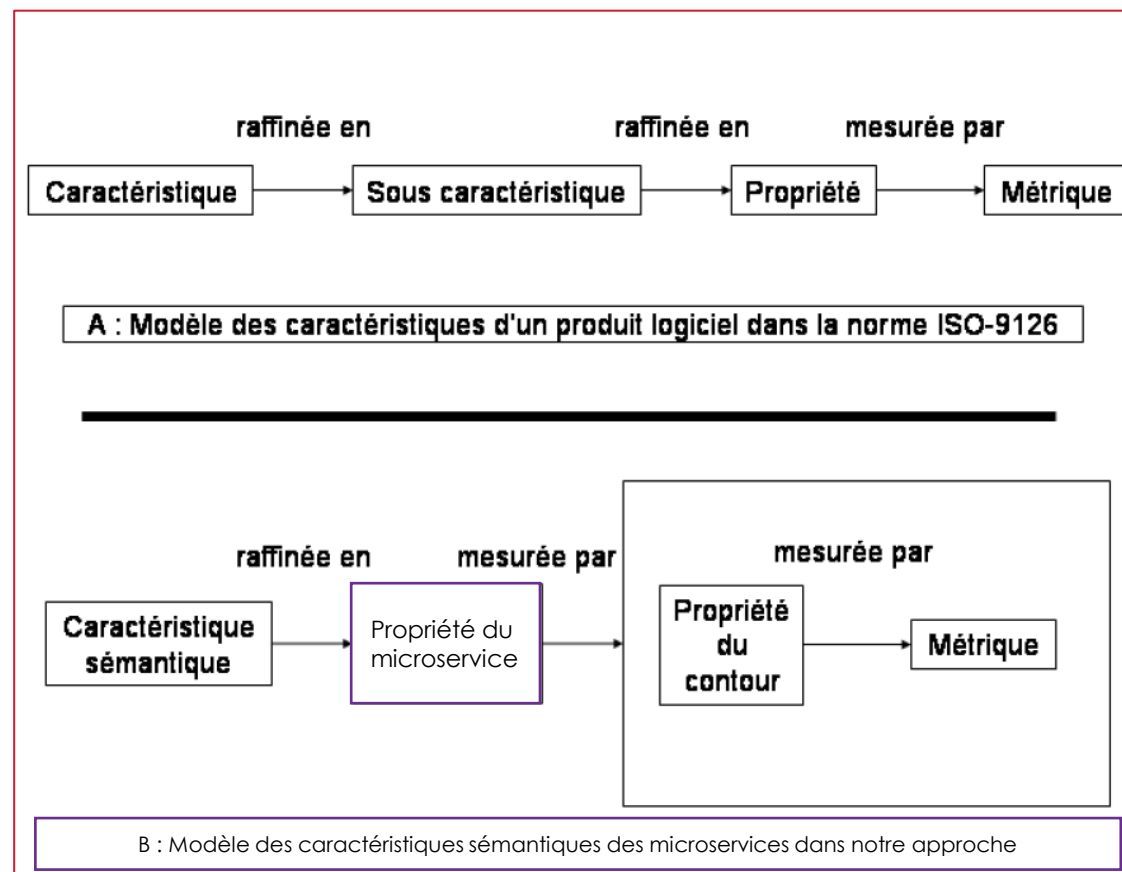
Processus en trois étapes:

1. Identification des liens entre les caractéristiques sémantiques et propriétés des microservices.
2. Identification des liens entre les propriétés des microservices et celles des contours.
3. Choix de métriques pour la mesure des propriétés des contours.



Processus en trois étapes:

1. **Identification des liens entre les caractéristiques sémantiques et propriétés des microservices.**
2. Identification des liens entre les propriétés des microservices et celles des contours.
3. Choix de métriques pour la mesure des propriétés des contours.



Identification des caractéristiques sémantiques et raffinement

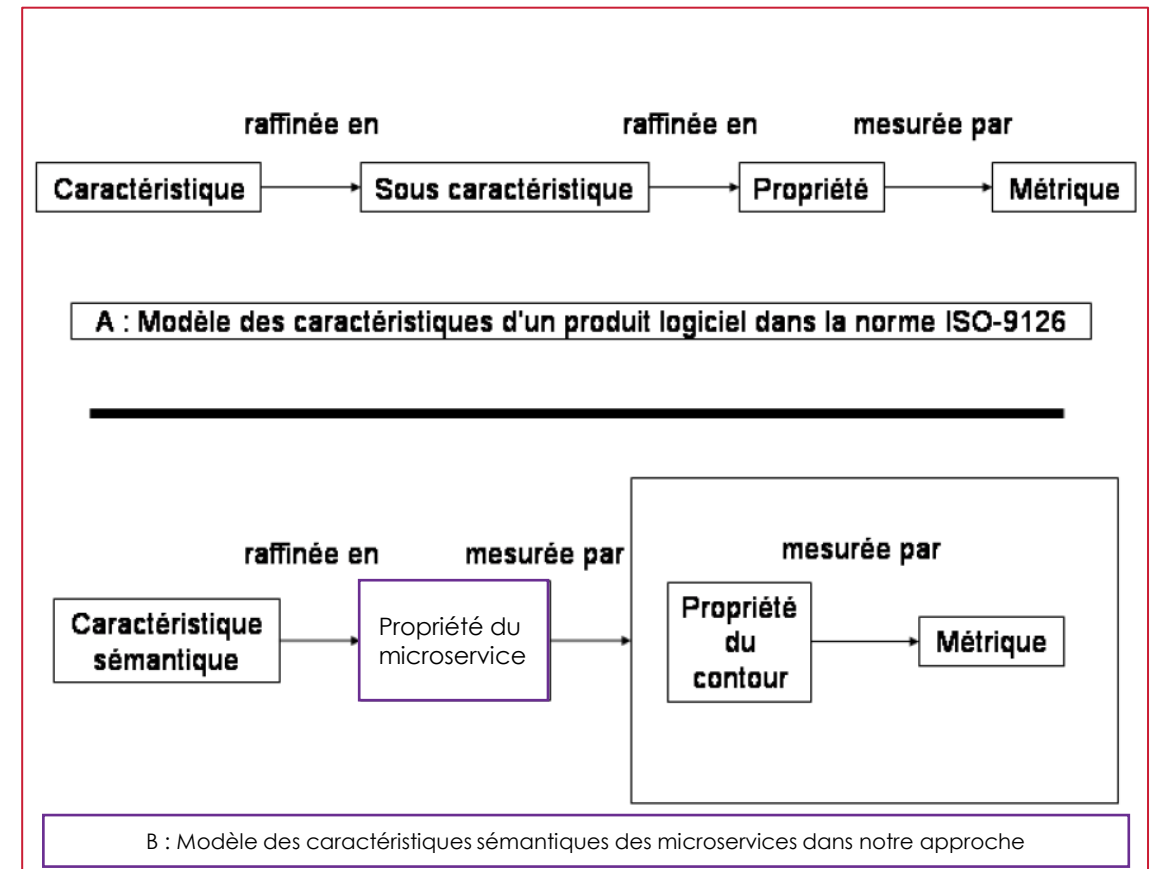
Une **petite** application (ou service) **indépendante** qui tourne sur son propre processus, fournit une **fonctionnalité précise**, utilisant des mécanismes légers, avec un déploiement automatisé et une **autonomie de données**. – Lewis and Fowler

1. « Petit » : Un microservice est petit s'il contient peu de classes.
2. « Indépendant » : Un microservice est indépendant s'il ne dépend pas d'autre microservices.
3. « fonctionnalité précise » : Un microservice tourne autour d'une fonctionnalité s'il remplit un besoin spécifique.
4. « autonomie des données » : Un microservice est autonome s'il n'utilise pas des données d'autres microservices.






Processus en trois étapes:

1. Identification des liens entre les caractéristiques sémantiques et propriétés des microservices.
2. **Identification des liens entre les propriétés des microservices et celles des contours.**
3. Choix de métriques pour la mesure des propriétés des contours.



Une **petite** application (ou service) **indépendante** qui tourne sur son propre processus, fournit une **fonctionnalité précise**, utilisant des mécanismes légers, avec un déploiement automatisé et une **autonomie de données**. – Lewis and Fowler

1. « Petit » : Un microservice est petit s'il contient peu de classes. 
2. « Indépendant » : Un microservice est indépendant s'il ne dépend pas d'autre microservices.
3. « fonctionnalité précise » : Un microservice tourne autour d'une fonctionnalité s'il remplit un besoin spécifique. 
4. « autonomie des données » : Un microservice est autonome s'il n'utilise pas des données d'autres microservices. 

