R3.03 – Analyse

2_ DÉVELOPPEMENT LOGICIEL

Dalila TAMZALIT

IUT de Nantes – Département Informatique

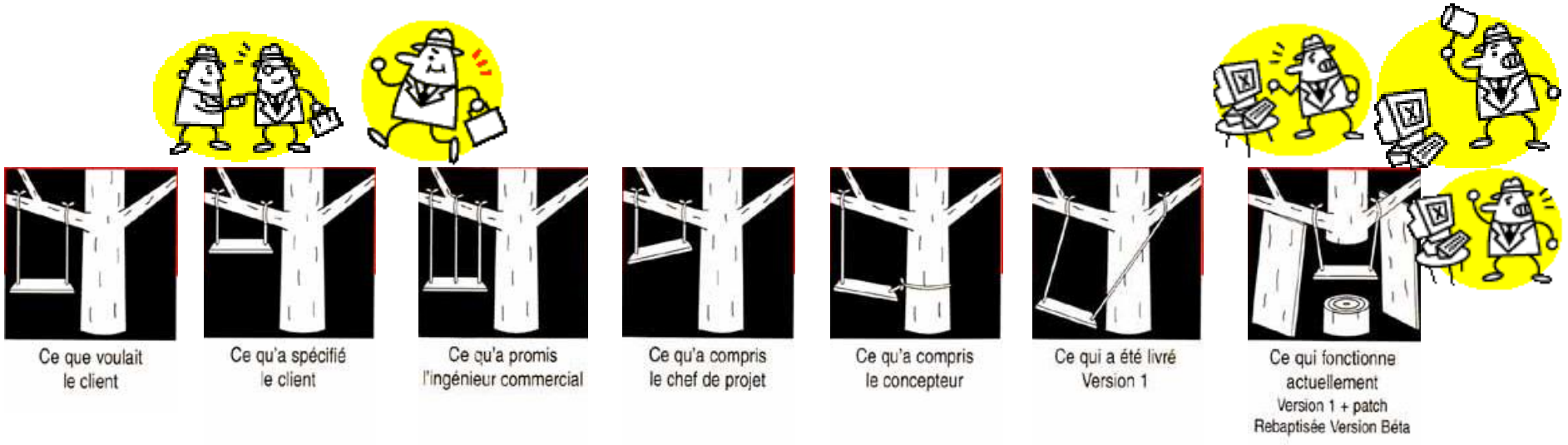
« Développer, Proposer, Satisfaire les besoins des utilisateurs »

Vous êtes l'unique membre de l'équipe de développement et votre chef vous soumet un projet :

