Master II Informatique
Parcours : Génie logiciel

Évolution dirigée par la qualité des architectures à base de composants

Chouki.TIBERMACINE@lirmm.fr
Maître de conférences en informatique

http://www.lirmm.fr/~tibermacin/ens/hmin306/

Plan du cours

- **Partie 1 :** Évolution des architectures à composants
- **Partie 2 :** Évolution des architectures à services

Plan du cours

- **Partie 1 :** Évolution des architectures à composants
- **Partie 2 :** Évolution des architectures à services

Contenu du cours

- Évolution et maintenance du logiciel (à base de composants)
  - Définitions et mots clés
  - Classification des activités
  - Coûts et facteurs
  - Processus d'évolution
- Assistance à l'évolution des logiciels à base de composants
  - Problématique
  - Solution apportée
  - Implémentation de la solution
- Conclusion

Contenu du cours

- Évolution et maintenance du logiciel (à base de composants)
  - Définitions et mots clés
  - Classification des activités
  - Coûts et facteurs
  - Processus d'évolution
- Assistance à l'évolution des logiciels à base de composants
  - Problématique
  - Solution apportée
  - Implémentation de la solution
- Conclusion

Définitions[1]

- **Évolution :** un processus de changement continu d'un état simple, inférieur ou mauvais vers un état plus complexe, supérieur ou meilleur
- **Maintenance :** le fait de garder une entité dans un état d'entretien, d'efficacité ou de validité pour prévenir une panne ou un déclin
- **Logiciel :** les programmes, la documentation et les procédures opérationnelles avec lesquels les systèmes informatiques peuvent être utiles à l'humain

[1]P.A. Grubb, A.A. Takang. "*Software Maintenance: Concepts and Practice*" (2ème édition). World Scientific Publishing Company (2003)

Définitions -suite-

- **Maintenance du logiciel :**
  - Modification d'un produit logiciel après sa livraison[1]
  - ”*The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment*”
    [IEEE Std 1219 - 1998]
  - ”*The software product undergoes modification to code and associated documentation due to a problem or the need for* improvement. The objective is to modify the existing software *product while preserving its integrity*” **[ISO Std 12207 - 1995]**

[1]P.A. Grubb, A.A. Takang. "*Software Maintenance: Concepts and Practice*" (2ème édition). World Scientific Publishing Company (2003)

Maintenance du logiciel

- La maintenance des logiciels est à la fois **préventive** et **curative**

- Elle comprend des activités à la fois techniques (tests, modification de code, etc.) et managériales (estimation de coûts, stratégie de gestion des demandes de modification, planification, transfert de compétences, etc.)

Évolution du logiciel

- Un terme éminemment ambigu
  - "System dynamics" pour les uns
  - "Part of maintenance" pour certains
  - "All" pour d'autres

- Un terme à la mode
  - renvoie une image plus positive
  - évoque des aspects évolutionnistes et suggère l'existence d'une "théorie" à découvrir pour les logiciels

Évolution du logiciel : vue « System dynamics »

- Changement essentiel dans le logiciel (ou une famille de logiciels) dans le temps

- La biologie a la théorie scientifique et les résultats empiriques, mais l'évolution est un phénomène observable partout
- Vers une théorie de l'évolution pour le logiciel :
  - Travaux de Lehman et Ramil[1] dans le cadre du projet FEAST

[1] M M Lehman and J F Ramil, ”Towards a Theory of Software Evolution - And Its Practical Impact”. Int. Symp. on the Principles of Software Evolution (ISPSE'00)

Évolution du logiciel : vue « Part of »

- Évolution du logiciel est toute la vie d'un logiciel après son cycle de développement initial (ou après la première livraison du système logiciel)

- Tout changement effectué pour fixer un bug, ajouter de nouvelles fonctionnalités, modifier des fonctionnalités existantes est considéré comme une évolution du logiciel

Domaines de recherche

- *System dynamics, modeling changes over time*
- *Software Process*
- *Software Metrics and Software Quality*
- *Impact Analysis*
- *Program Comprehension*
- *Software Reusability*
- *Version and Configuration Management*
- *Management and people issues*
- *Reengineering and Reverse Engineering*
- Validation and Software Testing (Regression Testing)

Types d'activités dans l'évolution et la maintenance

- Classification de Swanson[1] :
  - ***Corrective :*** [Fix program so that it meets its specs]
    - Erreurs de traitement, problème de performance ou d'implémentation
  - ***Adaptive:*** [Fix to adjust to new underlying environment/formats]
    - Changement de format de données ou d'environnement
  - ***Perfective :*** [It works fine, but let's make it even better]
    - Amélioration des performances ou de la maintenabilité

[1]Swanson, E.B., "The Dimensions of Software Maintenance".
2nd IEEE International Conference on Software Engineering (1976)

Types d'activités dans l'évolution et la maintenance

- Classification de Leintz et al.[1] :
  - ***Corrective :*** [fixing bugs to meet the initial requirements]
    - Erreurs de traitement, problème de performance ou d'implémentation (même que le précédent)
  - ***Adaptive:*** [adapting to new environment or format]
    - Pas de changement sémantique (pas d'ajout de propriétés)
  - ***Perfective :*** [any other kind of change intended to make the system “better”]
    - “Maintenance **préventive**” (*redesign*/*refactoring*)

[1]B. P. Lientz et al. "Characteristics of application software maintenance" Communications of the ACM, Volume 21 , Issue 6 (1978)

Types d'activités dans l'évolution et la maintenance

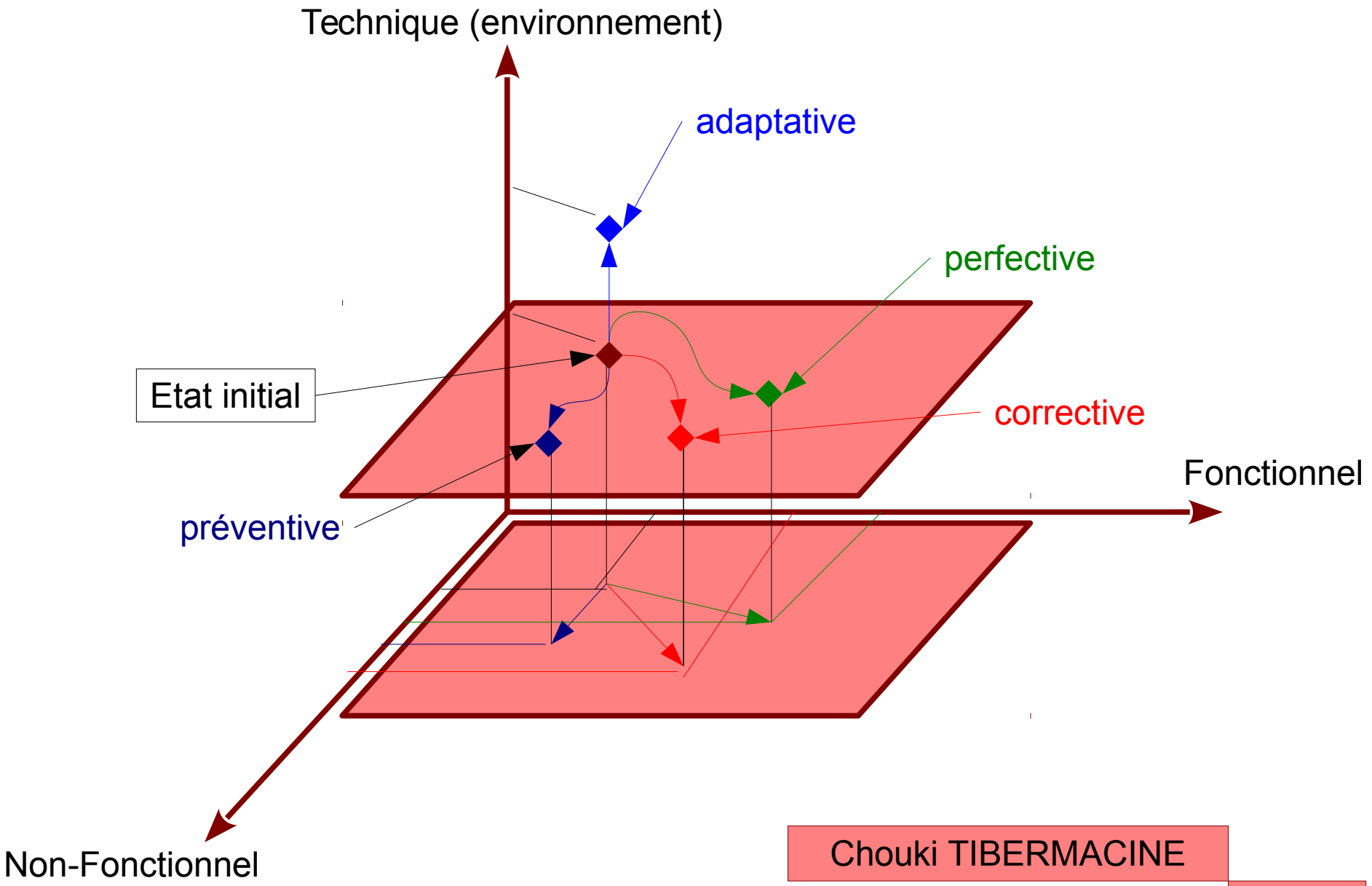
- Classification de Chapin et al.[1] :
  - *Business rules : (Evolution and Maintenance)*
    - ***Enhancive**, **Corrective** & **Reductive***
  - *Software properties :*
    - ***Adaptive** & **Perfective** (Evolution and Maintenance)*
    - ***Preventive** & **Groomative** (Maintenance)*
  - *Software Documentation :*
    - ***Updative** & **Reformative** (Maintenance)*
  - *Support interface :*
    - ***Evaluative**, **Consultive** & **Training** (Maintenance)*

[1]Ned Chapin et al. "Types of software evolution and software maintenance". Journal of Software Maintenance: Research and Practice, Vol. 13 , Issue 1 (2001)

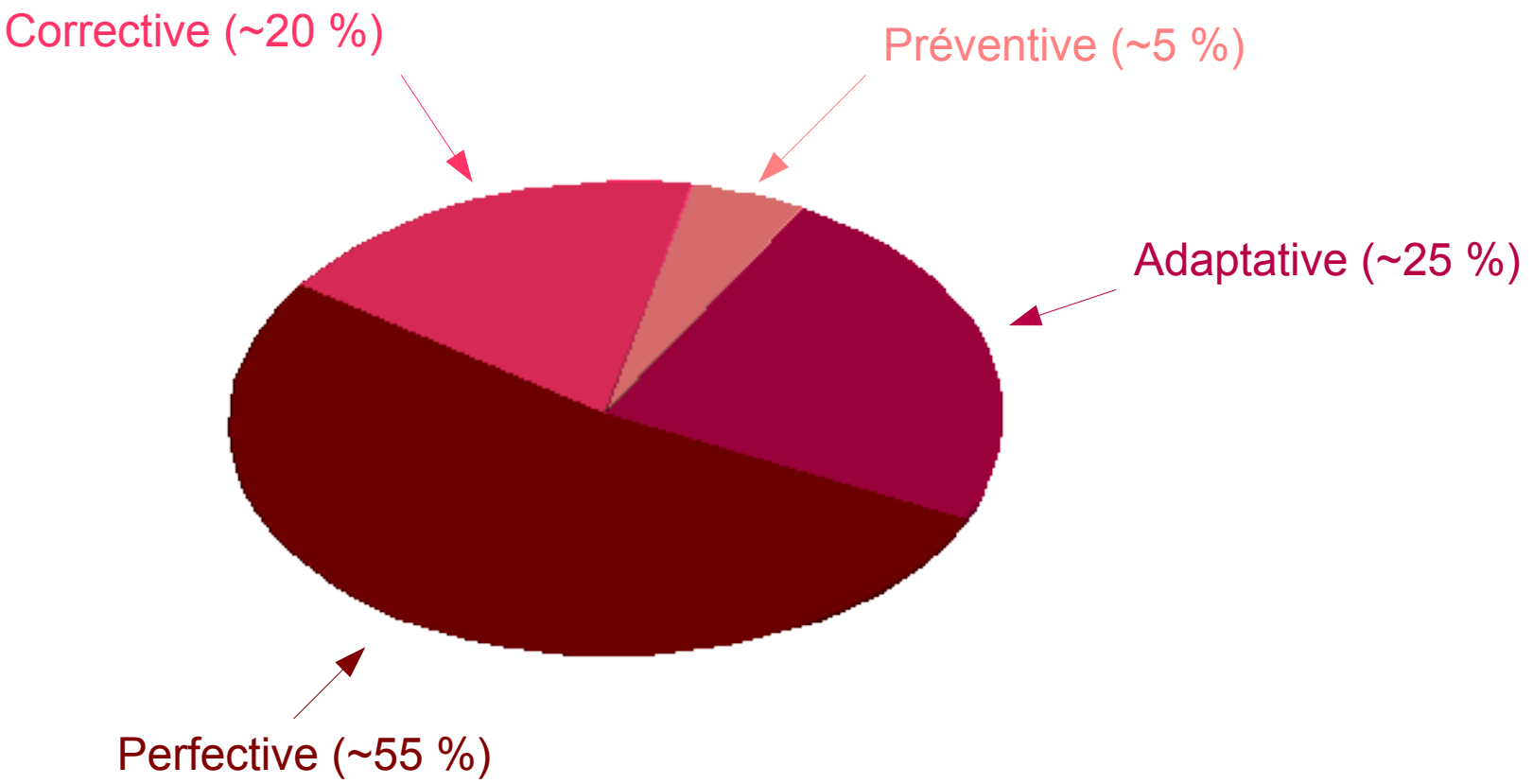
Classification « raisonnable » des activités

- **Maintenance corrective :**
  - Modifier le logiciel de telle sorte que les exigences initiales seront satisfaites
- **Maintenance adaptative :**
  - Modifier le logiciel pour qu'il supporte un nouvel environnement d'exécution ou un format de données
- **Maintenance perfective :**
  - Modifier le logiciel pour supporter de nouvelles exigences fonctionnelles ou non-fonctionnelles
- **Maintenance préventive :**
  - Modifier le logiciel pour prévenir d'éventuels problèmes futurs

Evolution du logiciel selon les axes des besoins



Distribution des activités[1]



[1]Différentes sources bibliographiques

Coût de la maintenance et l'évolution

- Largement supérieur au coût du développement
- Évolution du coût[1] :
  - Avant les années 80 : < 50 % du coût global
  - 1980 – 1990 : 65 à 75 %
  - 1991 – 2000 : plus de 75 %
  - De nos jours : > 90 %
- Ce coût élevé est affecté par des facteurs techniques et non-techniques
- La maintenance altère souvent la structure du logiciel, ce qui la rend de plus en plus coûteuse

[1] Jussi Koskinen. “Software Maintenance Costs”.