Taille de l'application

Fort couplage interclasse

Faible couplage donnée inter-microservice

Faible couplage inter microservice



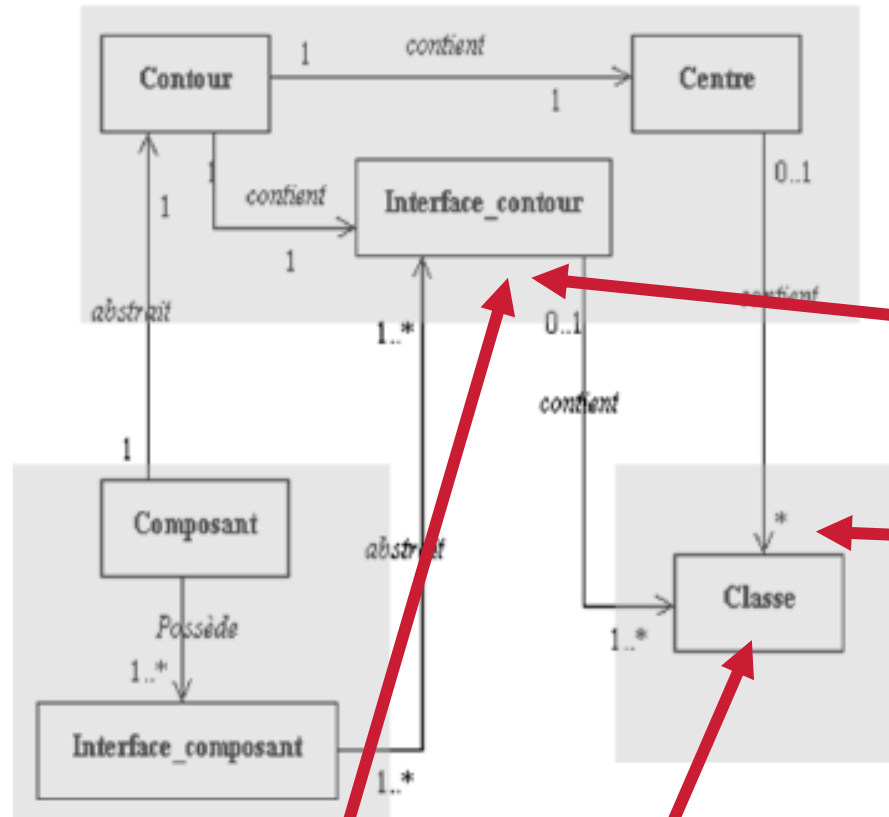
Propriétés du microservice

Propriétés structurelles



Une **petite** application fournit une **forte** cohésion et un **faible** couplage pour le déploiement

1. « Petit » : l'application est petite
2. « Indépendant » : les microservices sont indépendants
3. « fonctionnellement précise » : un microservice fournit une fonctionnalité précise et répond à un besoin spécifique.
4. « autonomie des données » : Un microservice est autonome et ne dépend pas d'autres microservices



Une application tourne sur son propre processus, les microservices sont légers, avec un faible couplage. – Lewis and Fowler

Faible couplage inter microservice

taille de classes.

Taille de l'application

Fort couplage interclasse

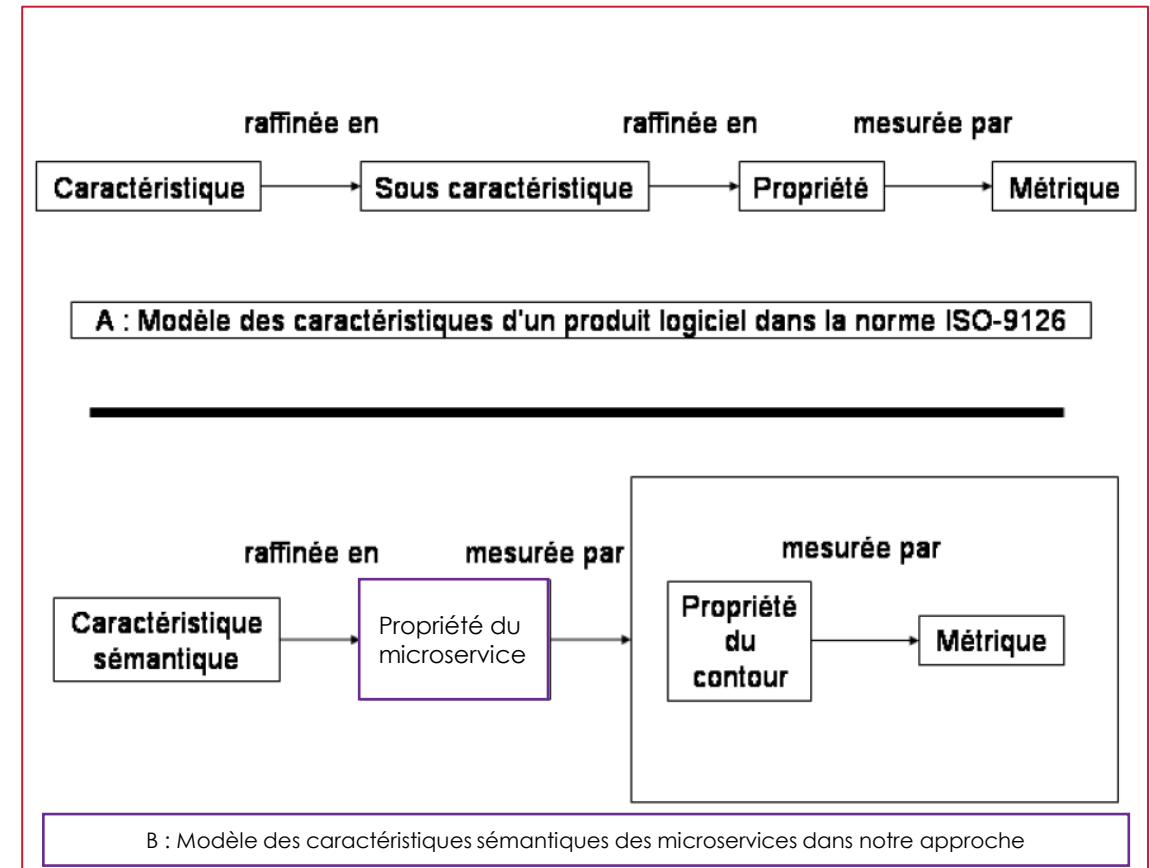
Propriétés du microservice

Faible couplage donnée inter-microservice

Caractéristique	Sous caractéristiques	Propriétés des composants	Propriétés des contours	Métriques
Validité sémantique	Autonomie	Nombre d'interfaces requises	Couplage externe du contour	Couplage
		Couplage du composant	Couplage interne du contour	
	Spécificité	Nombre d'interfaces fournies	Nombre de classes ayant une méthode publique	Nombre de classes ayant une méthode publique
		Cohésion du composant	Cohésion du contour	
		Cohésion des interfaces du composant	Cohésion de l'interface	LCC
	Composabilité	Moyenne de la cohésion des services par interface	Cohésion moyenne des classes de l'interface	

Processus en trois étapes:

1. Identification des liens entre les caractéristiques sémantiques et propriétés des microservices.
2. Identification des liens entre les propriétés des microservices et celles des contours.
3. **Choix de métriques pour la mesure des propriétés des contours.**



- A partir de ces

$$CouplingPair(C1, C2) = \frac{NbCalls(C1, C2) + NbCalls(C2, C1)}{TotalNbCalls}$$

Faible couplage
inter-microservice

$$ExternalCoupling(M) = \frac{\sum CouplingPair(P) - \sum_{PVal \in PairsVal} \sigma(PVal)}{NbPossibleExternalPairs}$$

Fort couplage
intraclasse

$$InternalCohesion(M) = \frac{NbDirectConnections}{NbPossibleConnections}$$

[1] A. Selmadji, A. Seriali, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza and C. Dony, "From Monolithic Architecture Style to Microservice one Based on a Semi-Automatic Approach," *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil, 2020, pp. 157-168, doi: 10.1109/ICSA47634.2020.00023.