Besoin : un logiciel qui calcule le factoriel d'un nombre

Besoin : logiciel de gestion des ressources humaines d'une multinationale

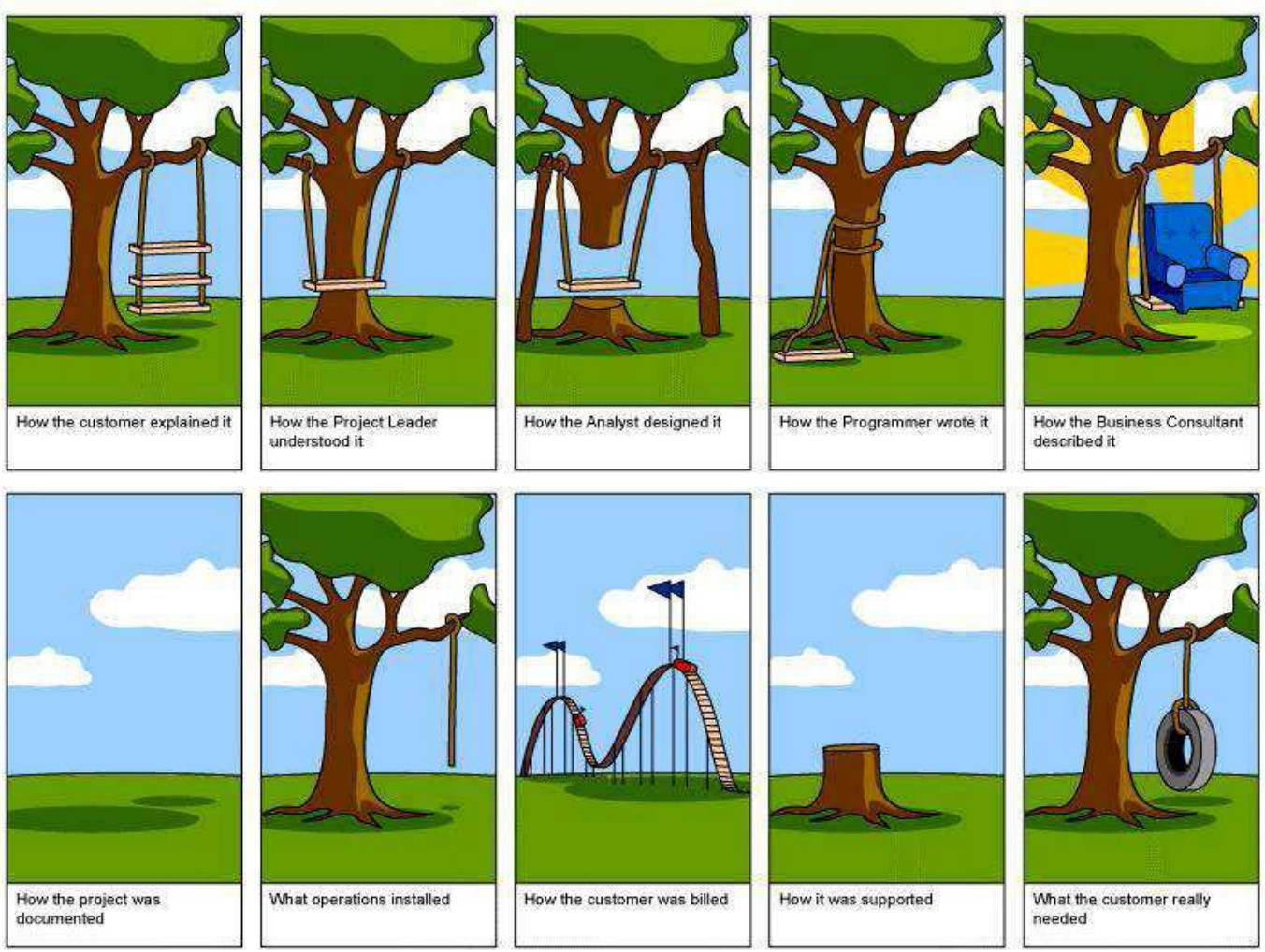
« Développer, Proposer, Satisfaire les besoins des utilisateurs »



Inspiré de *http://www.bric-a-brac.org/humour/images/informatique*

« Développer, Proposer, Satisfaire les besoins des utilisateurs »