Facteurs du coût

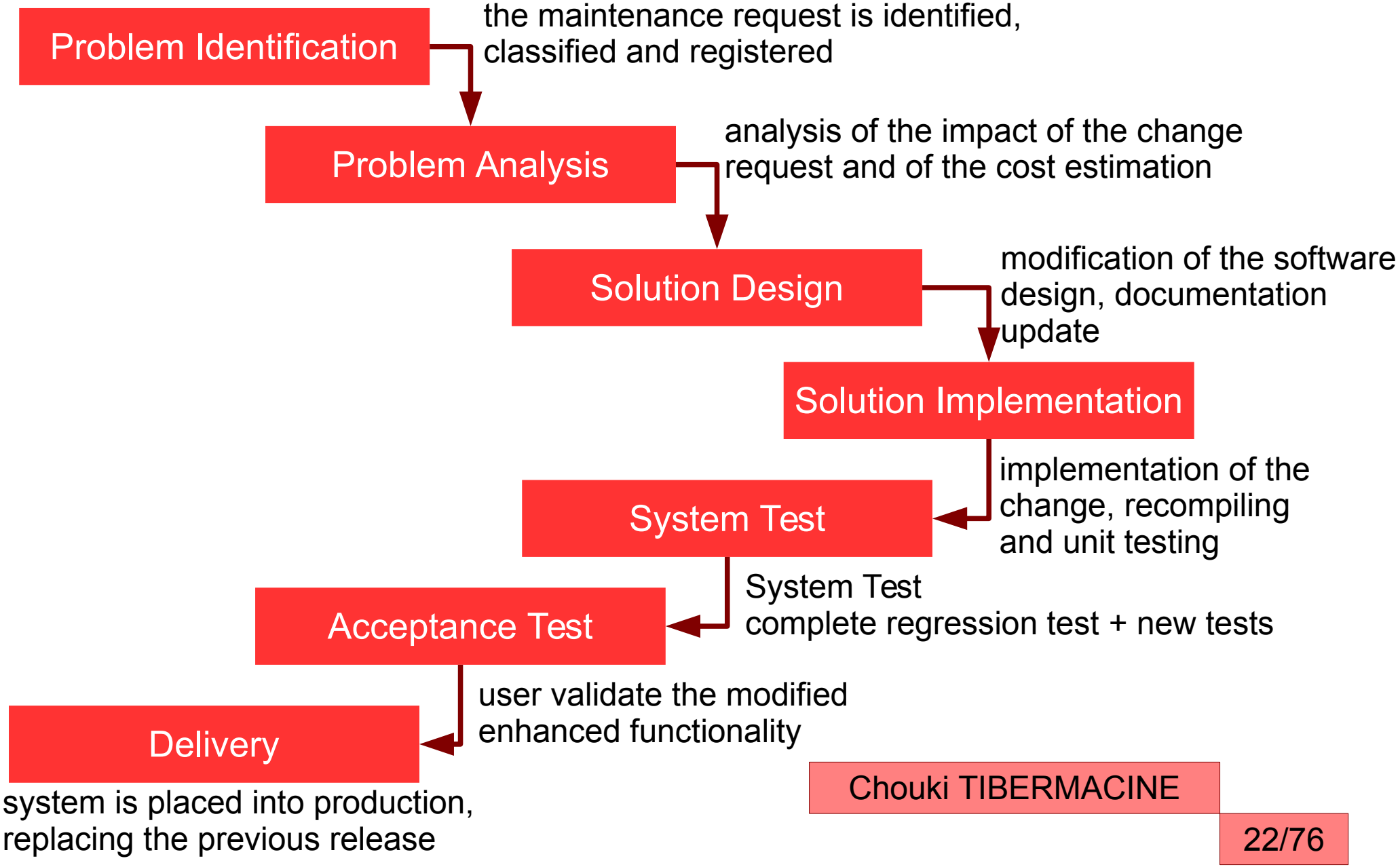
- Stabilité de l'équipe de développement
- Pression du client : mauvaise conception et documentation des changements
- Responsabilité contractualisée : pas de contrat de maintenance donc pas d'efforts pour un développement prenant en compte de futures évolutions
- Compétences des développeurs : souvent la maintenance est affectée aux personnes les moins expérimentées
- Vieillesse du logiciel (*software aging*[1]) : les logiciels vieillissent et deviennent plus difficile à comprendre et à modifier.

[1]David L. Parnas. "*Software Aging*". International Conference on Software Engineering (ICSE'94), IEEE CS Press (1994)

Engagement dans l'évolution

- Réception d’une demande interne ou externe
- Classification puis archivage de la demande
- Vérification du problème (reproduction de l’erreur, par exemple)
- Analyse de l’impact, estimation du coût et du bénéfice de la prise en compte
- Décision de poursuite, de report ou d’abandon
- Planification de la modification

Processus d'évolution : Standard IEEE 1219



Processus de changement d'architecture

- Processus à quatre phases :
  1. Appréhender l'architecture
  2. Déterminer une stratégie
  3. Réaliser la stratégie
  4. Vérifier la progression et la non-régression

- Spécialisation du processus précédent :
  - Phase 1 et 2 : Problem identification & Problem analysis
  - Phase 3 : Solution design & Solution implementation
  - Phase 4 : System test, Acceptance test & Delivery

- Approche centrée sur les architectures

1. Appréhender l'architecture

- Une description d'architecture est une abstraction sous la forme d'un graphe dont les nœuds sont des composants
- Les techniques de masquage d'information et de spécifications formelles ont permis de faciliter la compréhension d'un nœud
- Toute la difficulté est donc de comprendre en suivant les arcs ce que produit la collaboration
- Cette étape coûte à elle seule 50% des coûts de maintenance
- Les outils utilisés :
  - Toutes les techniques de la mouvance « *Software Comprehension* » et plus généralement du « *Reverse Engineering* »

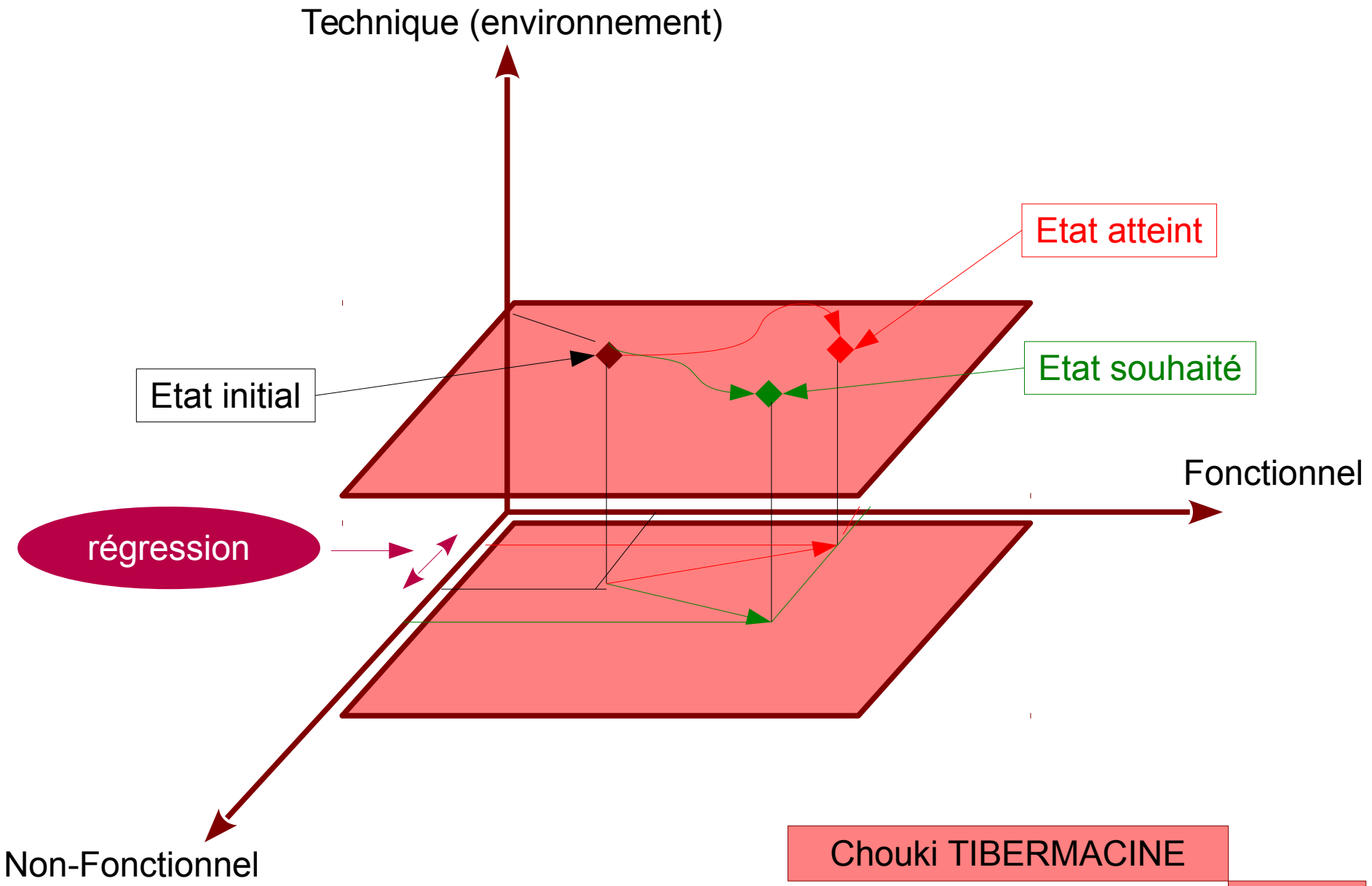
2. Définir une stratégie

- Identifier les stratégies possibles
- Déterminer des critères de choix
  - coûts, qualité, opportunité, compétences, etc.
- Déterminer des métriques pertinentes pour chacun des critères
  - évaluant des coûts de modification, des niveaux de qualité, etc.
- Mesurer les stratégies
- Choisir la « meilleure » stratégie dans le contexte
- Les outils utilisés :
  - *Change propagation* & *change impact analysis*
  - Effort and cost estimation

3. Réaliser la stratégie (actions atomiques)

- Création de nouveaux composants
- Création de nouvelles connexions (typage !!!)
- Suppression de composants existants (effet domino !!!)
- Suppression de connexions existantes (typage !!!)
- Remplacement de composants existants (substitution ou sous-typage)
- Modification de composants hiérarchiques (parcours en profondeur) :
  - Création/suppression de connexions hiérarchiques (typage !!!)
  - ...

4. Vérifier la non-régression



Tests de non-régression

- Il faut s'assurer que la modification réalisée :
  - n'a pas altéré ce qui ne devait pas l'être (non régression)
  - a bien ajouté les propriétés souhaitées (progression)
- Les challenges :
  - rejouer les tests de façon aussi automatique que possible
  - ne rejouer que les tests nécessaires
- Les outils utilisés :
  - Selective regression testing techniques

Contenu du cours

- Généralités sur la bibliographie du domaine
- Évolution et maintenance du logiciel (à base de composants)
  - Définitions et mots clés
  - Classification des activités
  - Coûts et facteurs
  - Processus d'évolution
- Assistance à l'évolution des logiciels à base de composants
  - Problématique
  - Solution apportée
  - Implémentation de la solution
- Conclusion

Problématique

- Coût de cette activité en constante augmentation (> 90 % du coût global associé à un logiciel)

- Ce qui coûte le plus cher :
  - Compréhension de l'architecture (phase 1) :
    - Activité qui coûte environ 50 % du coût de la maintenance
  - Vas et viens entre tests de non-régression (phase 4)

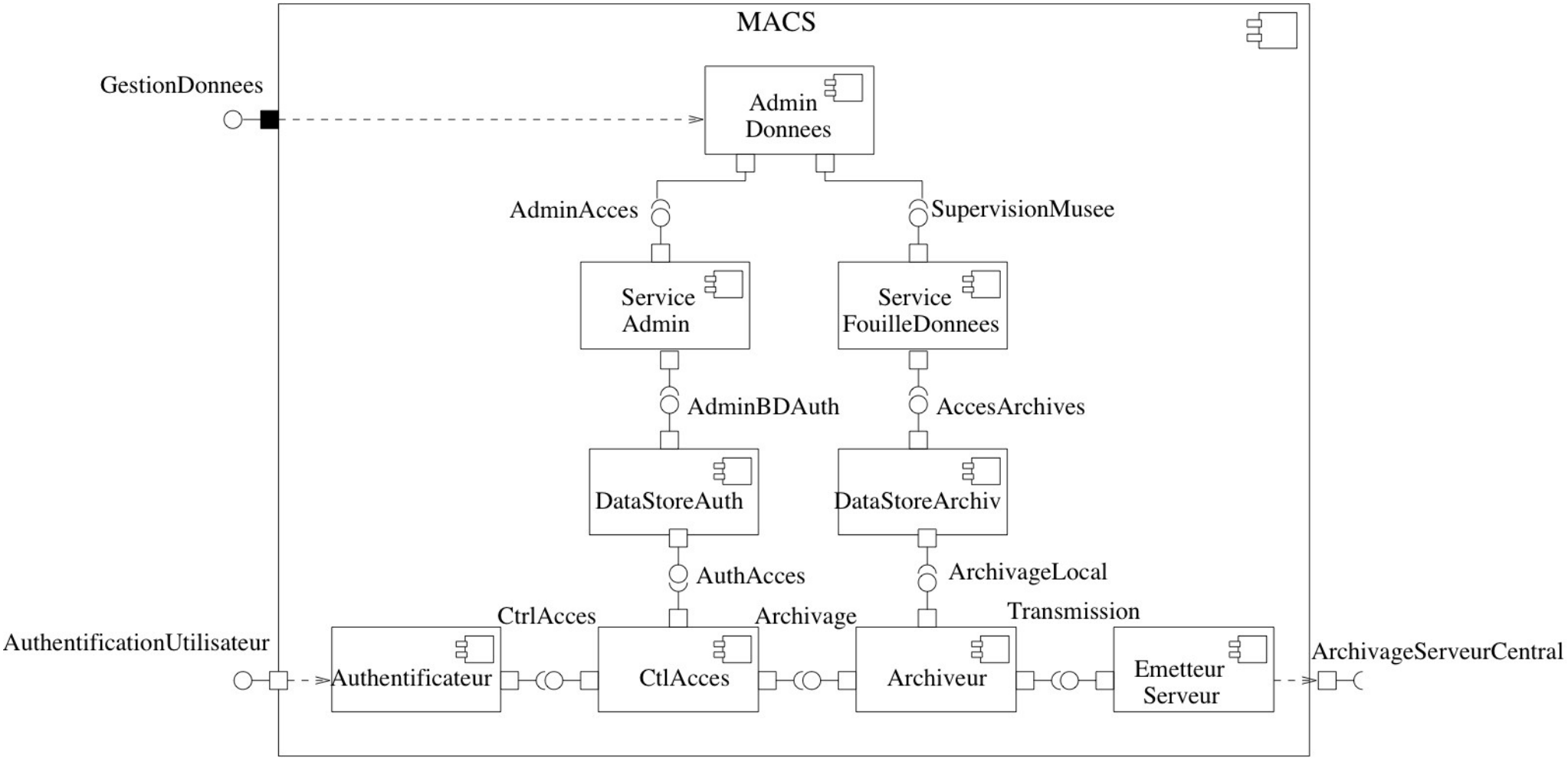
  et la définition des changements (phase 2)

Problématique à travers les lois de Lehman[1]