- Grace à ces métriques on peut créer une fonction de qualité à partir d'une caractéristique tel qu'un microservice a une « fonctionnalité précise » :

$$FOne(M) = \frac{1}{2}(InternalCoupling(M) + InternalCohesion(M))$$

- Qu'un microservice est indépendant structurellement :

$$FAutonomy(M) = ExternalCoupling(M)$$

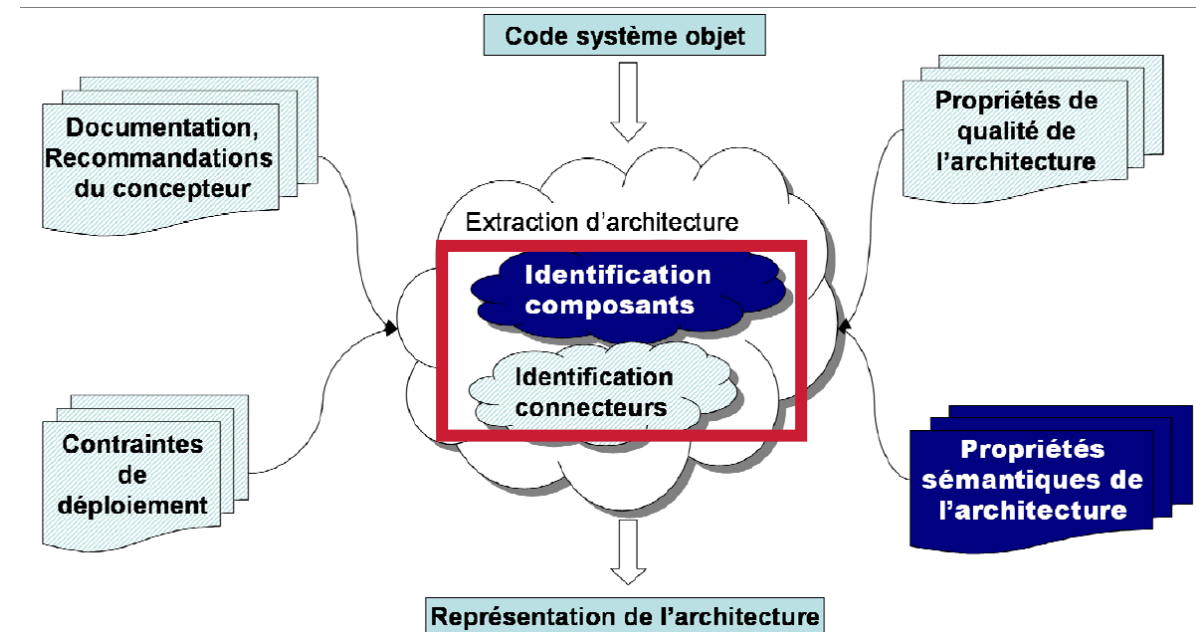
- Ou qu'un microservice est autonome en données:

$$FData(M) = \frac{1}{n} (\alpha FIntra(M) - \beta FInter(M))$$

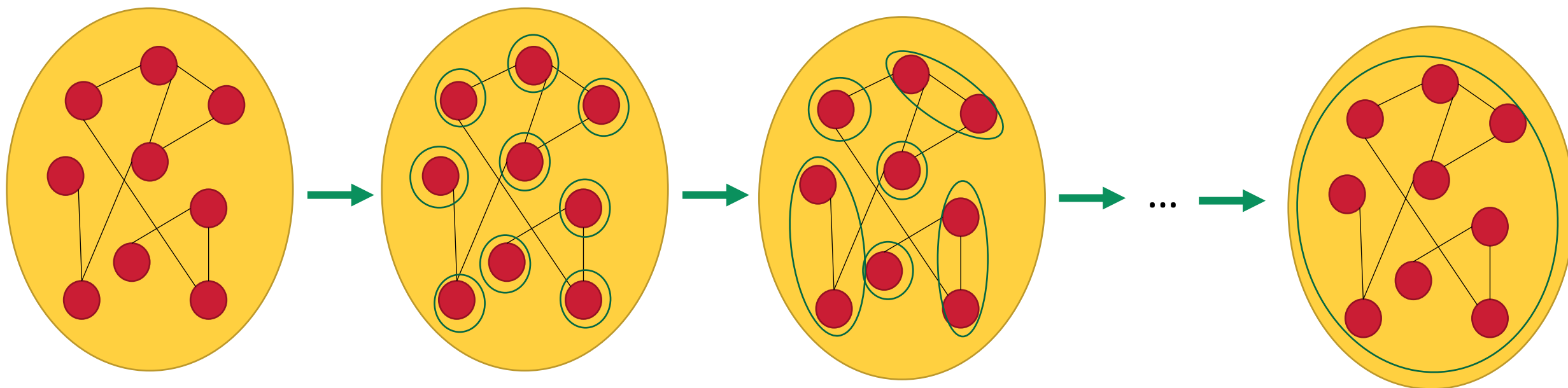


Processus d'identification d'architecture via partitionnement

- Clustering hiérarchique
 - Nécessite une fonction d'évaluation de la sémantique.
 - Résultat:
 - *Un dendrogramme*
 - *Une hiérarchie de clusters*
- Or on souhaite avoir une partition des classes
- Identification des microservices
 - Coupe dans le dendrogramme
 - Objectif : une partition



Identification de microservices : Regroupement Hiérarchique



Légende

- classe
- partition / cluster
- dépendance orienté-objet
- Système existant

Identification de microservices : Regroupement Hiérarchique

Algorithme de regroupement Hiérarchique :

1. Initialisation avec 1 nœud pour chaque classe.

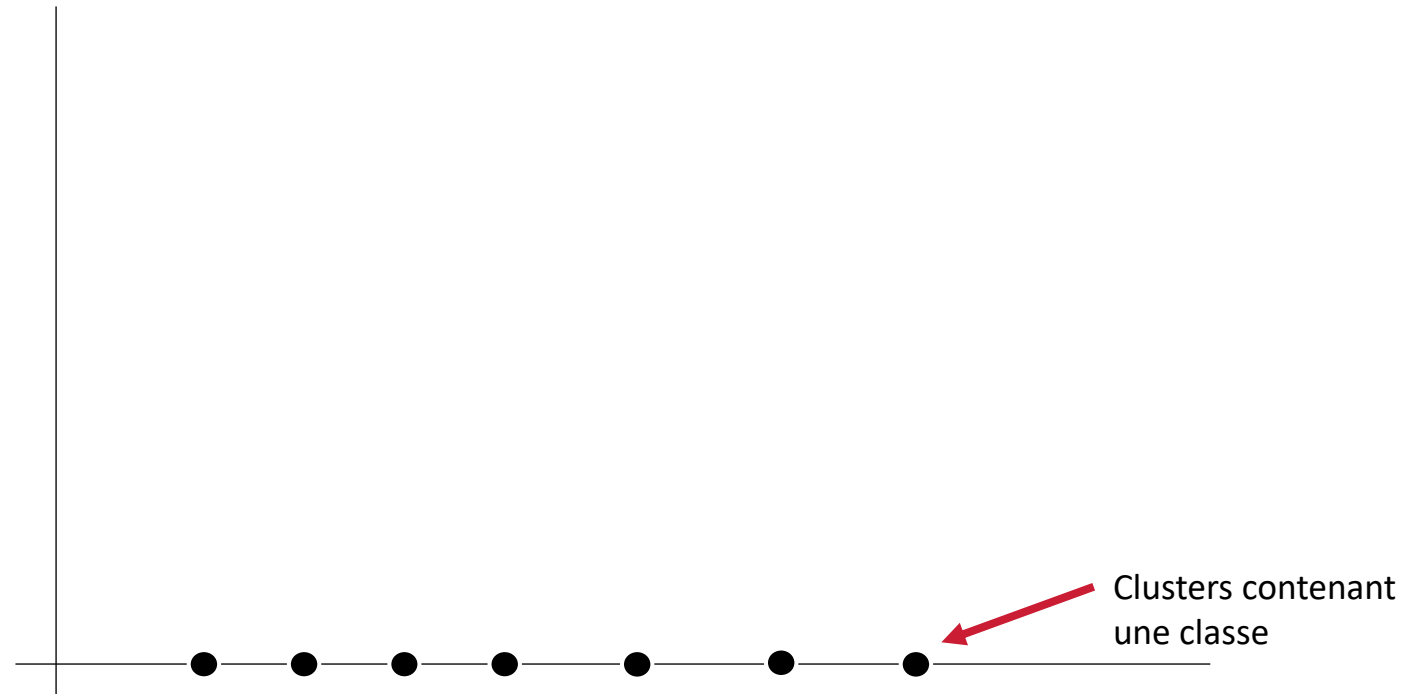


Figure. Création du dendrogramme

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Algorithme de regroupement Hiérarchique :

1. Initialisation avec 1 nœud pour chaque classe.
2. Enumérer l'ensemble des couplages de nœuds et sélectionner celui avec le meilleur score en utilisant la fonction de qualité.