Déjà vu en R2.10 en BUT1

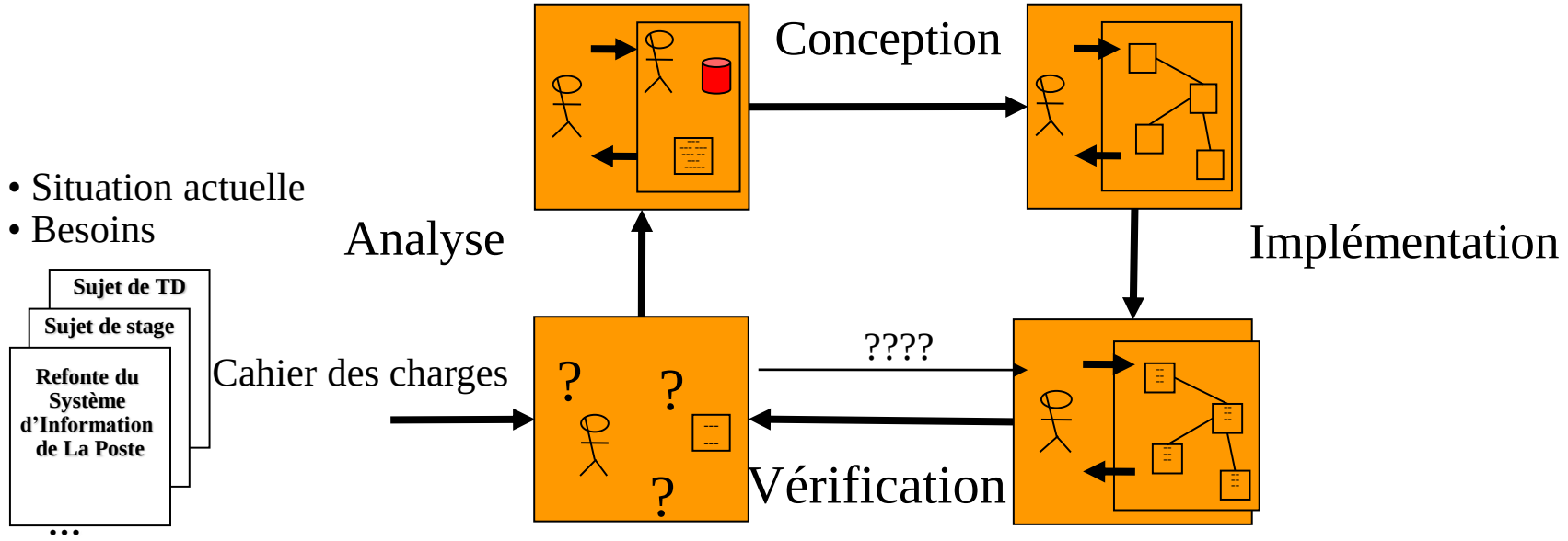


www.agiledeveloper.com

Génie logiciel

- L'effort doit être adapté à la complexité du système à développer.
- Nécessite **:**
  - **Décrire** : besoins, spécification de conception, documentation
  - **Implémenter** : conception, programmation
  - **Évaluer** : test, vérification, validation, révision
  - **Gérer** : planification, échelonnage, communication
  - **Faire fonctionner** : déploiement, installation, maintenance
- La complexité du système doit être contrôlée :
  - Définitions, méthodes, approches, standards…
  - Outils