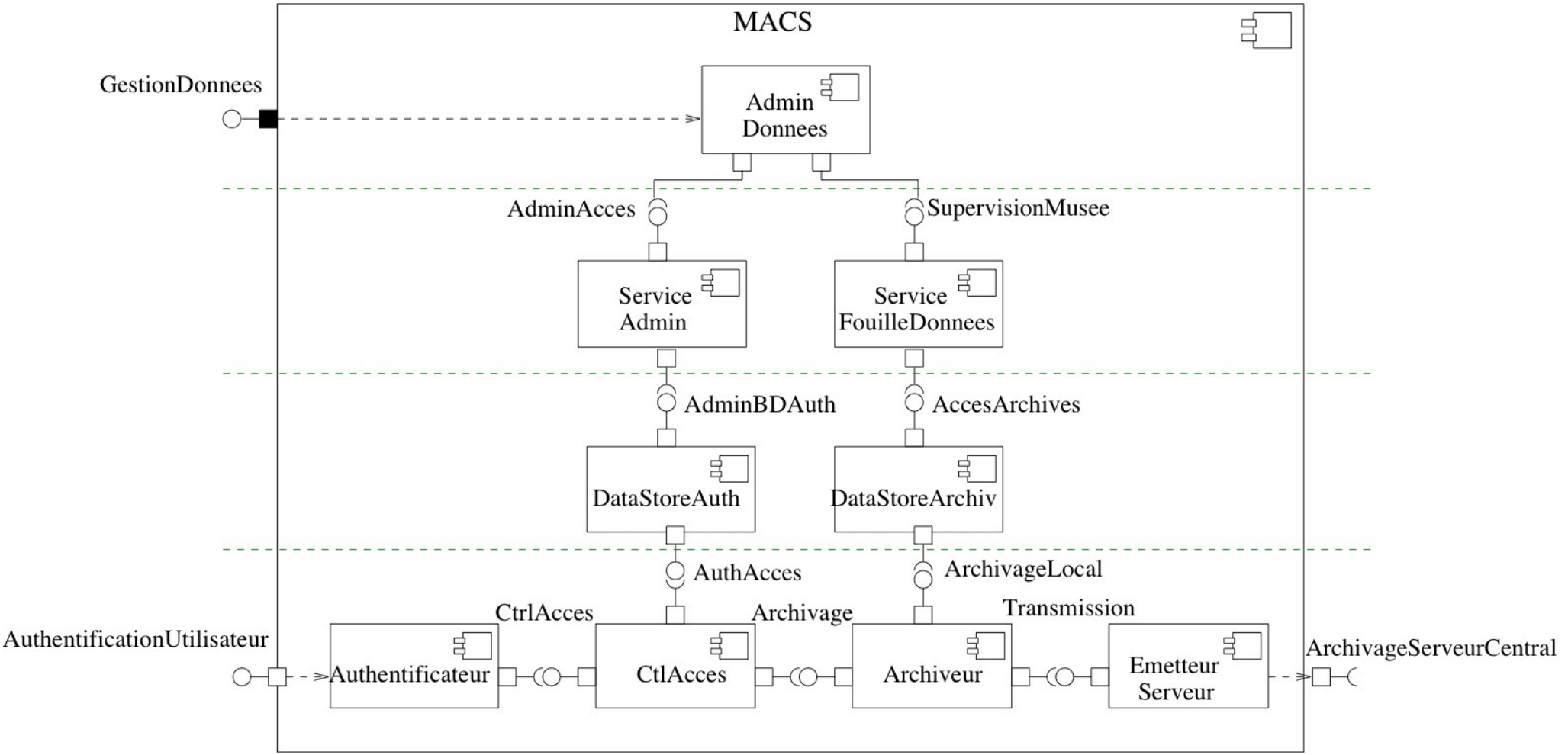
- Loi 1 : Continuing Change
- Loi 2 : Increasing Complexity
- Loi 3 : Large Program Evolution
- Loi 4 : Organisational Stability
- Loi 5 : Conservation of Familiarity
- Loi 6 : Continuing Growth
- Loi 7 : Declining Quality
- Loi 8 : Feedback System

[1]M.M. Lehman et J.F. Ramil. Software evolution in the age of component-based software engineering. IEE Proceedings – Software, vol. 147, n° 6 (2000) 249 – 255

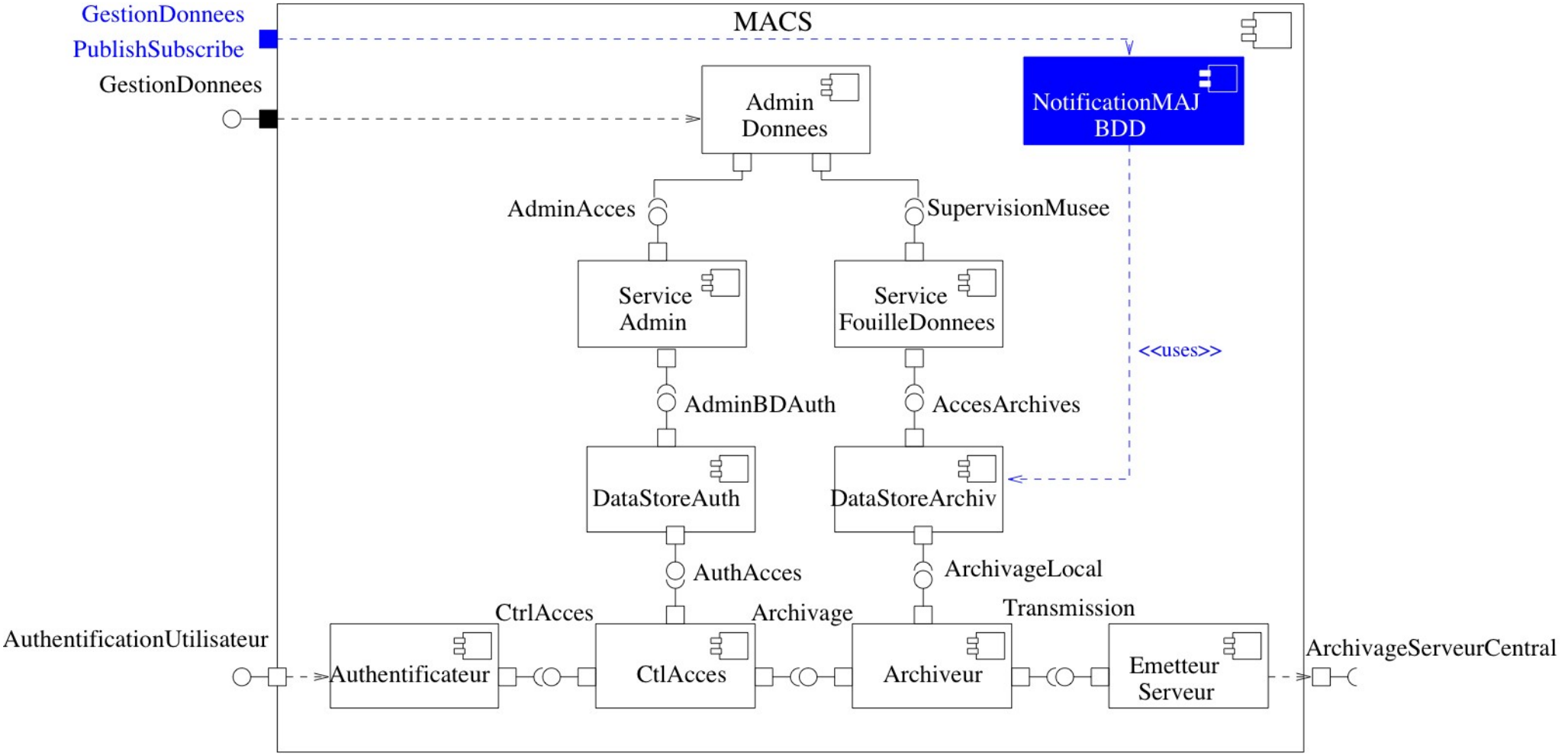
Problématique par l'exemple



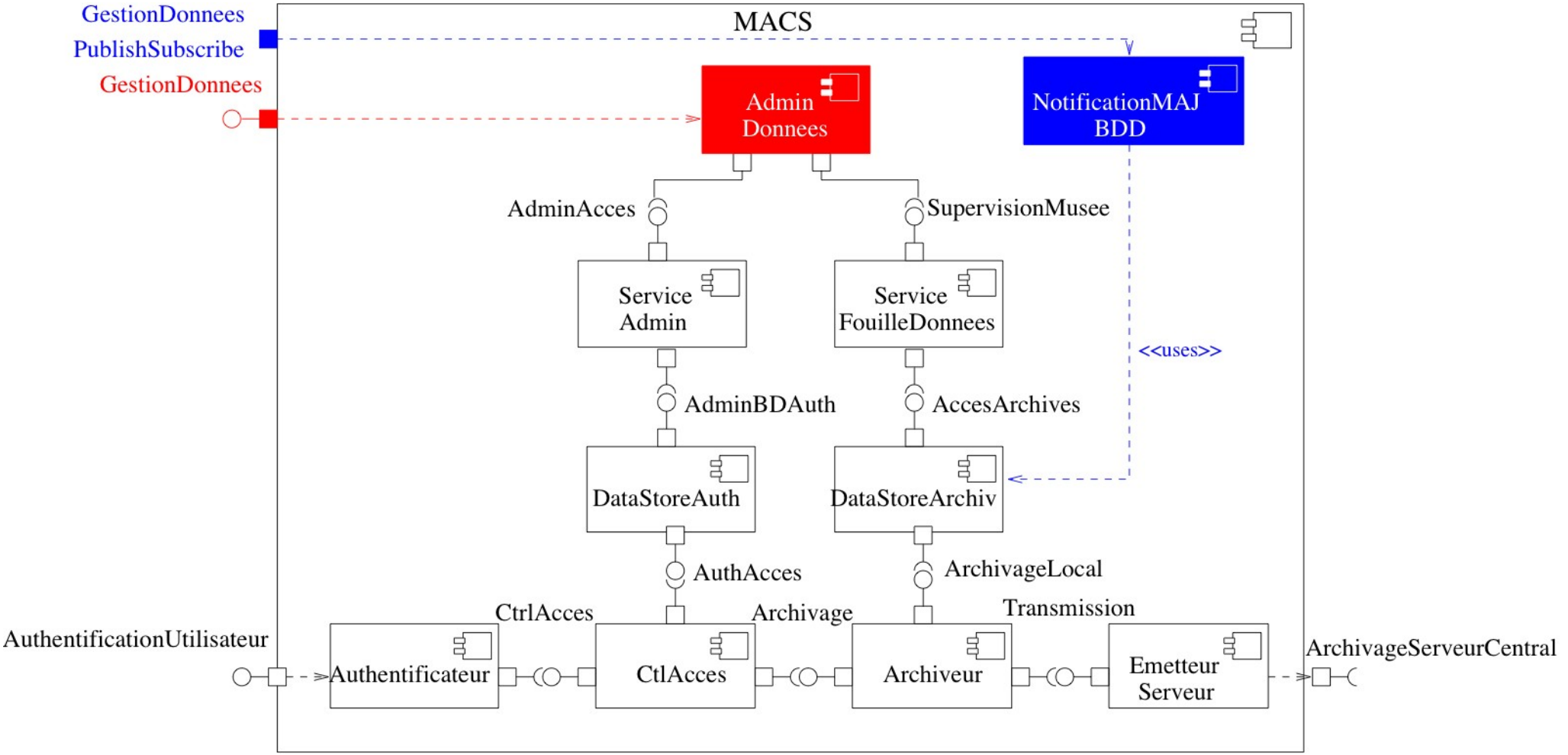
Problématique par l'exemple -suite-



Scénario d'évolution



Conséquences de l'évolution



Sources de la problématique

- Contexte de l'évolution :
  - souvent effectuée par des tiers
  - dans l'urgence
  - à distance dans le temps (perte de la connaissance initiale)

    ⇒ Maintien de la connaissance (documentation)
- Pertinence de la documentation actuelle :
  - incomplète
  - non à jour
  - ambiguë
  - lourde à manipuler (pas de support automatique)

    ⇒ Formalisation de la documentation

Solution proposée

- Documenter de manière "formelle" les décisions architecturales et les raisons de ces décisions (propriétés non-fonctionnelles) :
  - Objectif : mieux **comprendre** l'architecture lors de l'évolution

- Utiliser cette documentation pour l'assistance automatique à l'évolution :
  - Objectif : réduire le nombre d'**allers/retours** entre tests de non-régression et application des changements

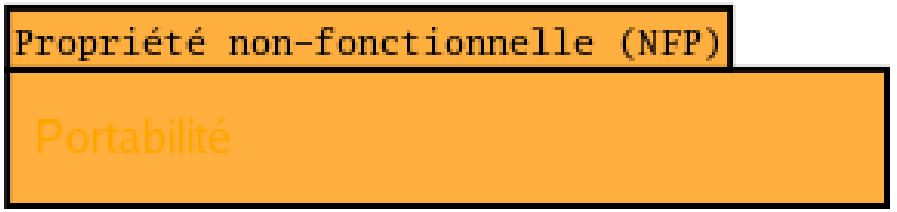
Contenu du cours

- Généralités sur la bibliographie du domaine
- Évolution et maintenance du logiciel (à base de composants)
  - Définitions et mots clés
  - Classification des activités
  - Coûts et facteurs
  - Processus d'évolution
- Assistance à l'évolution des logiciels à base de composants
  - Problématique
  - Solution apportée
  - Implémentation de la solution
- Conclusion

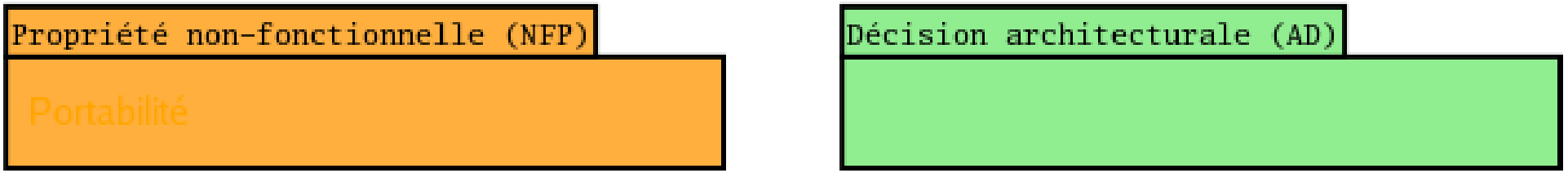
Documentation des composants

- Documentation des décisions architecturales (**AD**)
- Documentation des attributs qualité (**NFP** : *Non-Functional Property*)
- Documentation de l'association AD-NFP (NFT : *Non-Functional Tactic*)
- Documentation finale = stratégie non-fonctionnelle (NFS) :
  - ensembles des NFT associées à un composant