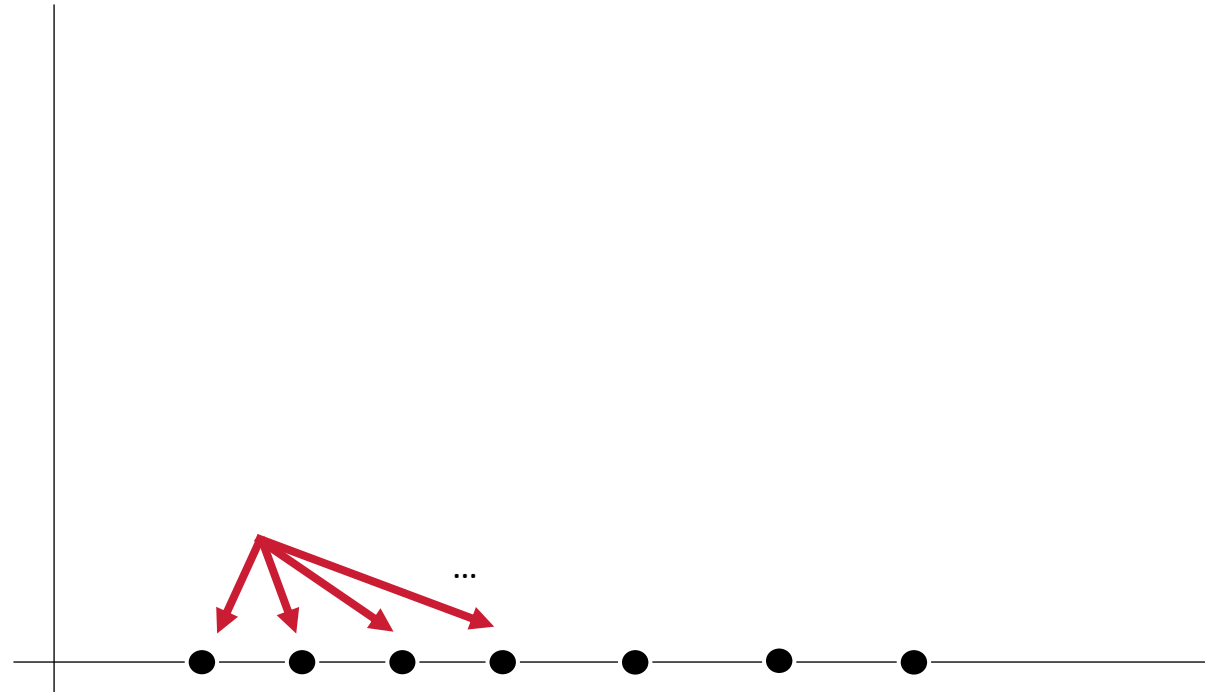


Figure. Création du dendrogramme

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Algorithme de regroupement Hiérarchique :

1. Initialisation avec 1 nœud pour chaque classe.
2. Enumérer l'ensemble des couplages de nœuds et sélectionner celui avec le meilleur score en utilisant la fonction de qualité.
3. Réunir les deux nœuds avec le meilleur score.

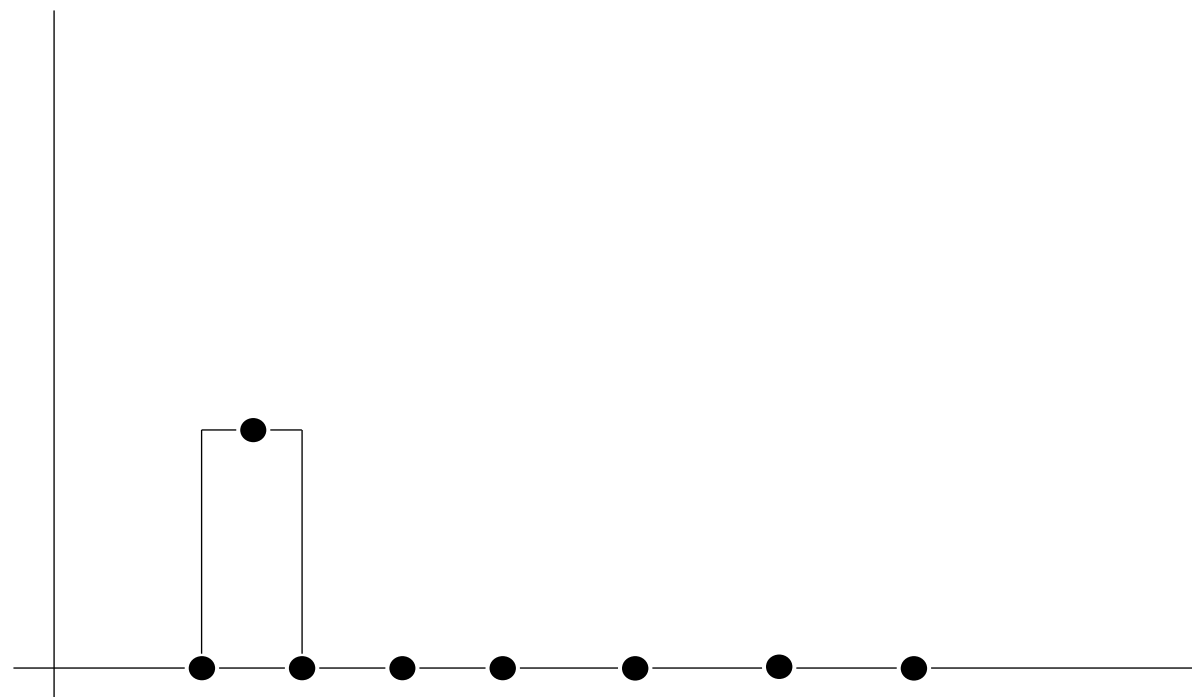


Figure. Création du dendrogramme

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Algorithme de regroupement Hiérarchique :

1. Initialisation avec 1 nœud pour chaque classe.
2. Enumérer l'ensemble des couplages de nœuds et sélectionner celui avec le meilleur score en utilisant la fonction de qualité.
3. Réunir les deux nœuds avec le meilleur score.
4. Répéter jusqu'il n'y ai qu'un nœud.
5. ... Profit?

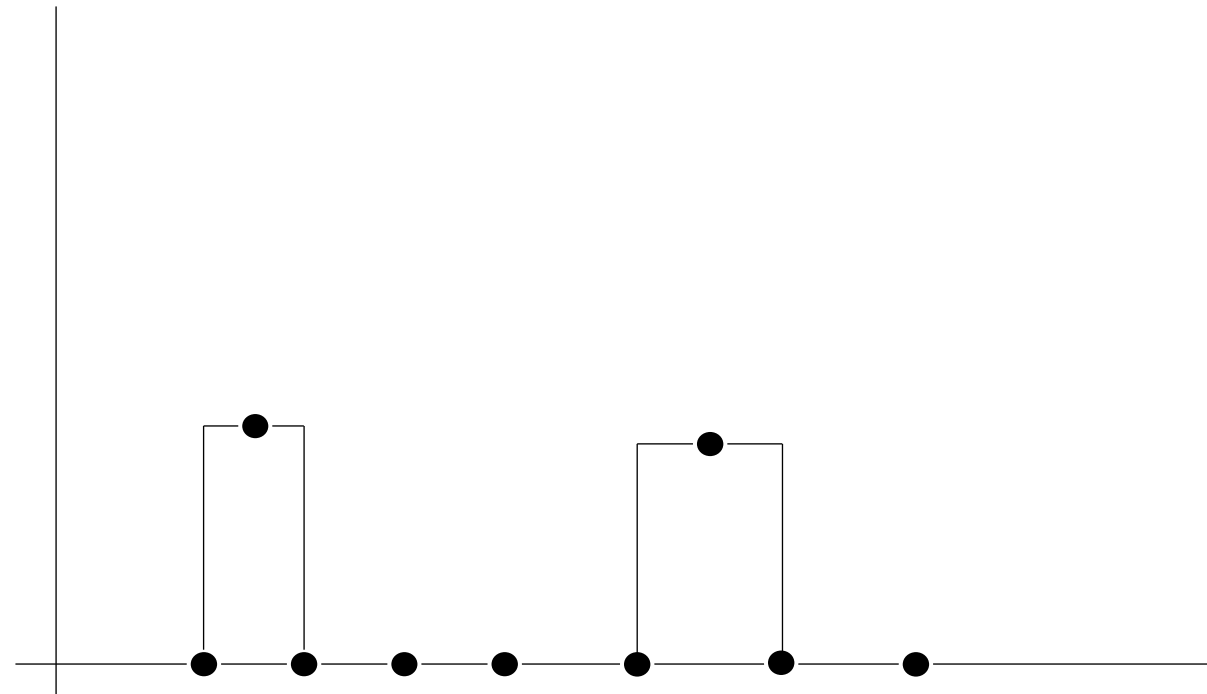


Figure. Création du dendrogramme

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Algorithme de regroupement Hiérarchique :

1. Initialisation avec 1 nœud pour chaque classe.
2. Enumérer l'ensemble des couplages de nœuds et sélectionner celui avec le meilleur score en utilisant la fonction de qualité.
3. Réunir les deux nœuds avec le meilleur score.
4. Répéter jusqu'il n'y ai qu'un nœud.
5. ... Profit?