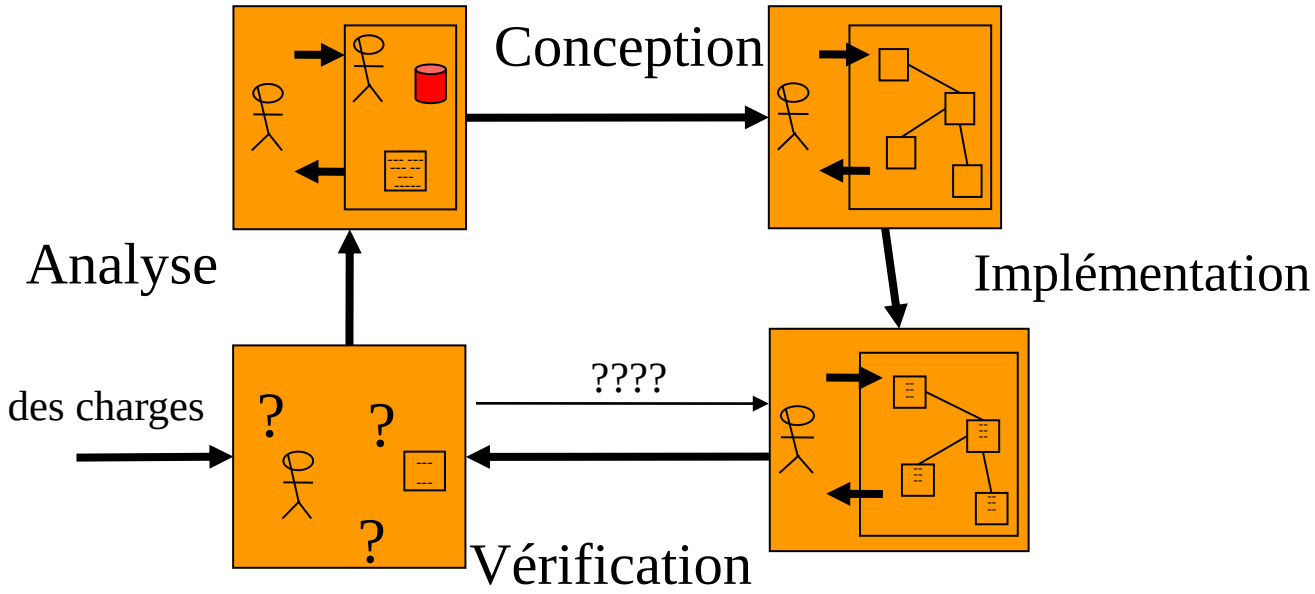
Principale étapes

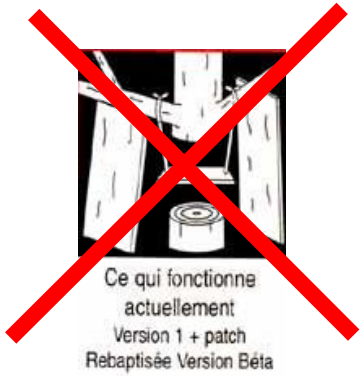


- **Analyse** : abstraction et extraction des besoins
- **Conception** : formalisation + solutions organisationnelles et techniques
- **Implémentation** : adaptation, codage, installation (logiciels + matériel)

Principale étapes

- Situation actuelle
- Besoins

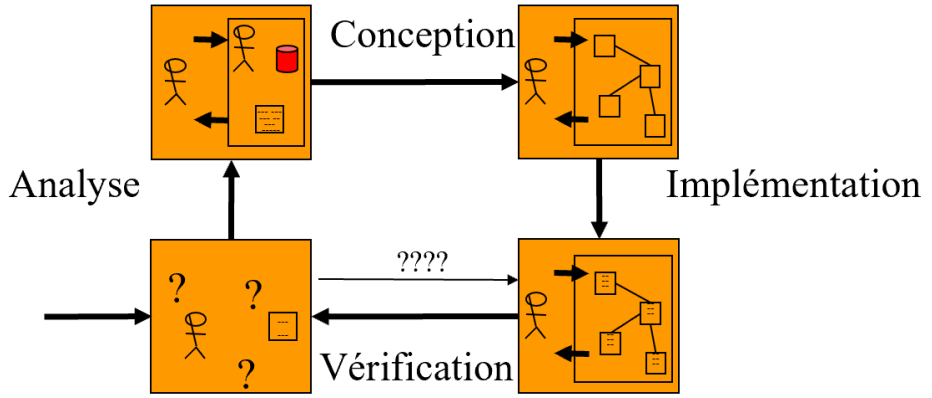




Activités de développement

Processus de développement

- **Planification** du projet
- Recueil des **exigences**
- **Analyse** et spécification
- **Conception**
- **Implémentation**
- Vérification / **Test**
- Intégration
- Livraison / **Déploiement**
- **Maintenance**



En continu :

- Documentation
- Vérification et validation
- Gestion

L'enchaînement de ces activités se passe plus ou moins bien

Développement initial

- **Décisions fondamentales**
  - *Logique métier* : bien la cerner, la formaliser.
  - *Technologie* : langage de programmation, conventions, librairies
  - *Architecture* : composants, interactions
  - *Connaissances du domaine du programme* : nécessaire à l'évolution