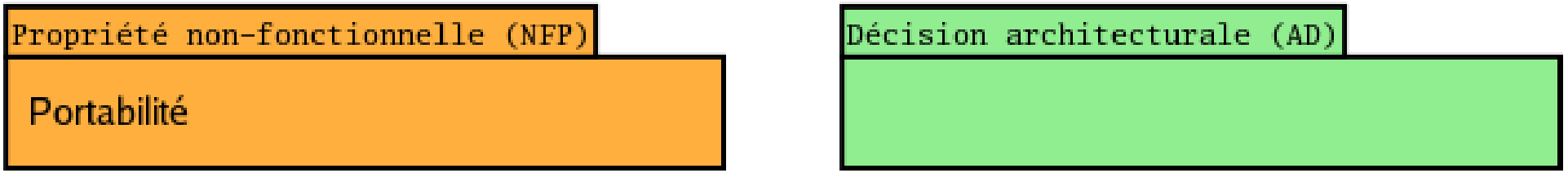
Description d'une NFS



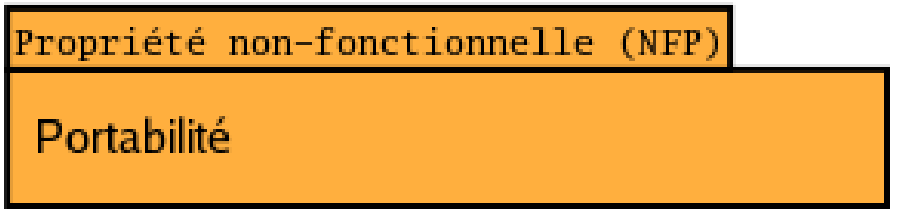
Description d'une NFS -suite-

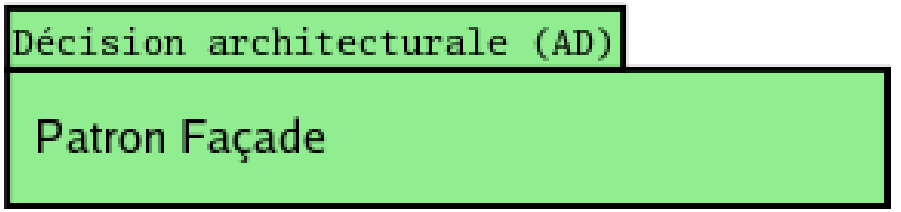


Description d'une NFS -suite-

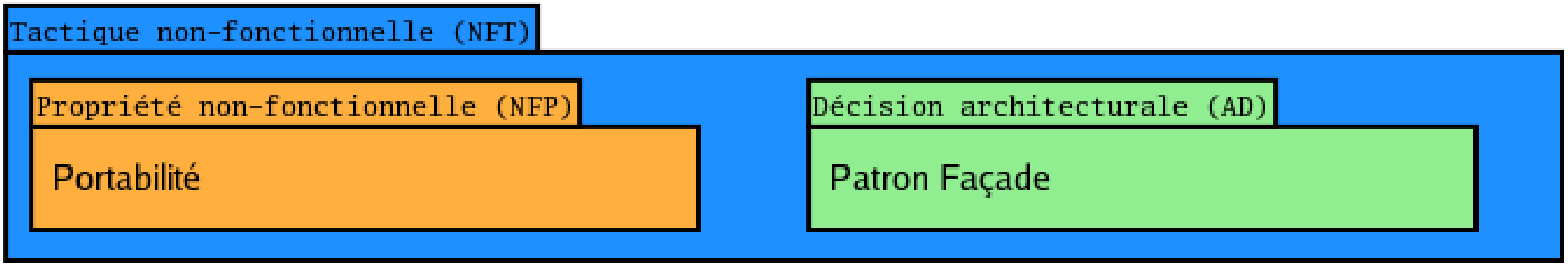


Description d'une NFS -suite-

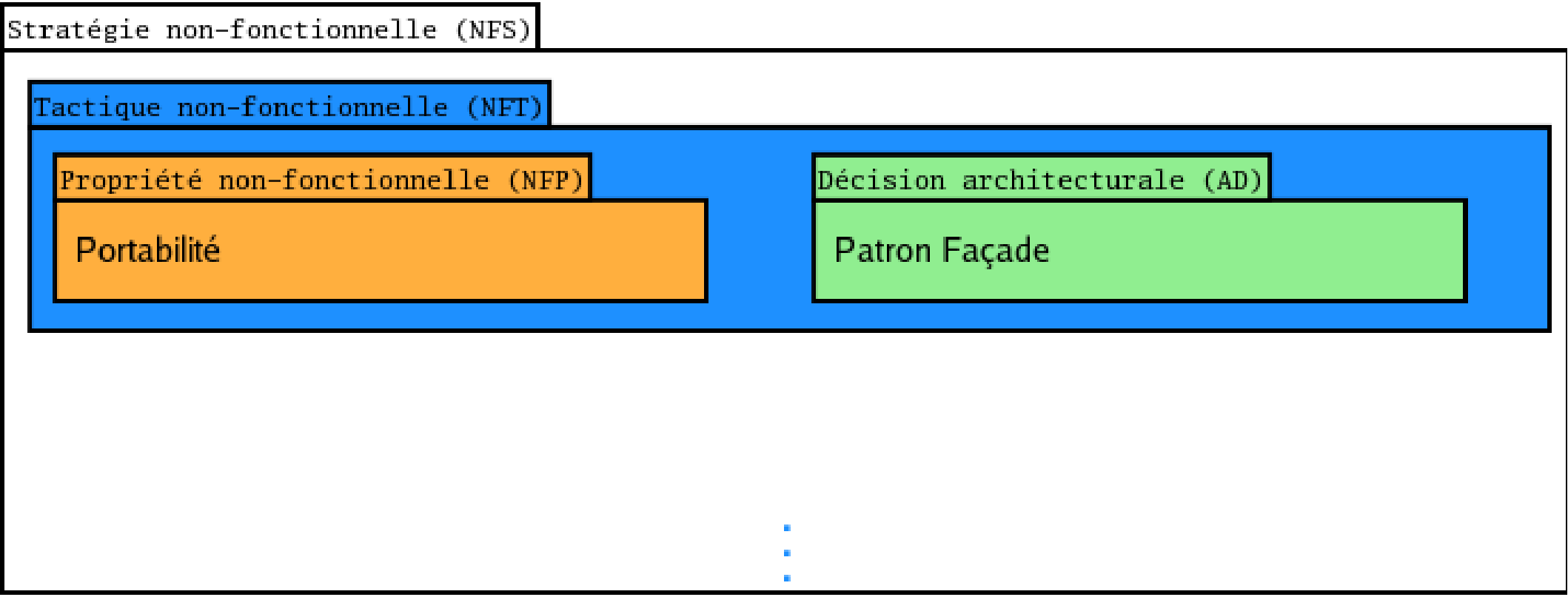




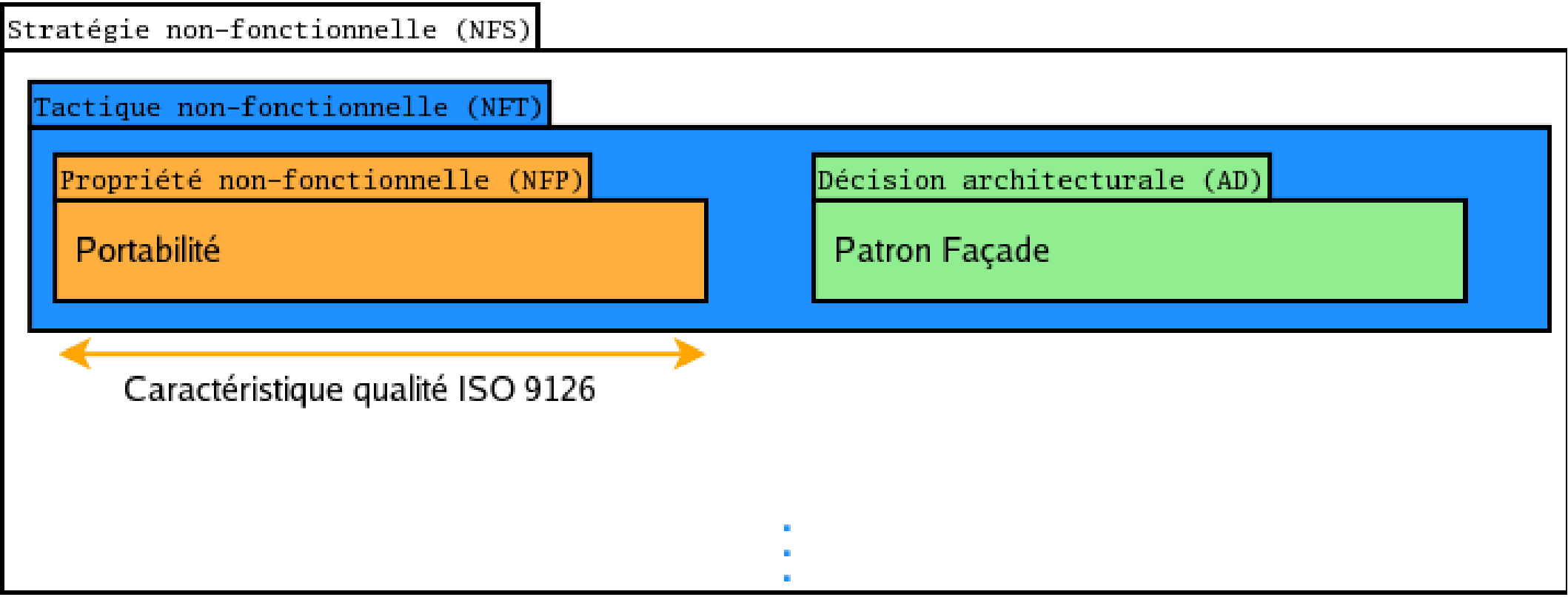
Description d'une NFS -suite-



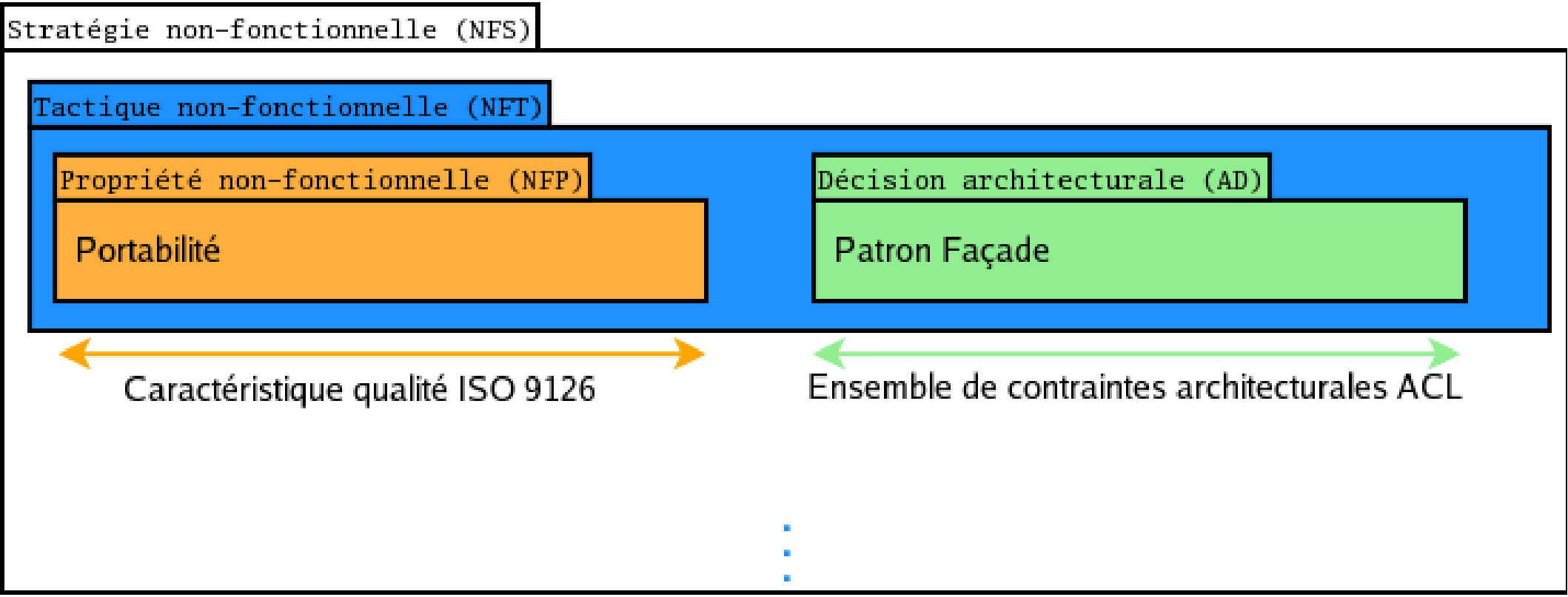
Description d'une NFS -suite-



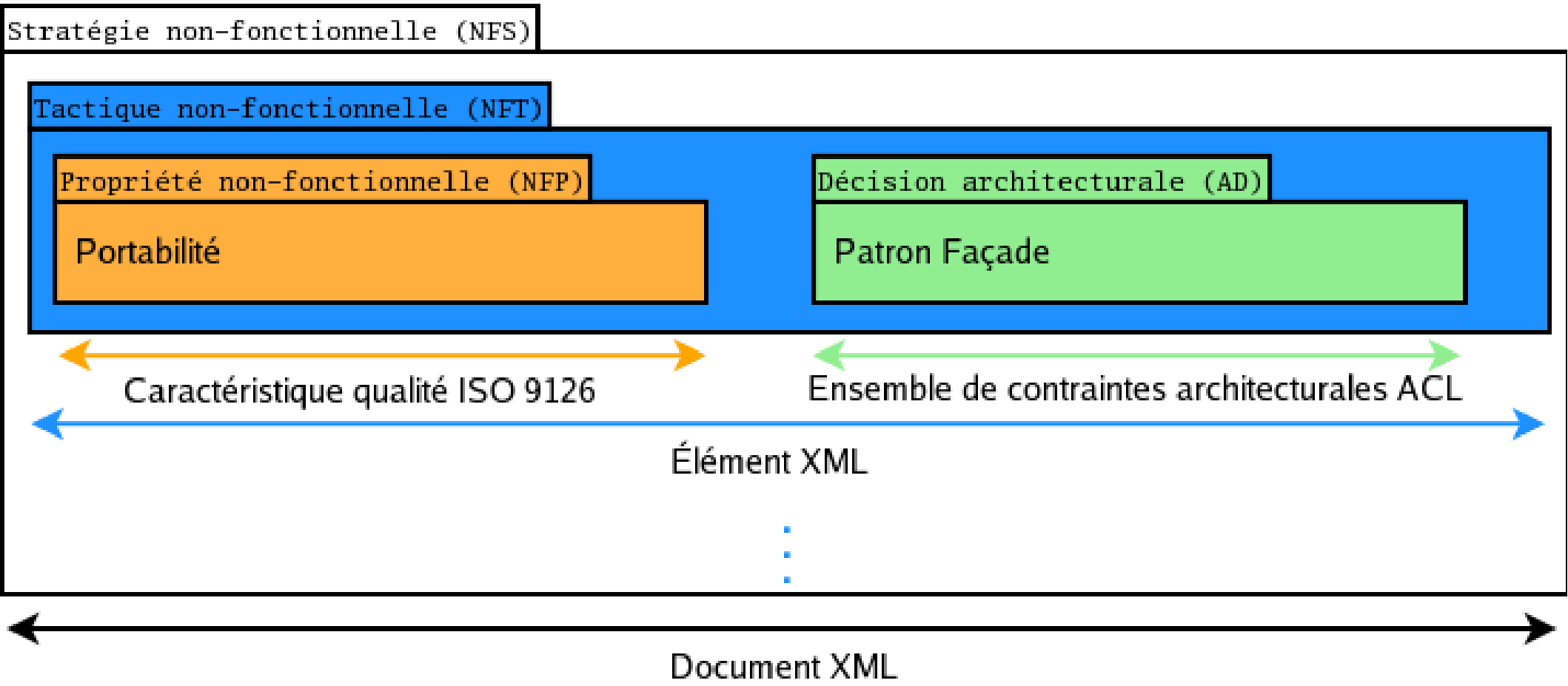
Description d'une NFS -suite-



Description d'une NFS -suite-



Description d'une NFS -suite-



Description des décisions architecturales

- Une décision architecturale = une contrainte décrite à l'aide du langage ACL
- Celle-ci peut représenter :
  - le choix d'un style/patron architectural
  - le choix d'un patron de conception
  - un invariant architectural
  - une contrainte de pure évolution (contrainte impliquant des comparaisons entre versions de composants)

Description des propriétés non-fonctionnelles

- Les propriétés non-fonctionnelles représentent les caractéristiques et sous-caractéristiques ISO 9126 :
  - ***Functionality :*** *Suitability, Accuracy, Interoperability, Compliance, Security*
  - ***Reliability :*** *Maturity, Recoverability, Fault Tolerance*
  - ***Usability :*** *Learnability, Understandability, Operability*
  - ***Efficiency :*** *Time Behavior, Resource Behavior*
  - ***Maintainability :*** *Stability, Analyzability, Changeability, Testability*
  - ***Portability :*** *Installability, Replaceability, Adaptability*
- Description d'une NFP :
  - Le nom de la sous caractéristique
  - + la ou les clauses dans les documents de spécification des exigences non-fonctionnelles

Syntaxe concrète d'une NFS

```
<nonfunctional-strategy id="000001">
  <nonfunctional-tactic id="000100">
    <description>
      Cette tactique garantit la NFP portabilité
      par le biais du patron de conception façade
    </description>
    <nonfunctional-property id="001000" name="Portabilité"
                            characteristic="Portability"
                            external-arch-artifact="MACS">
      <description>
        Le composant doit être portable sur différents
        environnements. Il peut servir différents types
        d'applications clientes.
      </description>
    </nonfunctional-property>
    ...
```