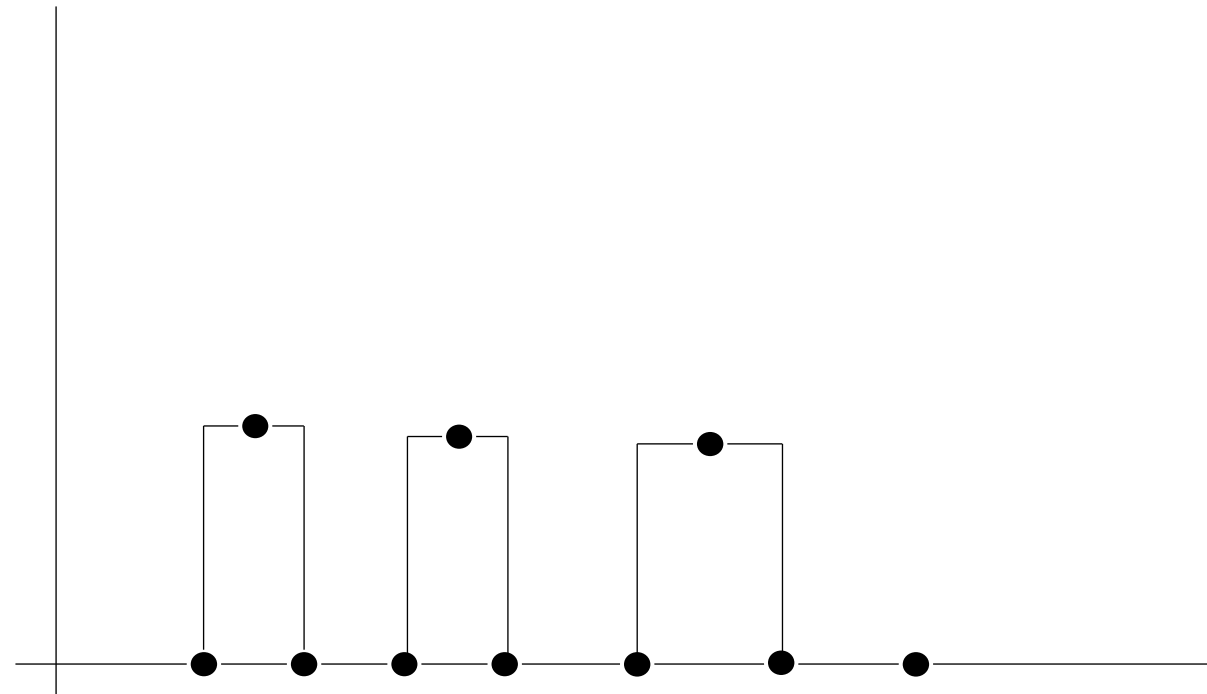


Figure. Création du dendrogramme

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Algorithme de regroupement Hiérarchique :

1. Initialisation avec 1 nœud pour chaque classe.
2. Enumérer l'ensemble des couplages de nœuds et sélectionner celui avec le meilleur score en utilisant la fonction de qualité.
3. Réunir les deux nœuds avec le meilleur score.
4. Répéter jusqu'il n'y ai qu'un nœud.
5. ... Profit?

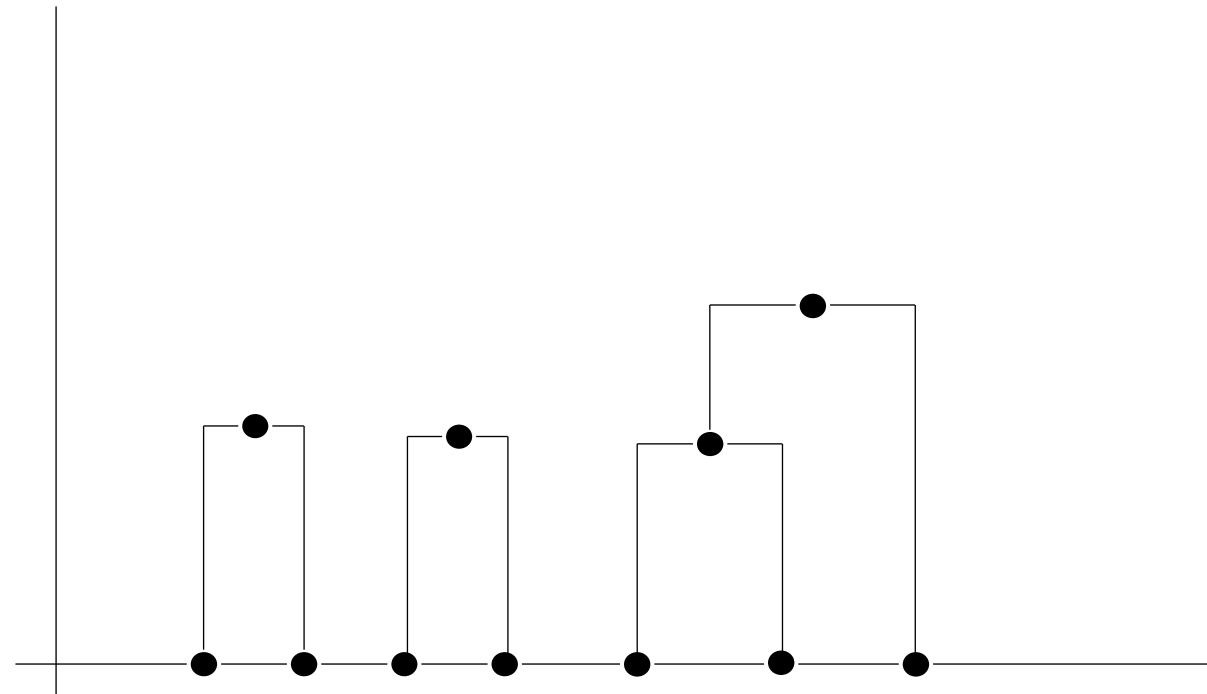


Figure. Création du dendrogramme

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Algorithme de regroupement Hiérarchique :

1. Initialisation avec 1 nœud pour chaque classe.
2. Enumérer l'ensemble des couplages de nœuds et sélectionner celui avec le meilleur score en utilisant la fonction de qualité.
3. Réunir les deux nœuds avec le meilleur score.
4. Répéter jusqu'il n'y ai qu'un nœud.
5. ... Profit?

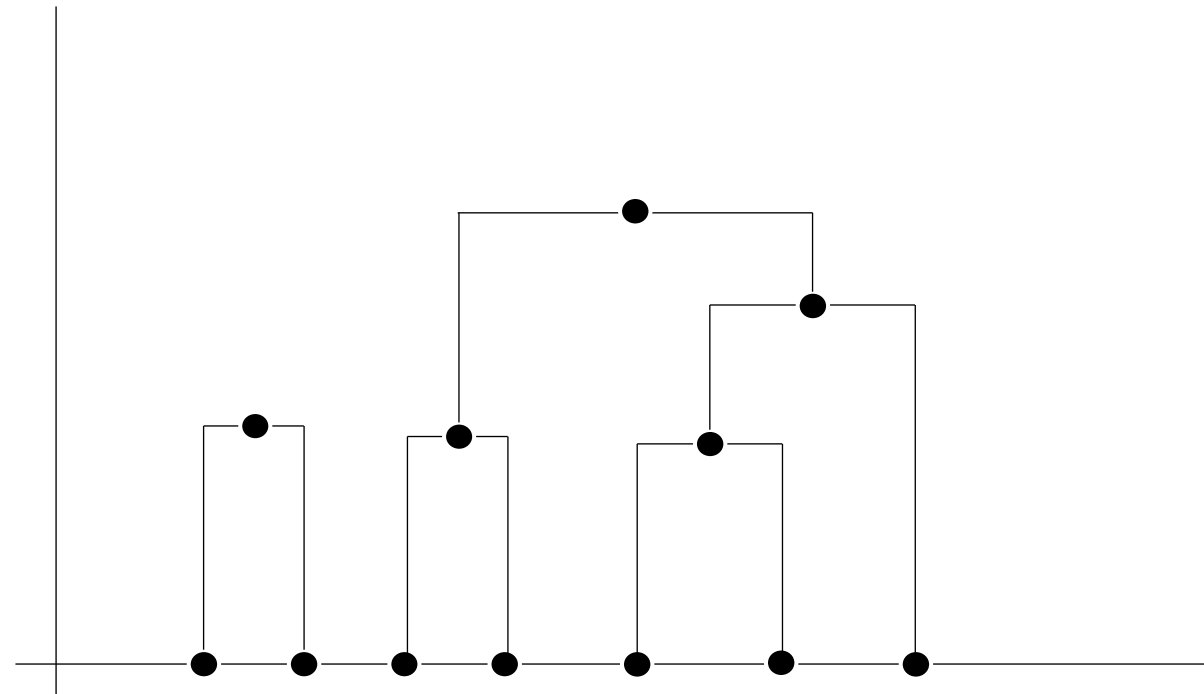


Figure. Création du dendrogramme

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Algorithme de regroupement Hiérarchique :

1. Initialisation avec 1 nœud pour chaque classe.
2. Enumérer l'ensemble des couplages de nœuds et sélectionner celui avec le meilleur score en utilisant la fonction de qualité.
3. Réunir les deux nœuds avec le meilleur score.
4. Répéter jusqu'il n'y ai qu'un nœud.
5. ... Profit?