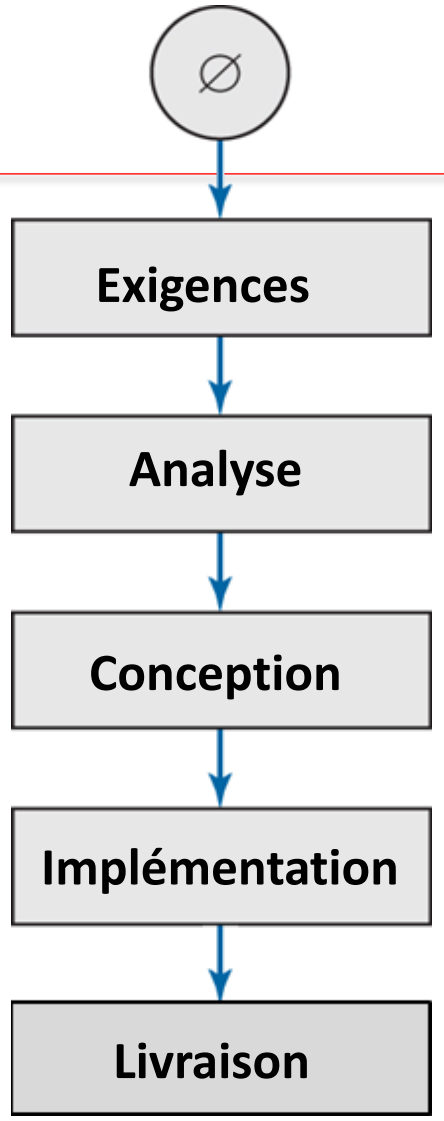
IUT Nantes
Pôle Sciences et technologie
Nantes Université

Développement logiciel idéal

- Linéaire
- Ne commence de rien
- Développement se termine à la livraison
- Suppose que chaque phase est correcte du premier coup

- **Idéaliste!**

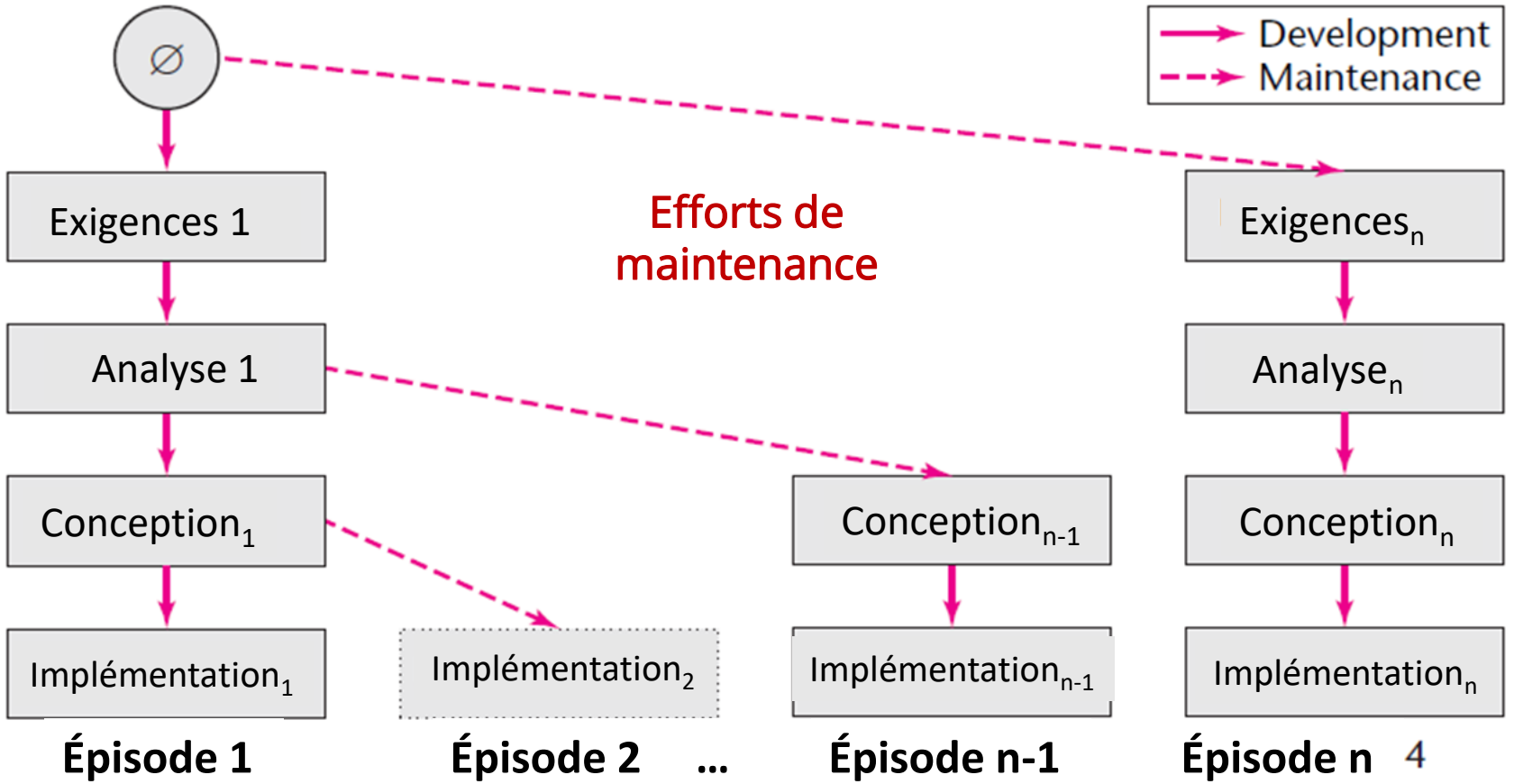
… voire utopique !