Syntaxe concrète d'une NFS -suite-

```
      ...
      <architecture-decision id="010000">
        <description>
          Patron de conception façade
        </description>
        <architecture-constraint id="100000" profile="CCM">
          <!-- Ici, on met la contrainte -->
        </architecture-constraint>
      </architecture-decision>
    </nonfunctional-tactic>
</nonfunctional-strategy>
```

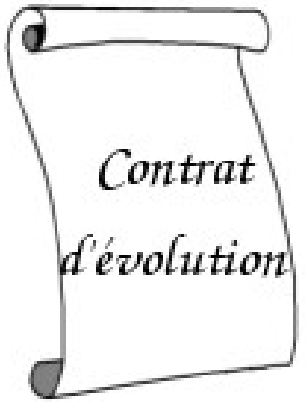
Contenu du cours

- Généralités sur la bibliographie du domaine
- Évolution et maintenance du logiciel (à base de composants)
  - Définitions et mots clés
  - Classification des activités
  - Coûts et facteurs
  - Processus d'évolution
- Assistance à l'évolution des logiciels à base de composants
  - Problématique
  - Contexte de recherche
  - Solution apportée
  - Implémentation de la solution
- Conclusion

NFS = Contrat d'évolution

Développeur de l'ancienne version

Je dois documenter les décisions architecturales afin de rendre persistante la qualité



Développeur de la nouvelle version

Je dois respecter le contrat pour que le coût de l'évolution soit minimal

Intérêt d'un contrat d'évolution

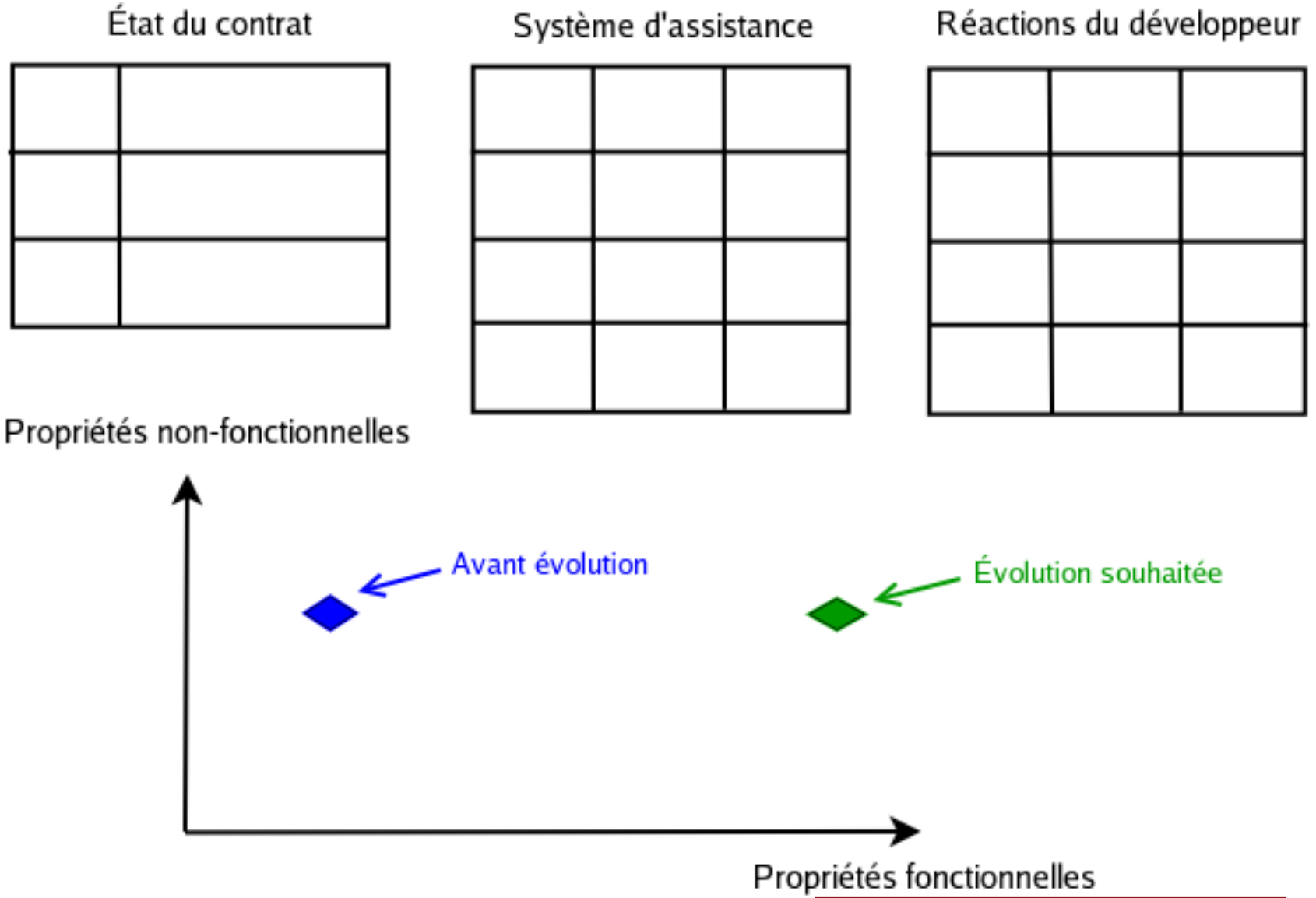
- Un contrat d'évolution est utilisé pour mieux comprendre l'architecture d'une application à base de composants :
  - permet de savoir quelles sont les décisions architecturales significatives qui ont été prises et à quel niveau
  - permet de savoir quelles sont les raisons de ces décisions (quels sont les attributs qualité concrétisés)

- Un contrat est utilisé pour assister l'activité d'évolution :
  - détecter l'impact des changements sur les décisions architecturales
  - déduire l'impact sur les besoins de qualité initiaux

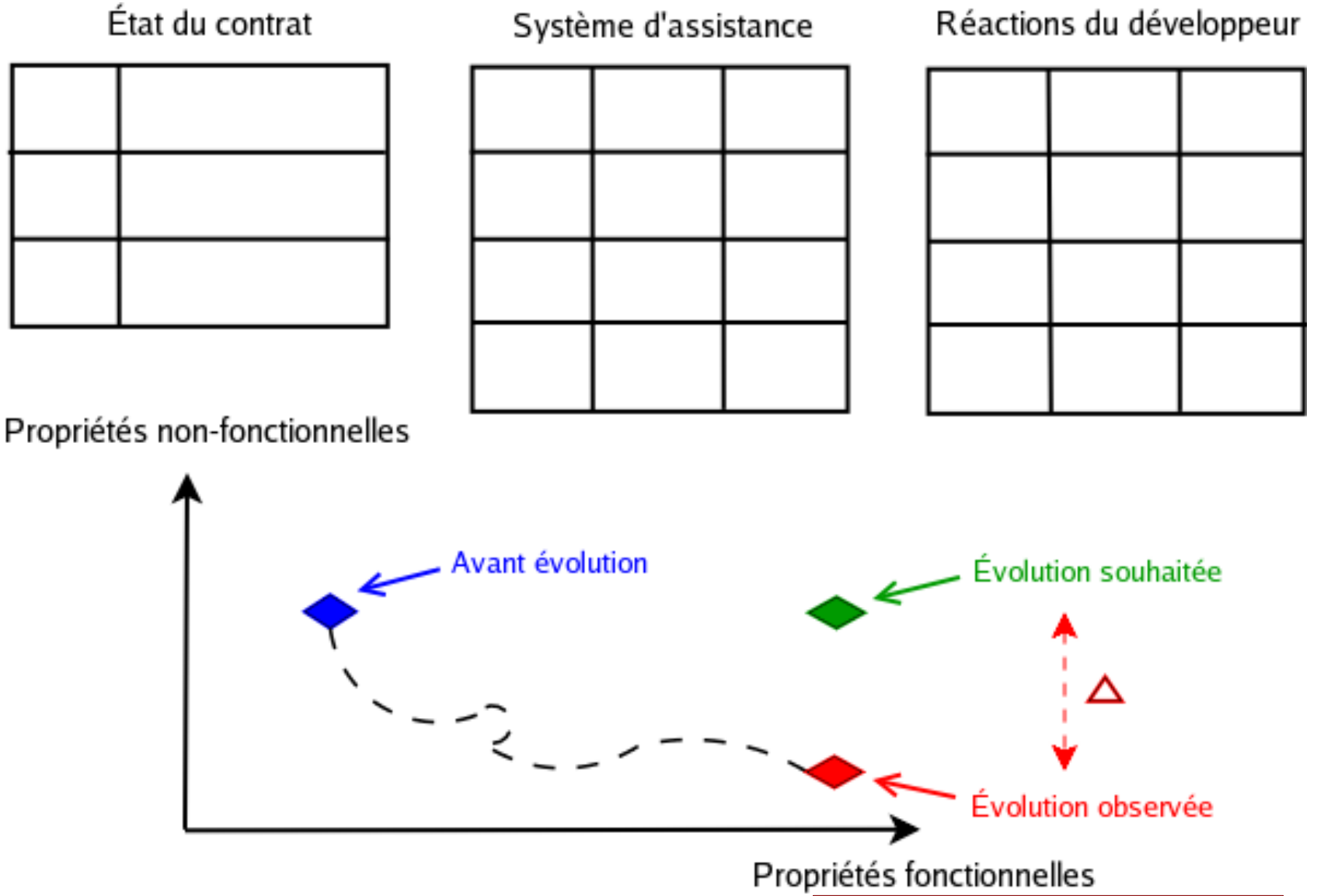
Algorithme d'assistance à l'évolution

- Des règles pour l’assistance :
  - **Règle 1 :** "un système cohérent est un système où toute propriété non-fonctionnelle est impliquée dans au moins une tactique non-fonctionnelle"
  - **Règle 2 :** "nous ne devons pas interdire un pas d'évolution. On notifie simplement une tentative d'affecter une ou plusieurs décisions architecturales et on spécifie les propriétés non-fonctionnelles altérées spécifiées dans le contrat"
  - **Règle 3 :** "on peut ajouter de nouvelles tactiques non-fonctionnelles au contrat d'évolution”