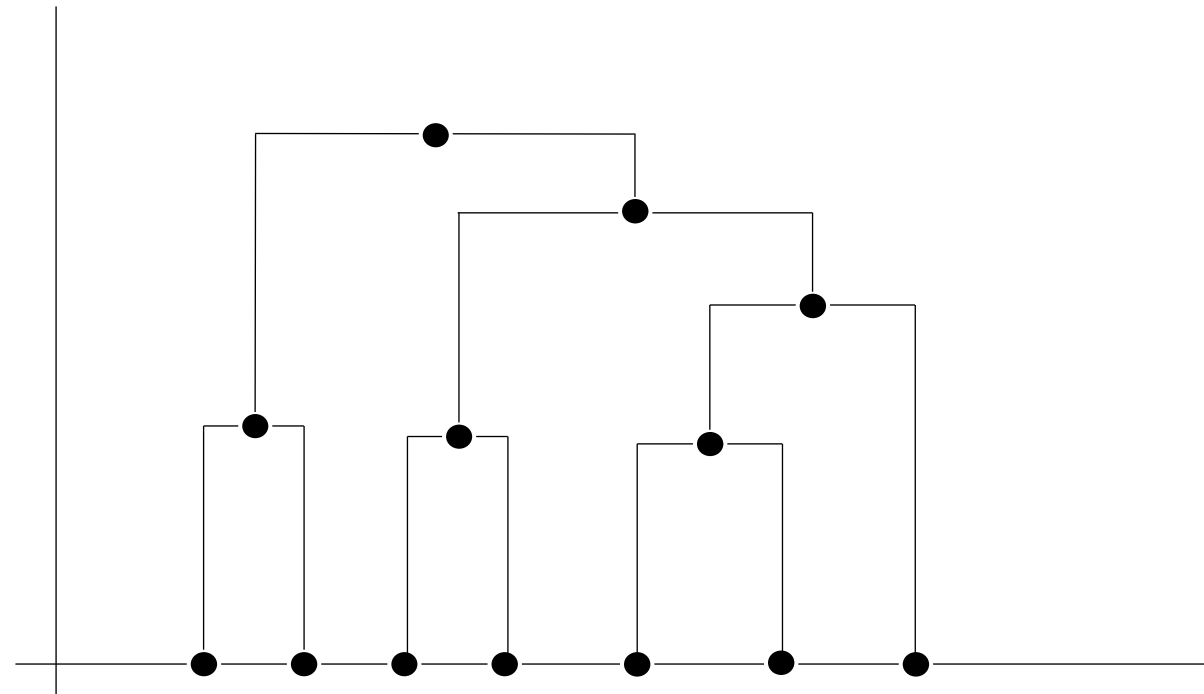


Figure. Création du dendrogramme

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Identification des microservices :

1. Parcours en profondeur du dendrogramme.
2. À chaque nœud :
 - Si le nœud a un meilleur score que la moyenne des fils, on s'arrête et il devient un microservice.

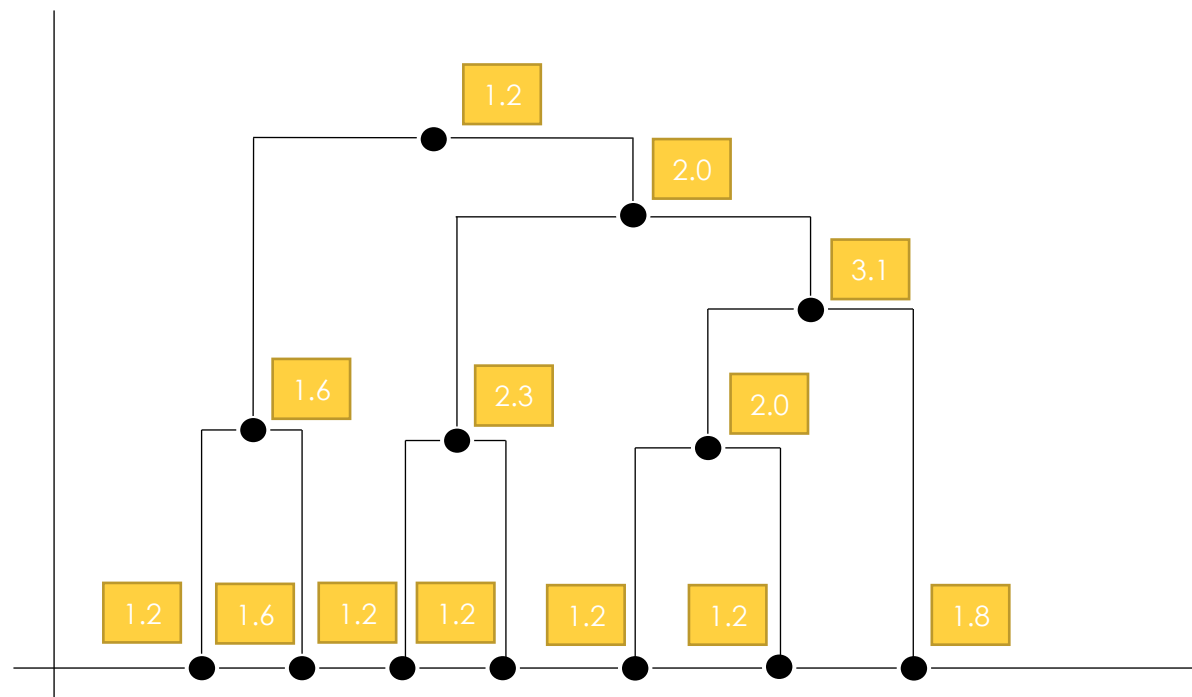


Figure. Création du dendrogramme

1.2 - Score

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Identification des microservices :

1. Parcours en profondeur du dendrogramme.
2. À chaque nœud :
 - Si le nœud a un meilleur score que la moyenne des fils, on s'arrête et il devient un microservice.

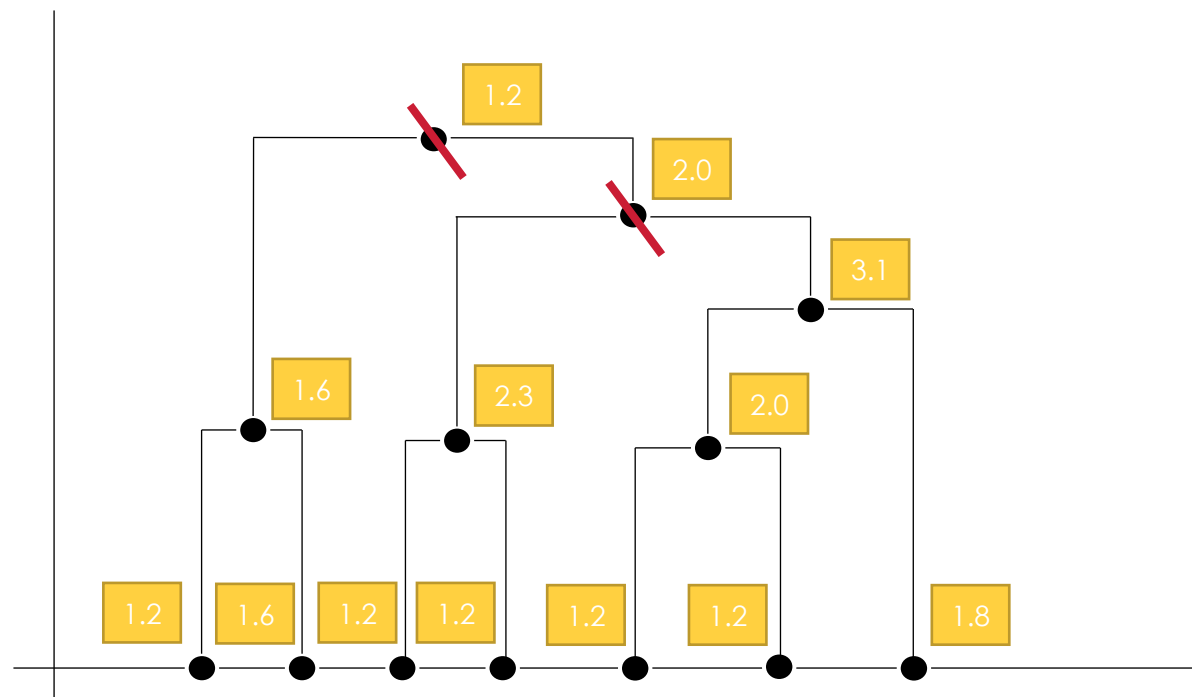


Figure. Création du dendrogramme

1.2 - Score

● - Nœud de notre dendrogramme

Identification de microservices : Regroupement Hiérarchique

Identification des microservices :

1. Parcours en profondeur du dendrogramme.
2. À chaque nœud :
 - Si le nœud a un meilleur score que la moyenne des fils, on s'arrête et il devient un microservice.