En pratique…

- Les développeurs font des erreurs
- Le client change les exigences pendant le développement du logiciel

En pratique…


Development
Maintenance
∅
Exigences 1
Analyse 1
Conception$_1$
Implémentation$_1$
Efforts de maintenance
Implémentation$_2$
Conception$_{n-1}$
Implémentation$_{n-1}$
Exigences$_n$
Analyse$_n$
Conception$_n$
Implémentation$_n$
Épisode 1
Épisode 2
…
Épisode n-1
Épisode n
4

Modèles de processus de développement

Cycle de vie du logiciel / Processus de développement

- Un processus de développement définit un ensemble d'activités de développement et leur enchaînement
- Une activité comprend :
  - des tâches,
  - des contraintes,
  - des ressources,
  - une façon d'être réalisée
- La plupart des modèles des processus reprennent les activités fondamentales mais les organisent différemment
- **Le savoir-faire humain est indispensable !**

Processus de développement de logiciels

- Description **abstraite** et **idéalisée** de l'organisation des activités du développement d'un logiciel
- Décrit un ensemble **ordonné** d'activités
- Doit être « personnalisé » pour l'entreprise de façon à définir l'ordonnancement idéal des activités
  - Que doit-on produire ?
  - Qui fait quoi ?
  - Comment superviser l'évolution du projet ?
  - Comment gérer les changements ?
- Il existe plusieurs processus de développement.

IUT Nantes
Pôle Sciences et technologie
Nantes Université

Processus de développement

- La manière selon laquelle une organisation produit un logiciel
  - Méthodologie
  - Modèle de développement du logiciel
  - Techniques et outils à utiliser
  - Rôles et individus nécessaires
- Différents modèles de développement.