Déroulement de l'algorithme



Déroulement de l'algorithme -suite-



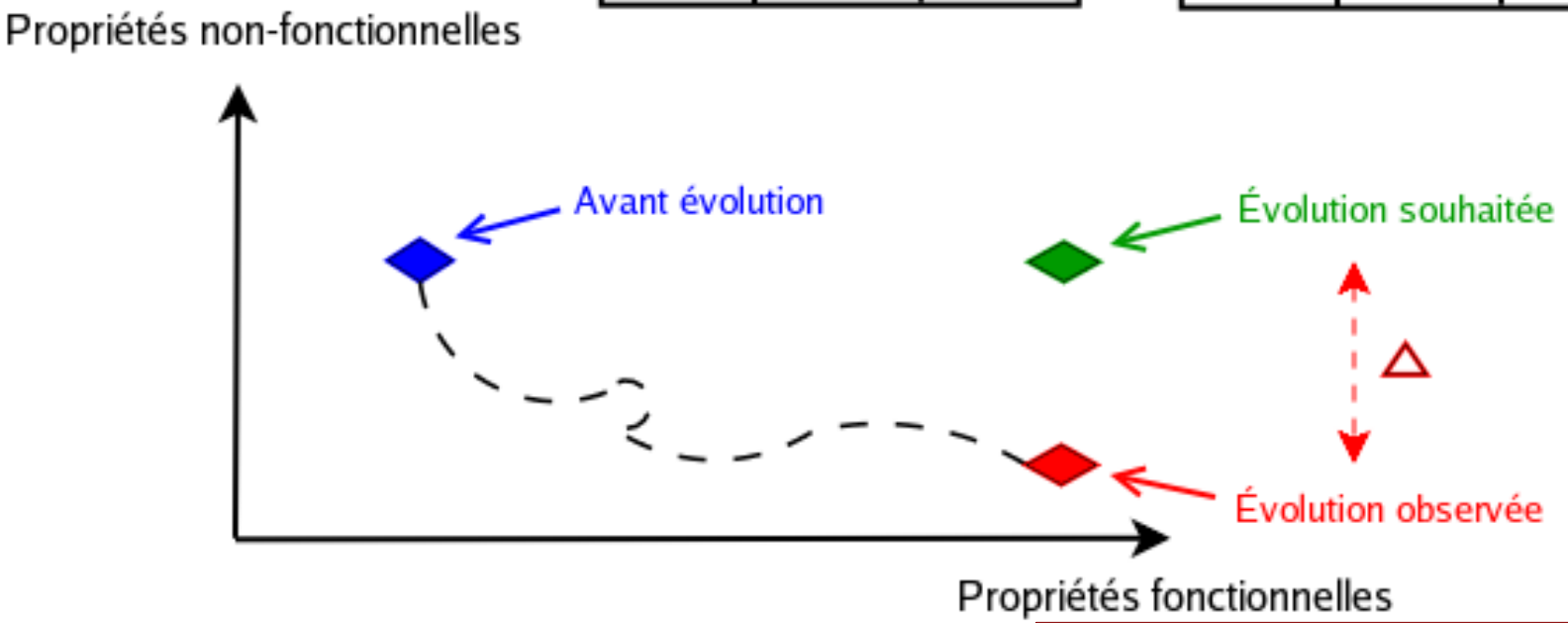
Déroulement de l'algorithme -suite-

État du contrat

NFP1	AD1 AD3 AD4
NFP2	AD2 AD3
NFP3	AD4 AD5

Système d'assistance

Réactions du développeur



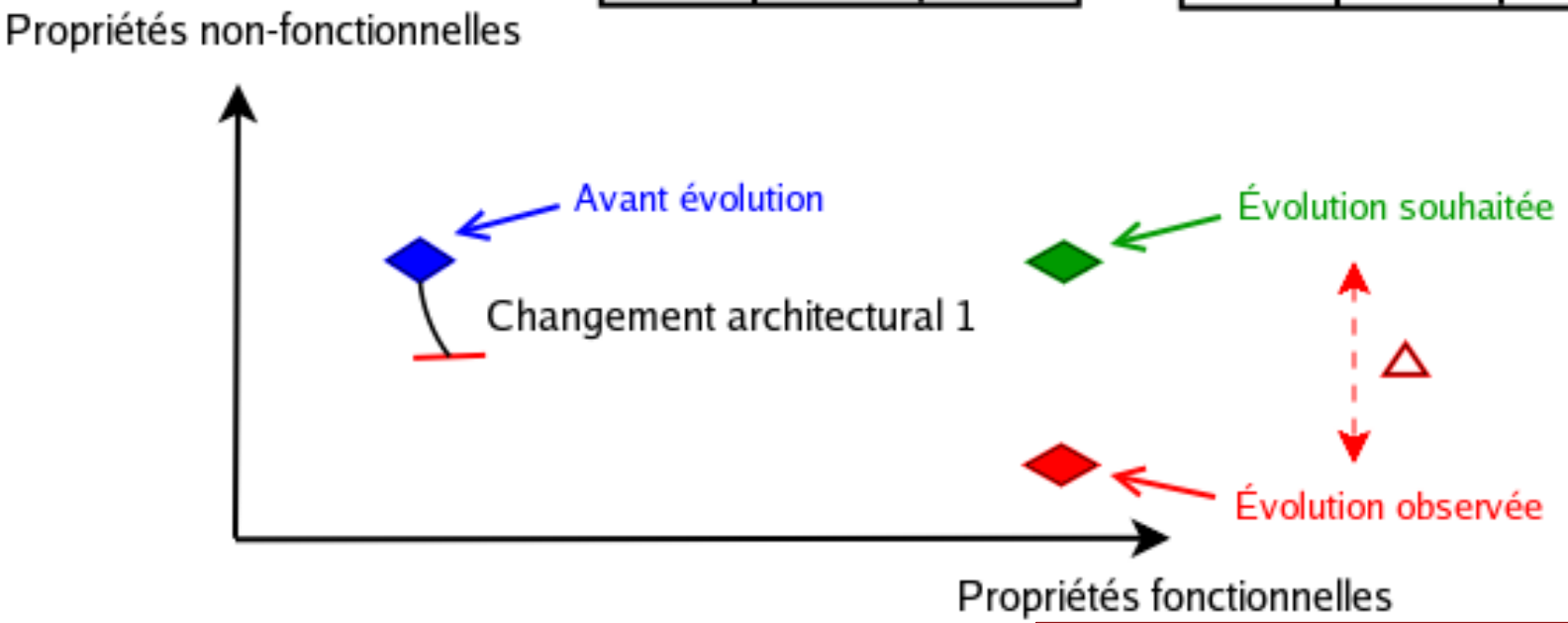
Déroulement de l'algorithme -suite-

État du contrat

NFP1	AD1 AD3 AD4
NFP2	AD2 AD3
NFP3	AD4 AD5

Système d'assistance

Réactions du développeur



Déroulement de l'algorithme -suite-

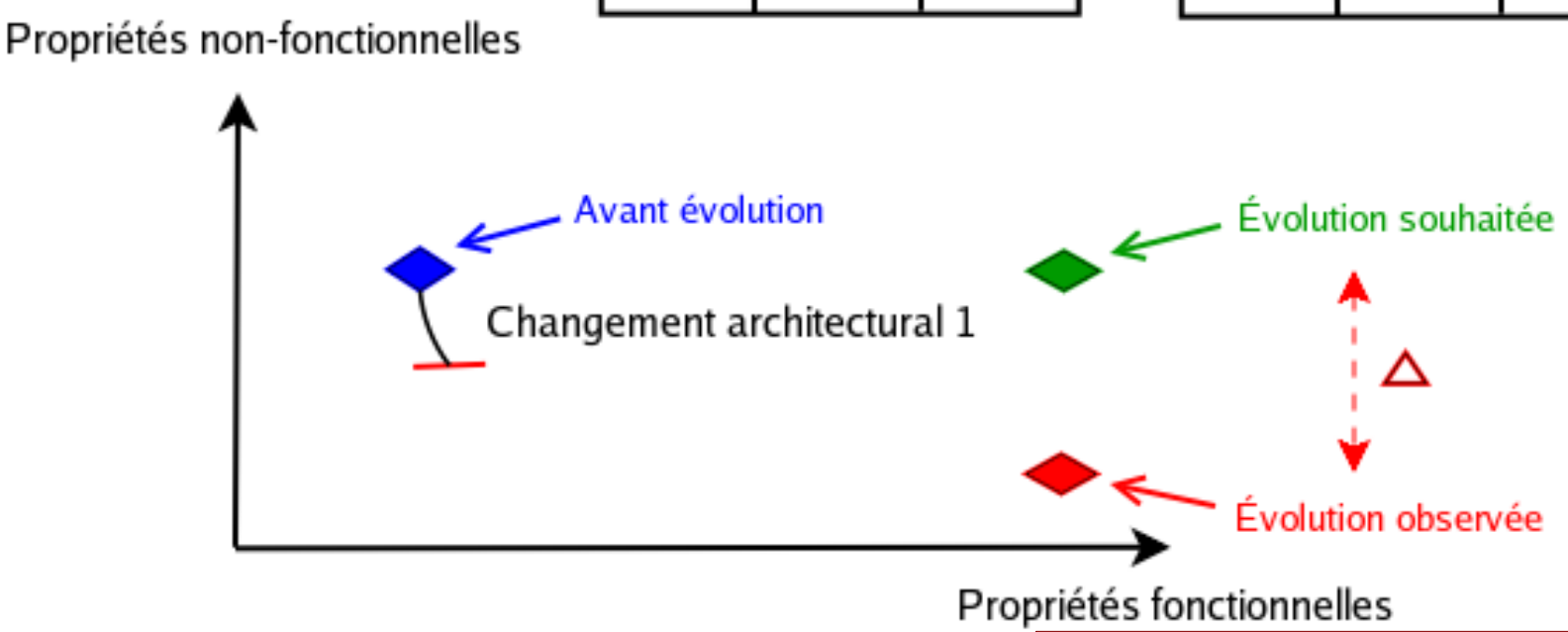
État du contrat

NFP1	AD1	AD4
NFP2	AD2	
NFP3	AD4	AD5

Système d'assistance

AD3	NFP1 NFP2	✔

Réactions du développeur



Déroulement de l'algorithme -suite-

État du contrat

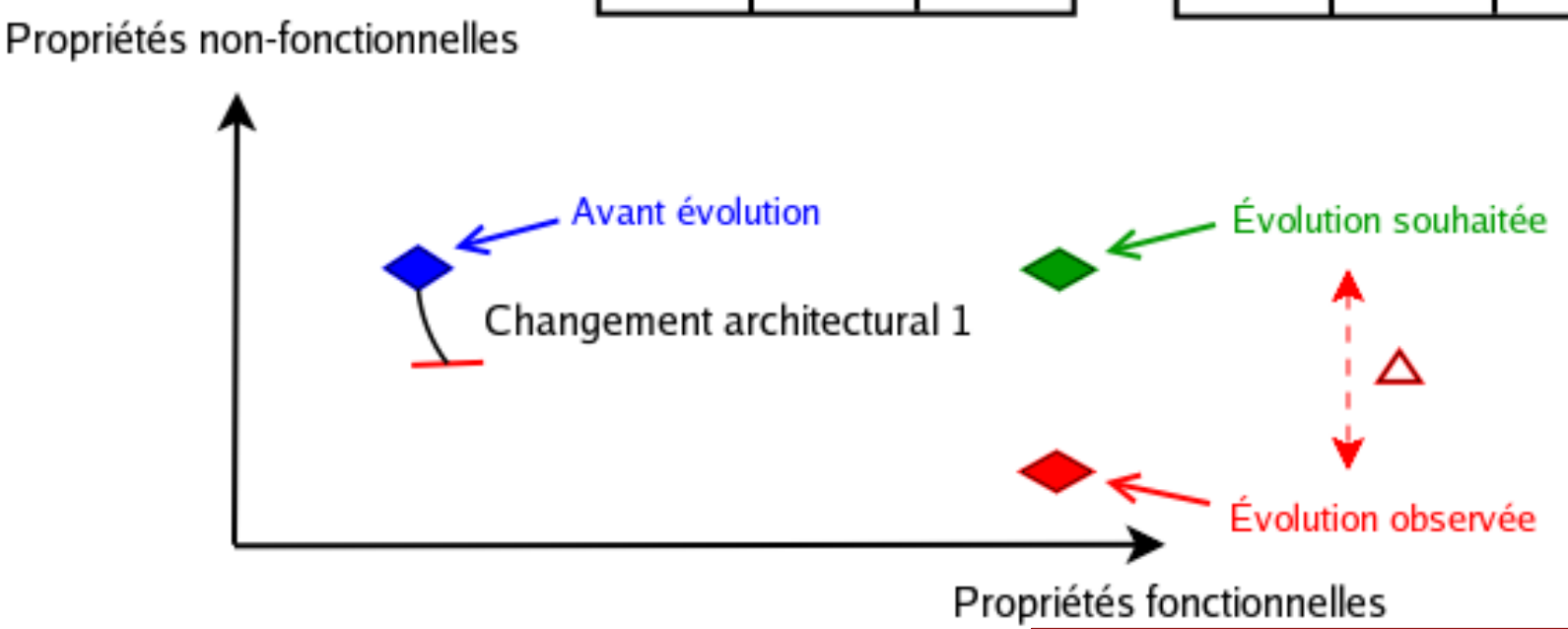
NFP1	AD1 AD4
NFP2	AD2
NFP3	AD4 AD5

Système d'assistance

AD3	NFP1 NFP2	✔

Réactions du développeur

AD3	-	↷



Déroulement de l'algorithme -suite-

État du contrat

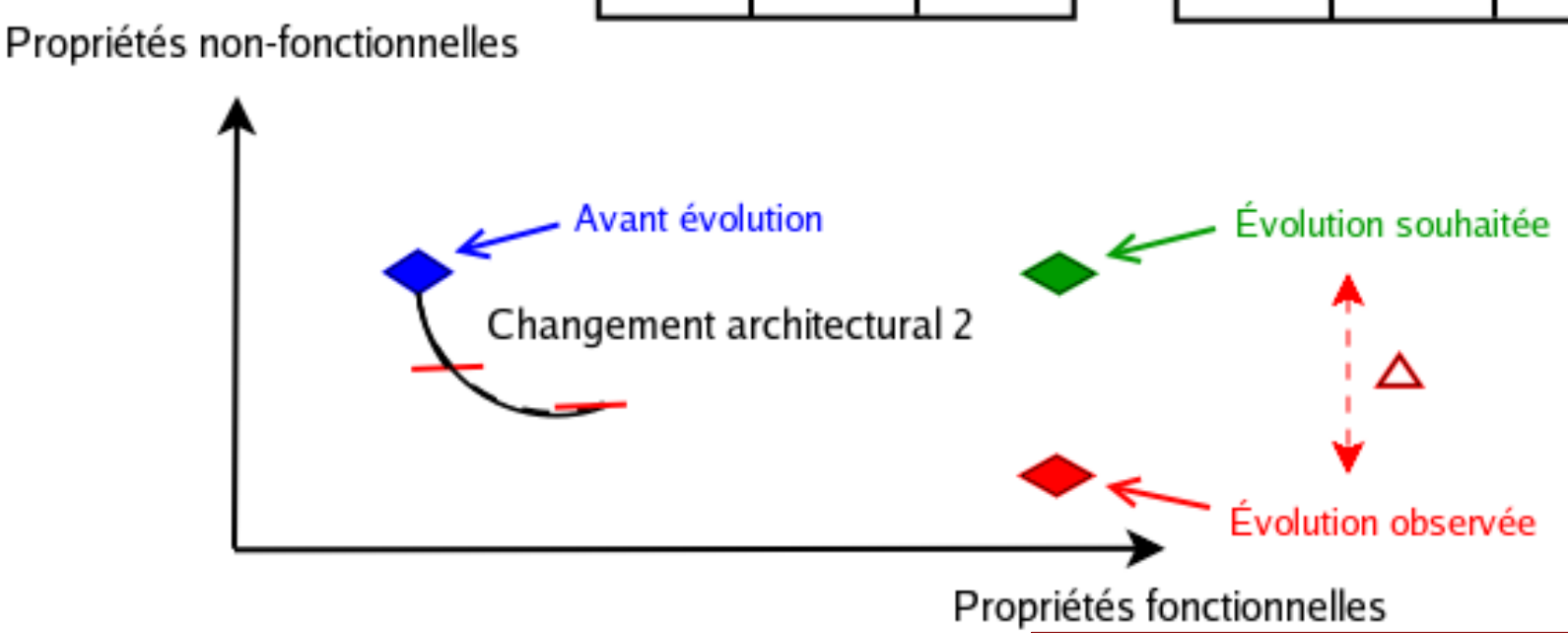
NFP1	AD1 AD4
NFP2	AD2
NFP3	AD4 AD5

Système d'assistance

AD3	NFP1 NFP2	✔

Réactions du développeur

AD3	-	↷



Déroulement de l'algorithme -suite-

État du contrat

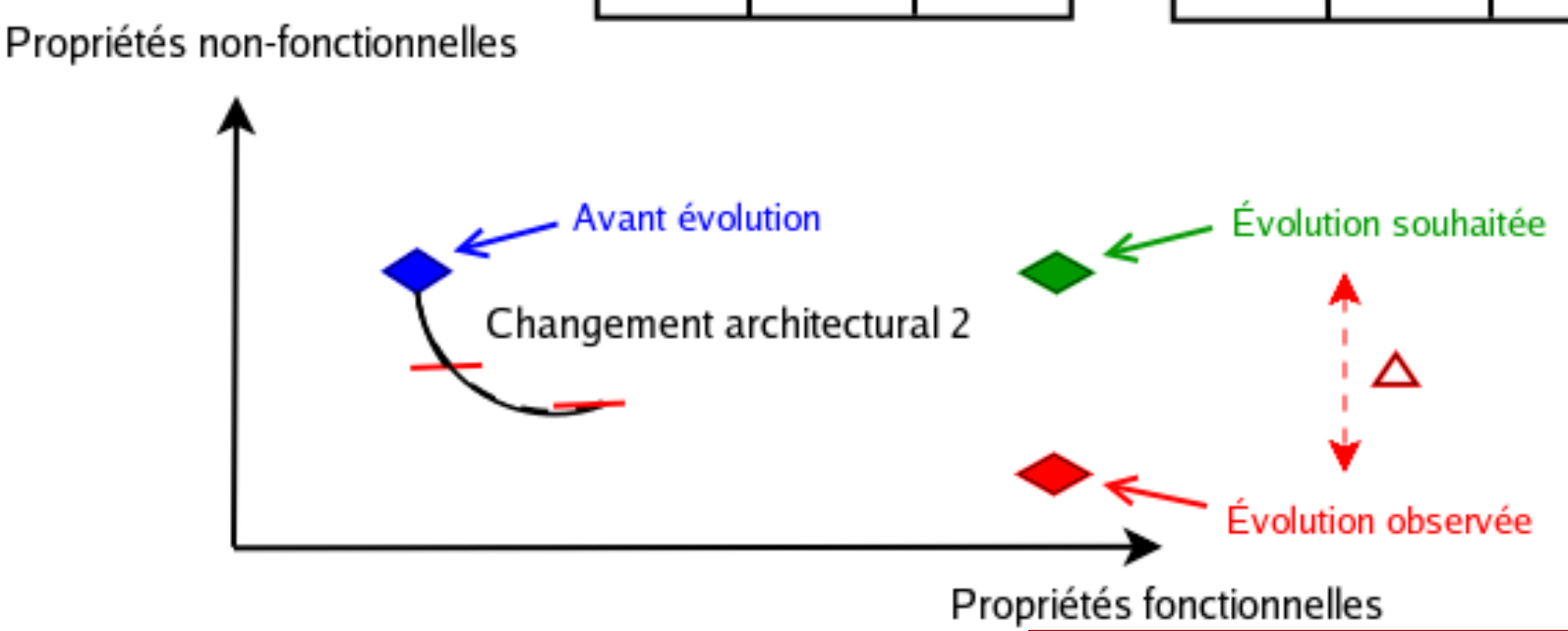
NFP1	AD1	AD4
NFP2		
NFP3	AD4 AD5	

Système d'assistance

AD3	NFP1 NFP2	✔
AD2	NFP2	✘

Réactions du développeur

AD3	-	↷



Déroulement de l'algorithme -suite-

État du contrat

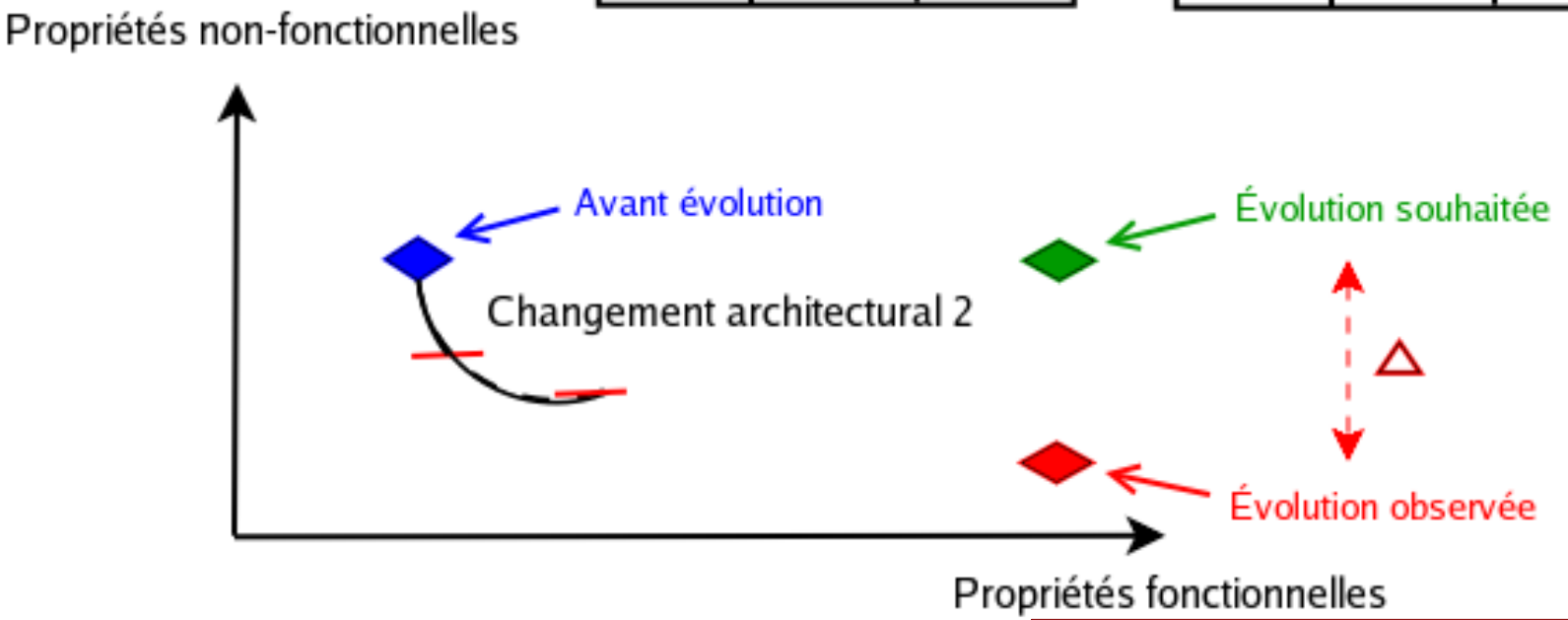
NFP1	AD1	AD4