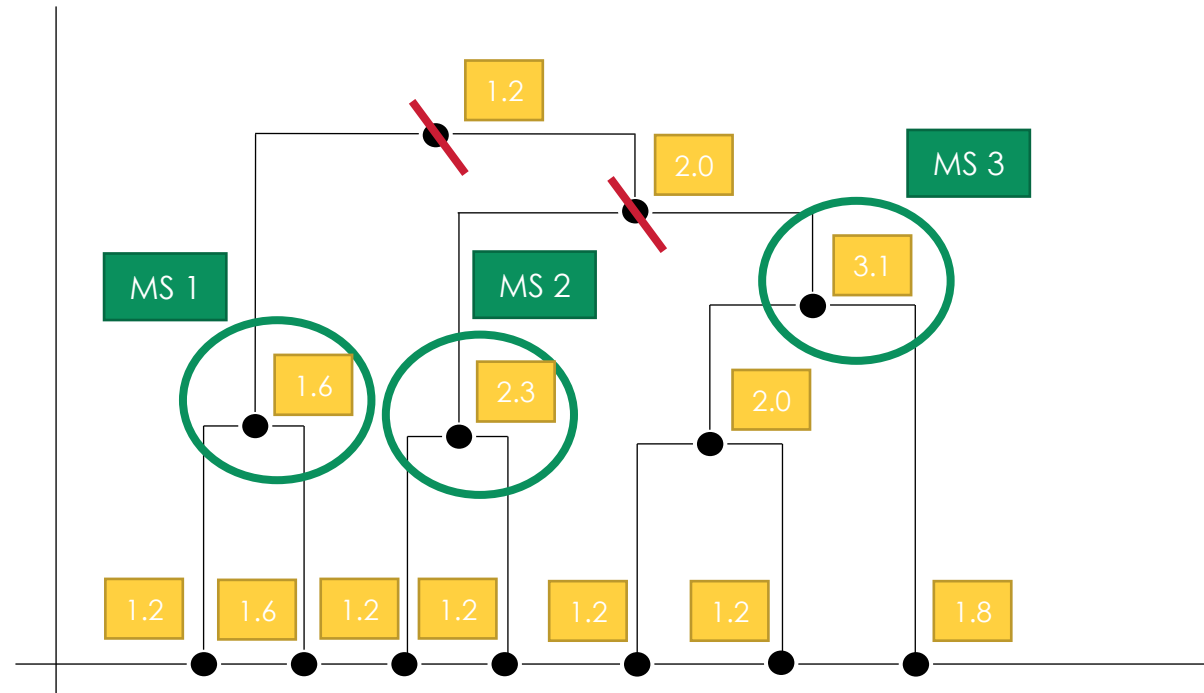


Figure. Création du dendrogramme



- Microservice



- Score

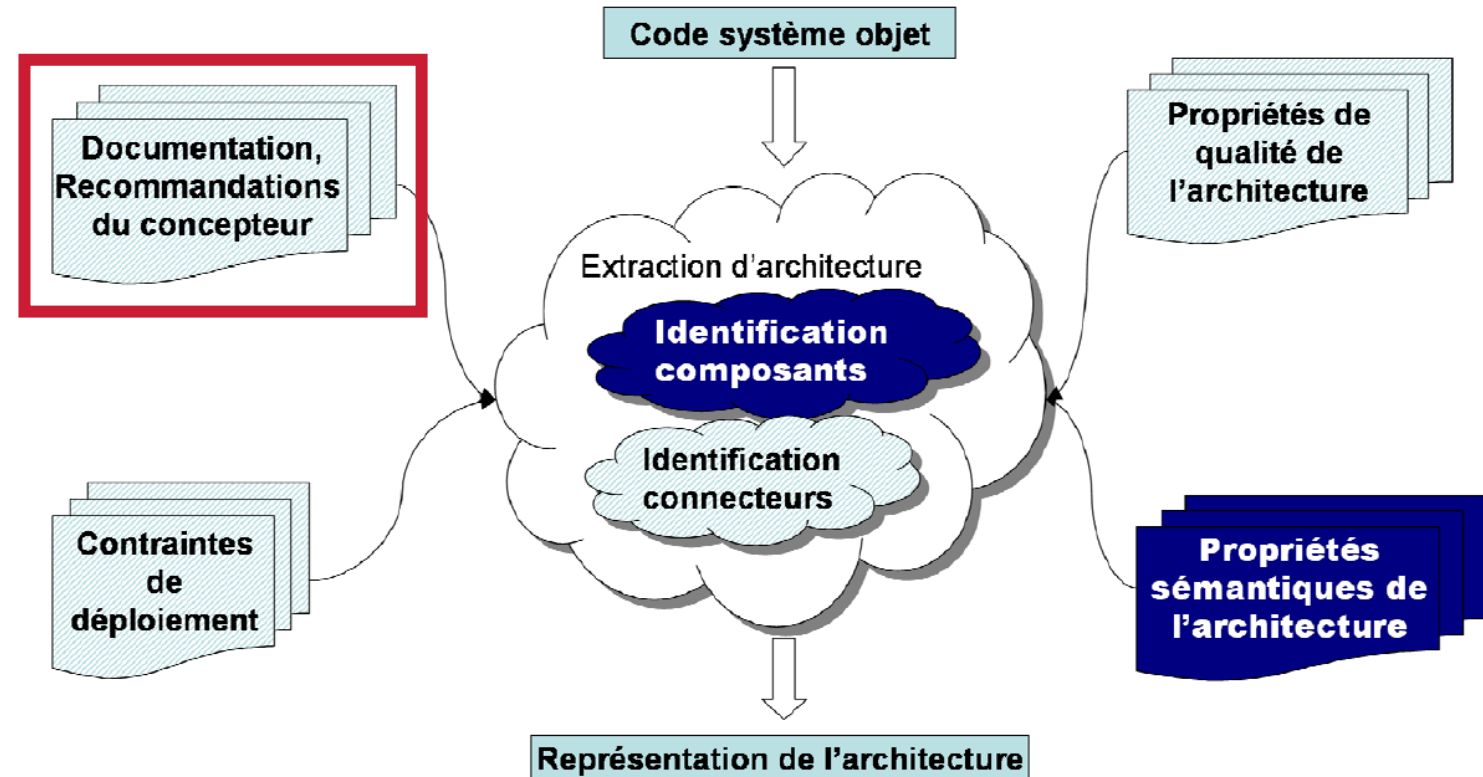


- Nœud de notre dendrogramme

Toutes méthodes de clustering automatiques peuvent être augmentées via un ensemble de ressources :