Quelque soit l'approche adoptée…

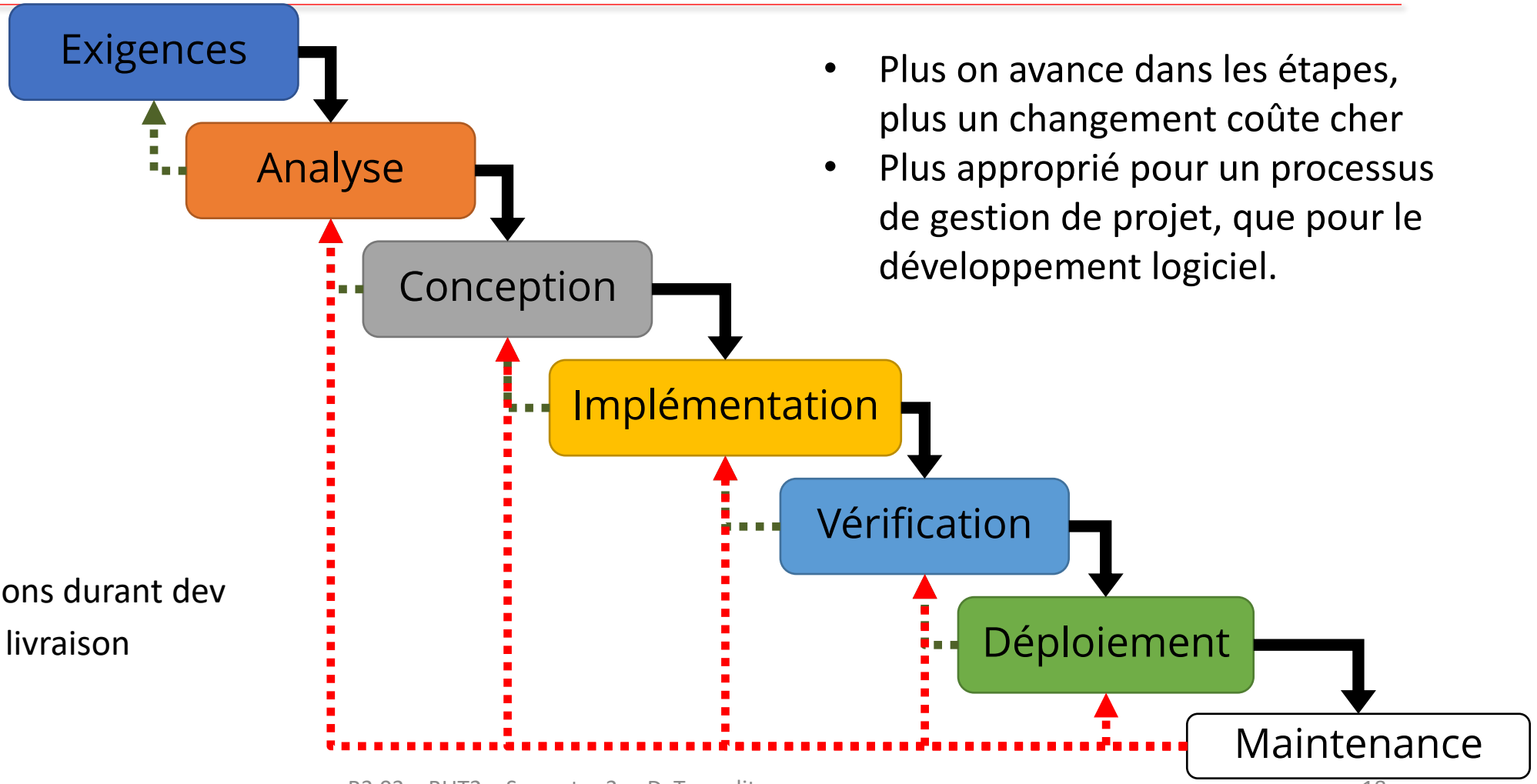
- Recueil des exigences
- Analyse et spécification
- Conception
- Implémentation
- Vérification / Test
- Livraison / Déploiement

Et ensuite :

- Maintenance
- Evolution

Modèle en cascade

[Royce 1970]