NFP2		
NFP3	AD4 AD5	

Système d'assistance

AD3	NFP1 NFP2	✔
AD2	NFP2	✘

Réactions du développeur

AD3	-	↷
AD2	-	↷



Déroulement de l'algorithme -suite-

État du contrat

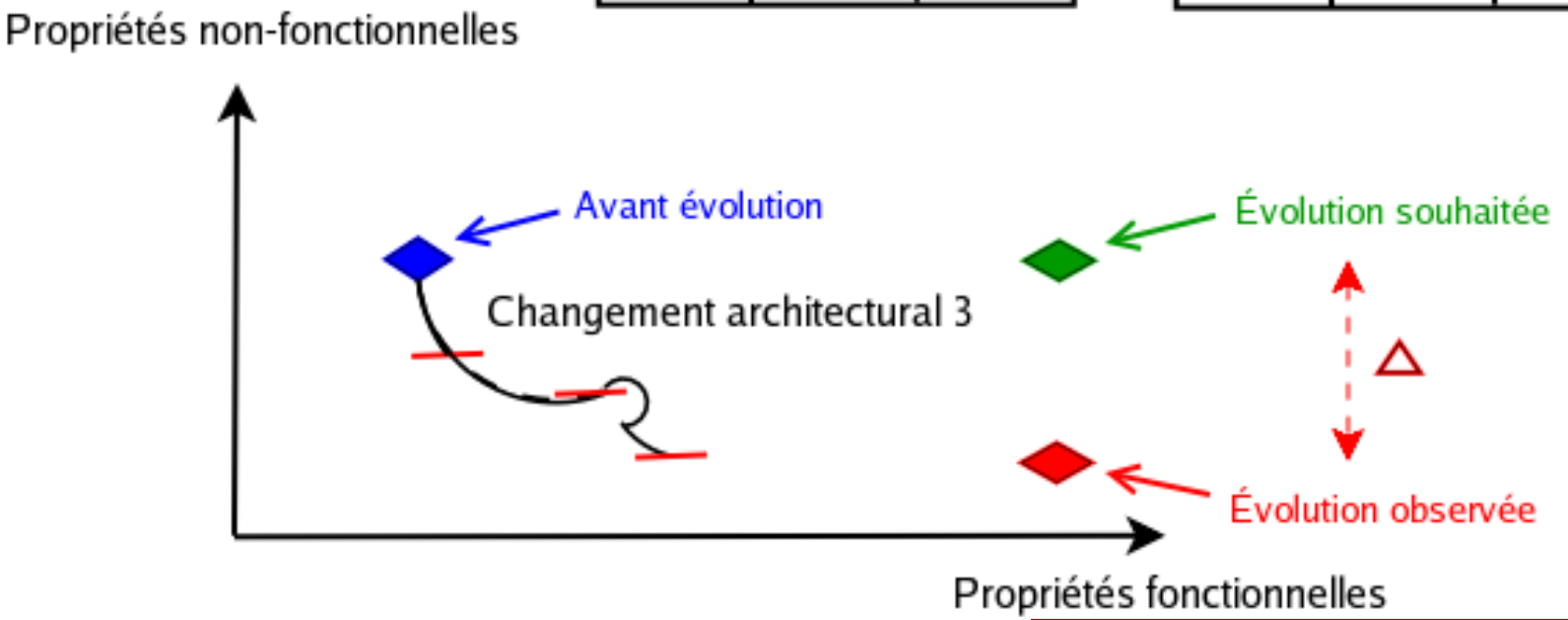
NFP1	AD1	AD4
NFP2		
NFP3	AD4 AD5	

Système d'assistance

AD3	NFP1 NFP2	✔
AD2	NFP2	✘

Réactions du développeur

AD3	-	↷
AD2	-	↷



Déroulement de l'algorithme -suite-

État du contrat

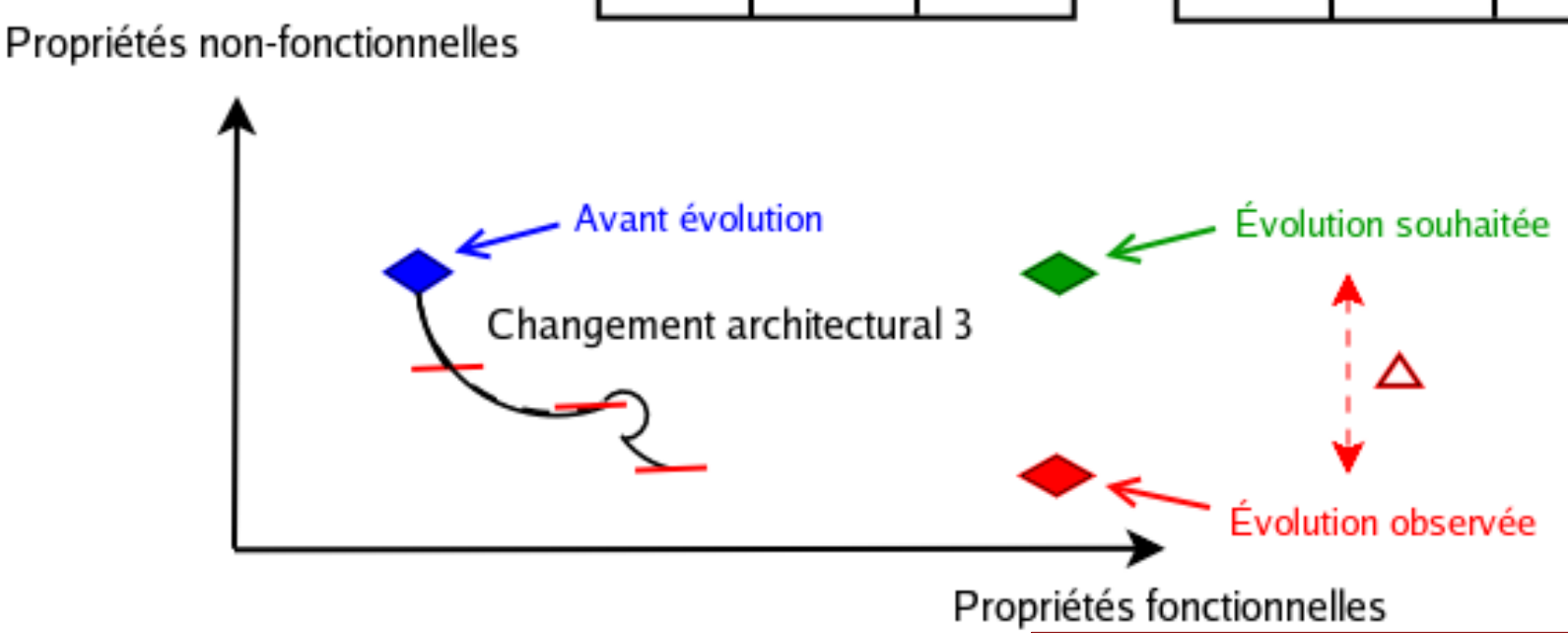
NFP1	AD1
NFP2	
NFP3	AD5

Système d'assistance

AD3	NFP1 NFP2	✔
AD2	NFP2	✘
AD4	NFP1 NFP3	✘

Réactions du développeur

AD3	-	↷
AD2	-	↷



Déroulement de l'algorithme -suite-

État du contrat

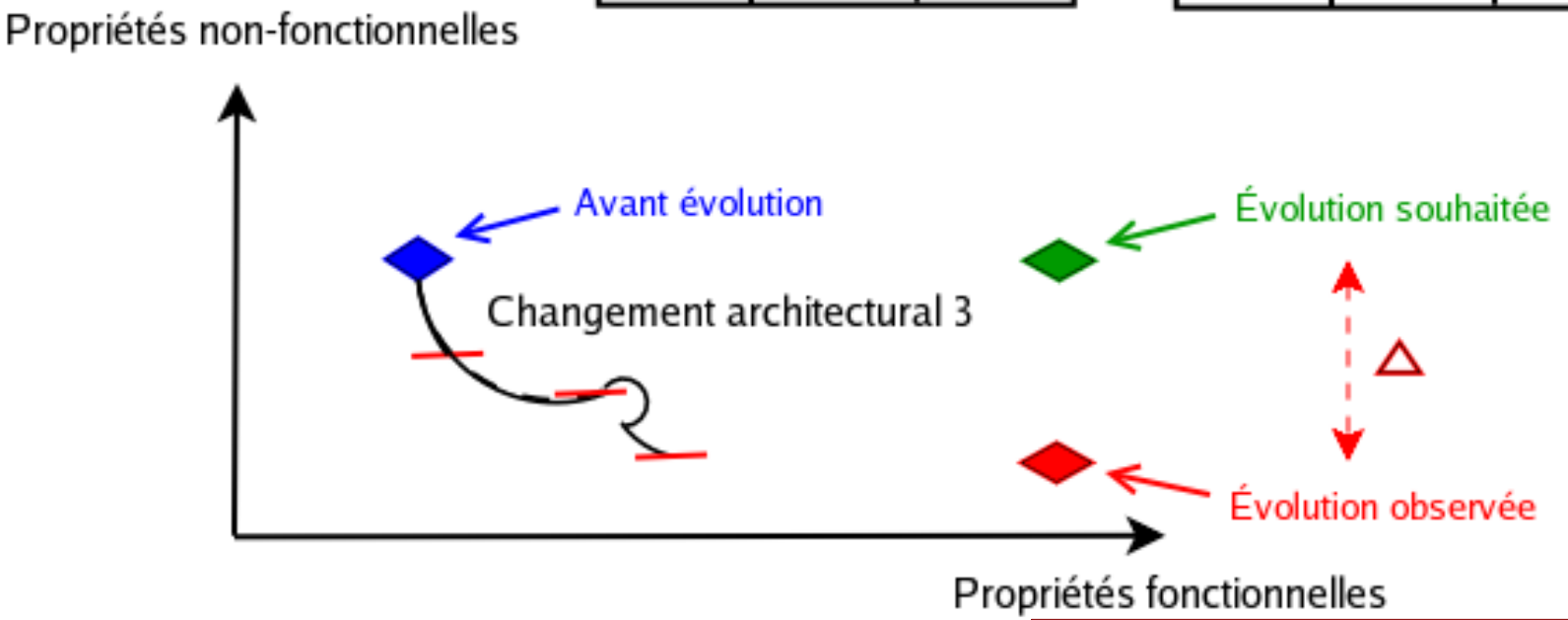
NFP1	AD1	AD4
NFP2		
NFP3	AD4 AD5	

Système d'assistance

AD3	NFP1 NFP2	✔
AD2	NFP2	✘
AD4	NFP1 NFP3	✘

Réactions du développeur

AD3	-	↷
AD2	-	↷
AD4	-	↶



Déroulement de l'algorithme -suite-

État du contrat

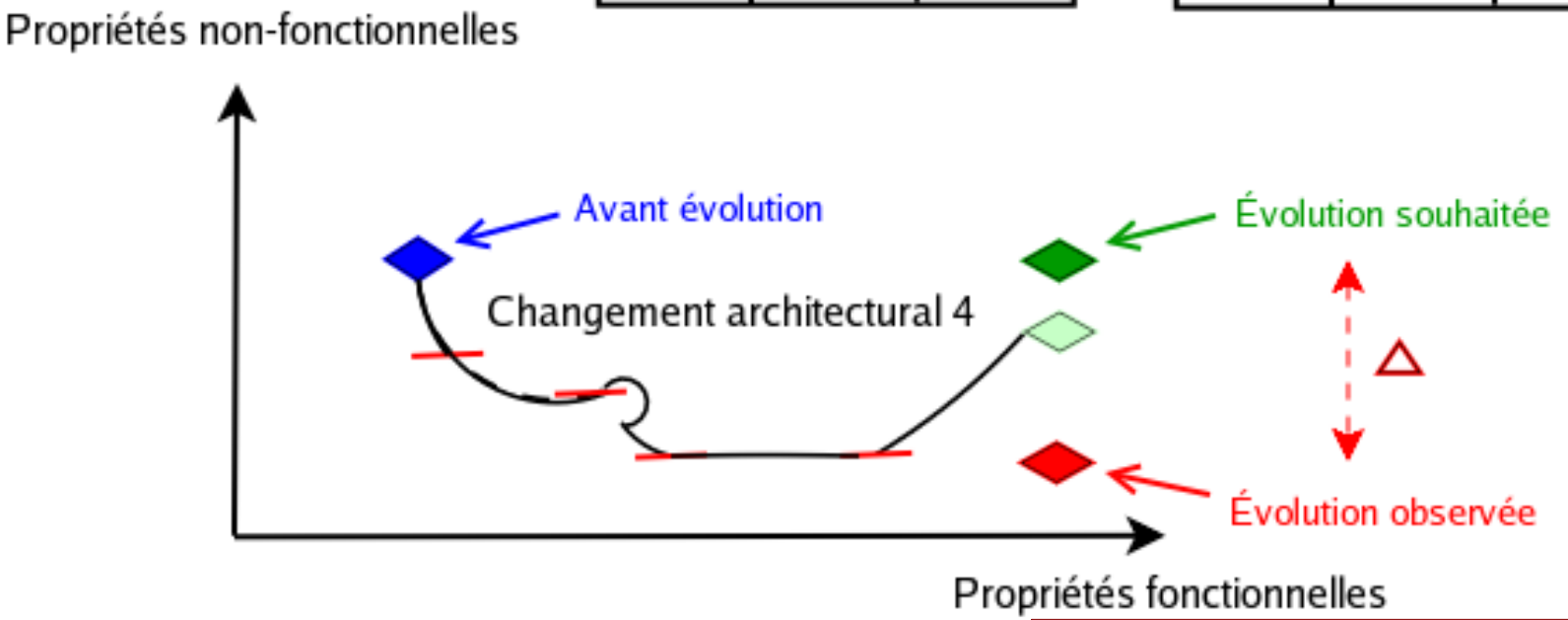
NFP1	AD1	AD4
NFP2		AD6
NFP3	AD4 AD5	AD6

Système d'assistance

AD3	NFP1 NFP2	✔
AD2	NFP2	✘
AD4	NFP1 NFP3	✘

Réactions du développeur

AD3	-	↷
AD2	-	↷
AD4	-	↶



Déroulement de l'algorithme -suite-

État du contrat

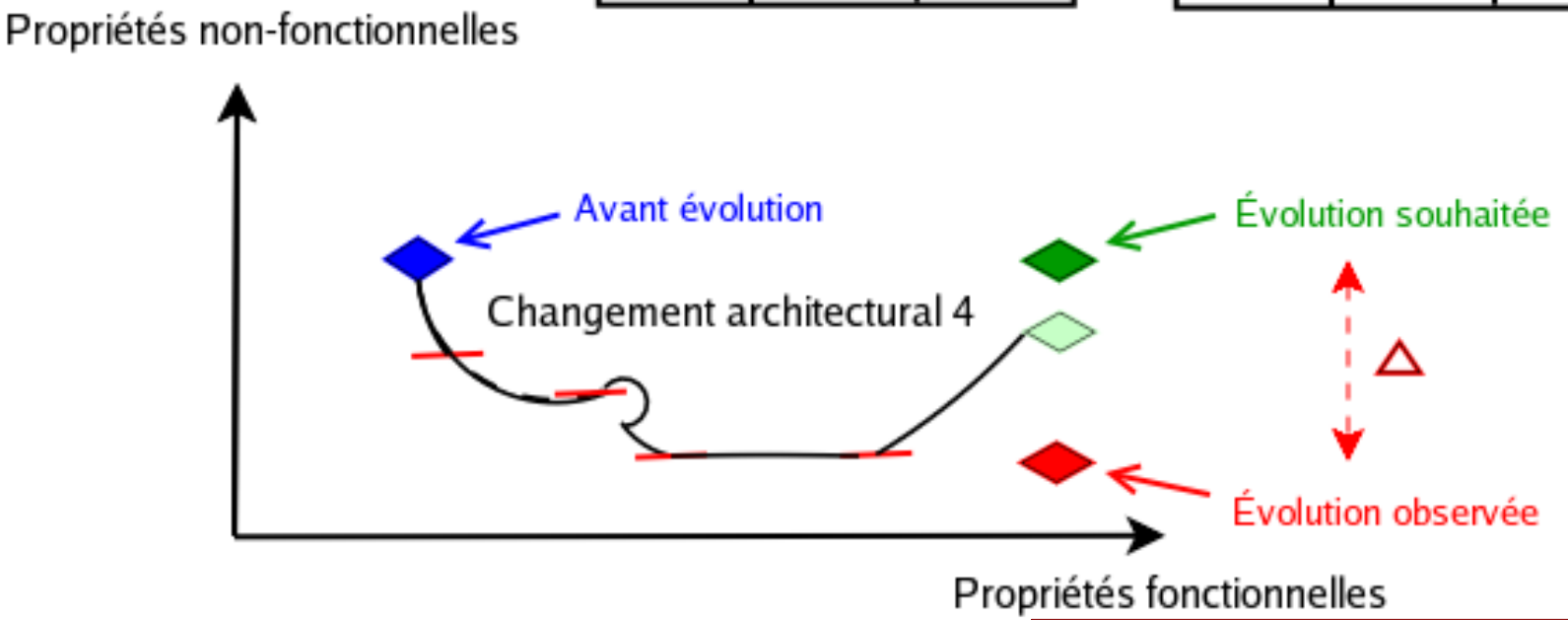
NFP1	AD1	AD4
NFP2		AD6
NFP3	AD4 AD5	AD6

Système d'assistance

AD3	NFP1 NFP2	✔
AD2	NFP2	✘
AD4	NFP1 NFP3	✘
AD6	NFP2 NFP3	✔

Réactions du développeur

AD3	-	↷
AD2	-	↷
AD4	-	↶



Déroulement de l'algorithme -suite-