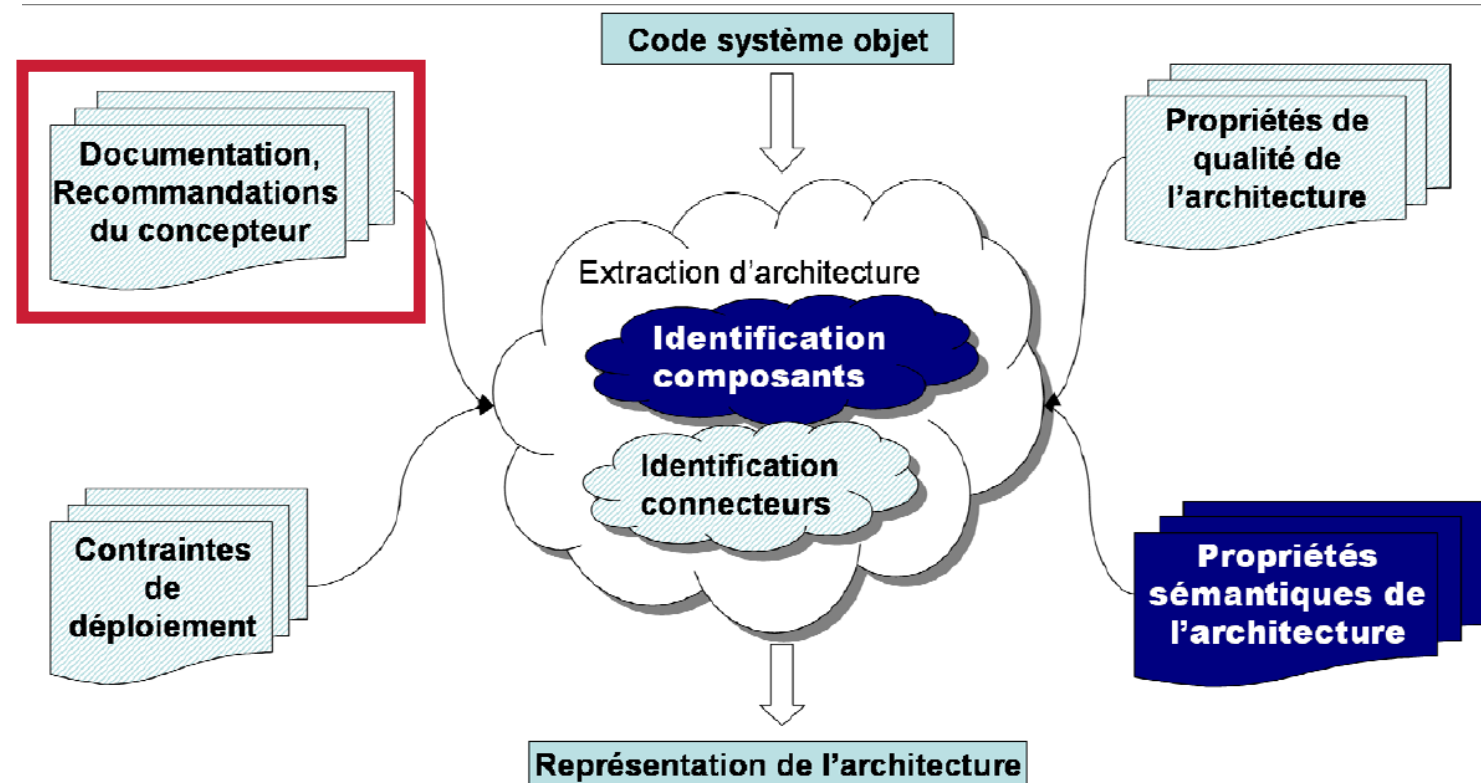
- Recommandations d'experts
 - Nombre de features
 - Centre de clusters
 - Nombre de microservices
- Documentation : UML, commentaires, sémantique du code, etc...
- Etc..

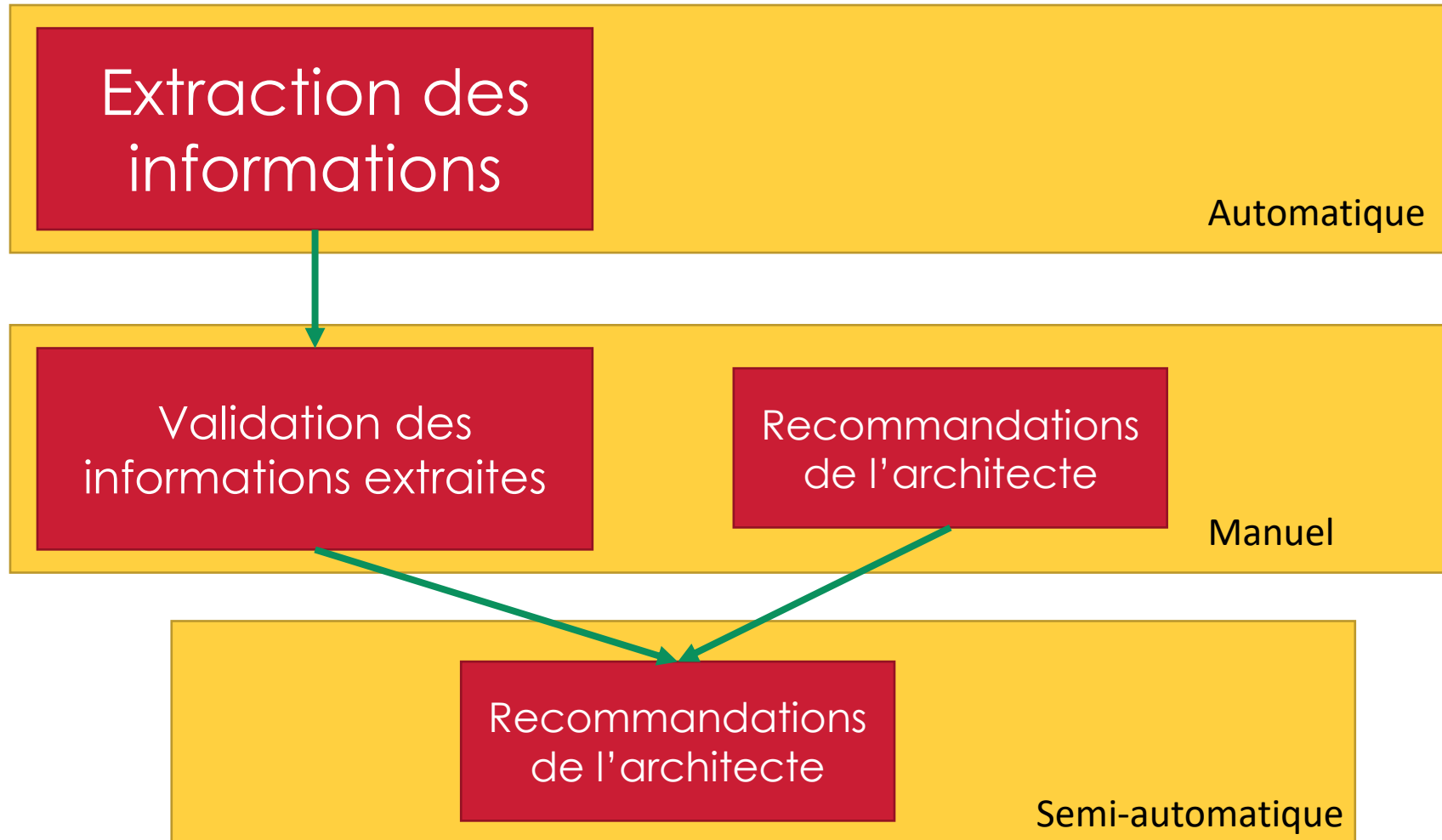
Pour déterminer le point de départ des approches à base d'exploration, pour diriger l'exploration, réduire l'espace des solutions, etc...



Pour chaque hiérarchie obtenue

- Étape 1 : l'architecte peut
 - *Modifier le résultat*
 - *Ajouter/supprimer des informations*
 - *Valider les informations sans modification*
- Étape 2 : l'architecture peut ensuite
 - *Sélectionner une partition à partir d'une hiérarchie*
 - *Lancer la sélection par défaut puis valider/réfuter*



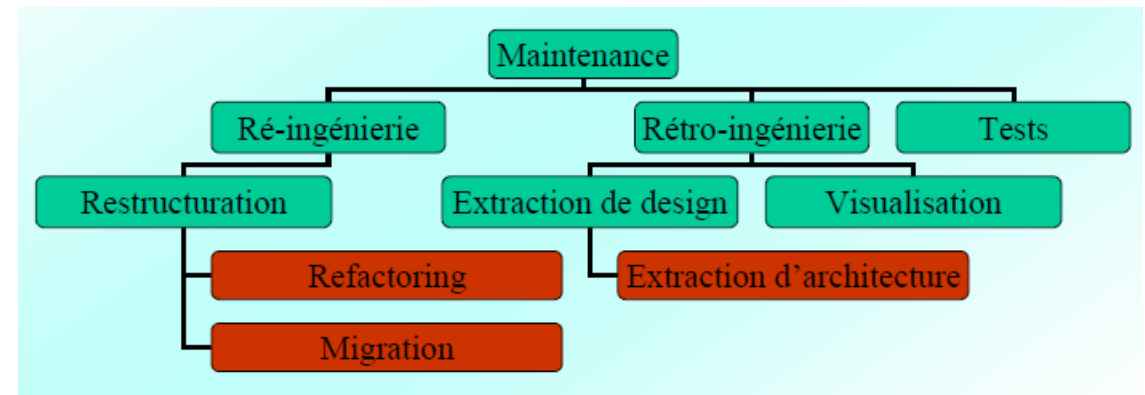


Nous avons vu :

- Réingénierie via une migration
- Rétro-ingénierie via l'extraction d'architectures
- Comment extraire une architecture à partir d'une application orienté-objet en utilisation une technique de machine learning.

Pour le TD/TP:

- Nous allons revoir le regroupement hiérarchique via un exemple simple avec Spoon sur l'application RestSuite.





Il est temps de faire une pause

Rendez-vous à 9h45 😊