État du contrat

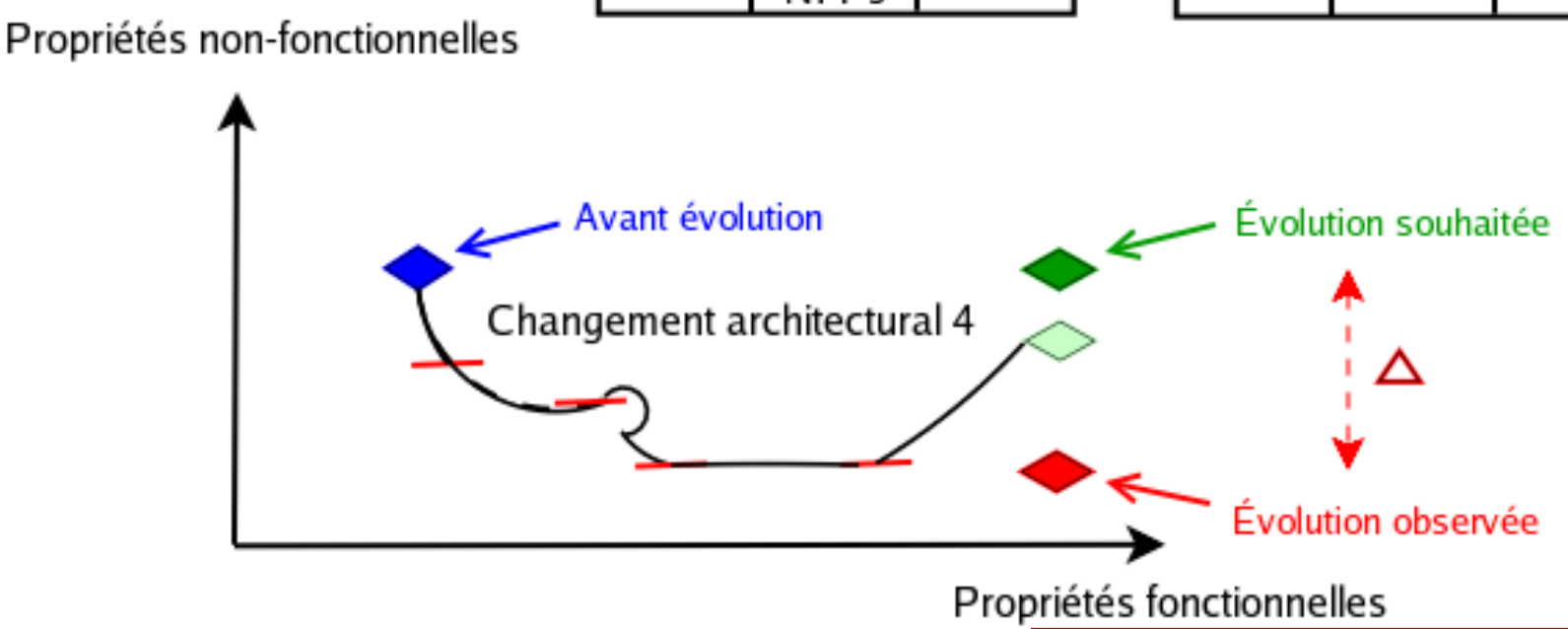
NFP1	AD1	AD4
NFP2		AD6
NFP3	AD4 AD5	AD6

Système d'assistance

AD3	NFP1 NFP2	✔
AD2	NFP2	✘
AD4	NFP1 NFP3	✘
AD6	NFP2 NFP3	✔

Réactions du développeur

AD3	-	↷
AD2	-	↷
AD4	-	↶
AD6	+	↷



Déroulement de l'algorithme -suite-

État du contrat

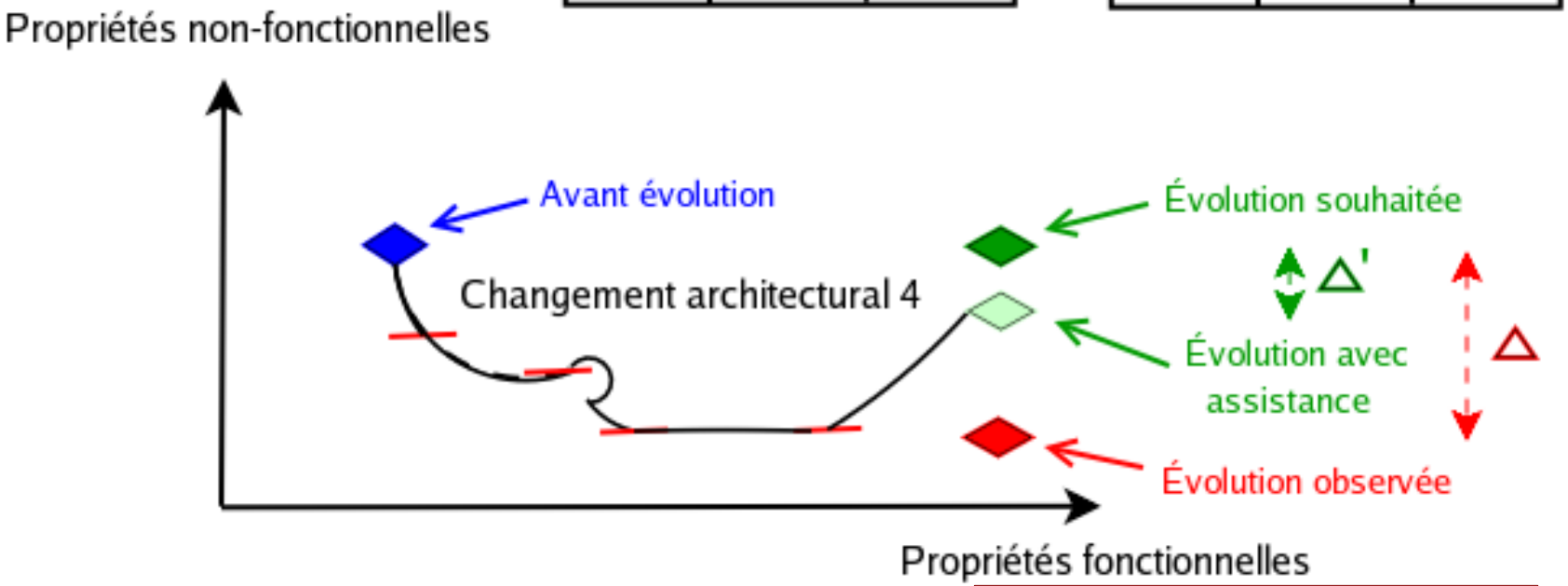
NFP1	AD1	AD4
NFP2		AD6
NFP3	AD4 AD5	AD6

Système d'assistance

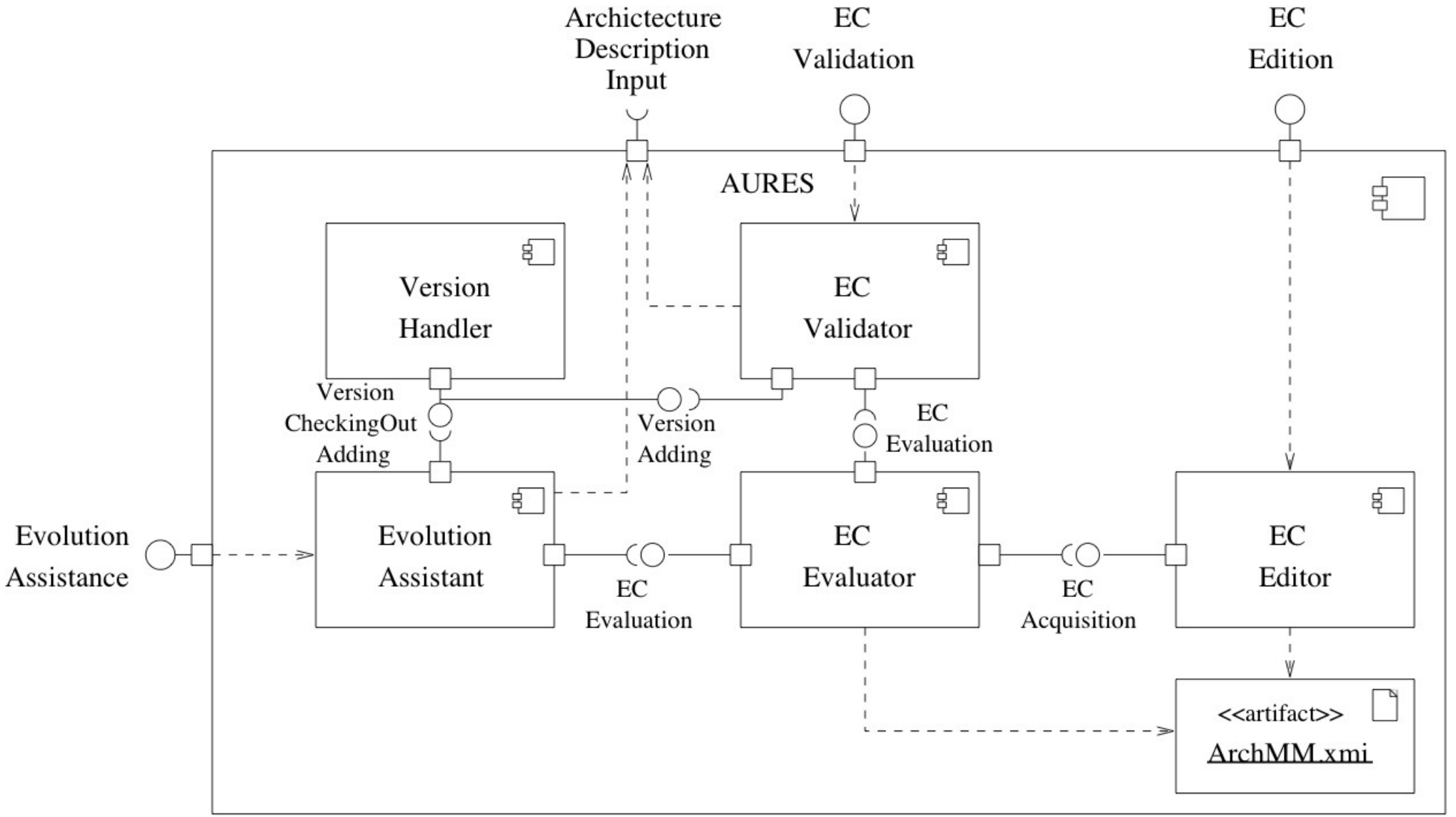
AD3	NFP1 NFP2	✔
AD2	NFP2	✘
AD4	NFP1 NFP3	✘
AD6	NFP2 NFP3	✔

Réactions du développeur

AD3	-	↷
AD2	-	↷
AD4	-	↶
AD6	+	↷



AURES : outil d'assistance à l'évolution



Contenu du cours

- Généralités sur la bibliographie du domaine
- Évolution et maintenance du logiciel (à base de composants)
  - Définitions et mots clés
  - Classification des activités
  - Coûts et facteurs
  - Processus d'évolution
- Assistance à l'évolution des logiciels à base de composants
  - Problématique
  - Contexte de recherche
  - Solution apportée
  - Implémentation de la solution
- Conclusion

Conclusion

- Guider le développeur dans l'évolution de l'architecture de son application
- Ses changements doivent avoir un minimum d'impact sur la qualité initialement souhaitée
- Si changement imposé (obligatoire), le développeur est au courant de son impact
- Contraintes architecturales = outil pour vérifier la qualité d'une architecture lors d'une évolution

Questions