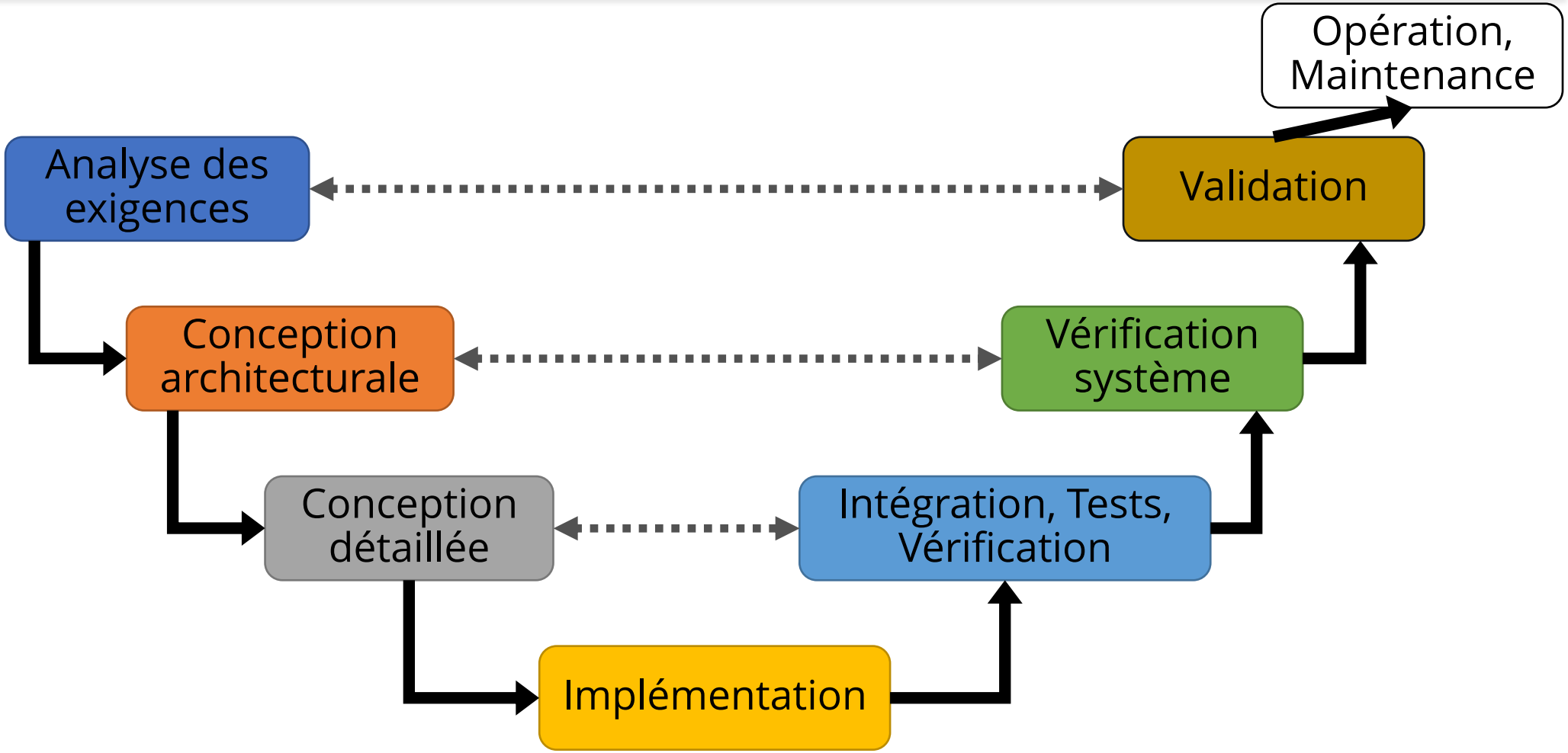


- Plus on avance dans les étapes, plus un changement coûte cher
- Plus approprié pour un processus de gestion de projet, que pour le développement logiciel.

Vert: maintenance et révisions durant dev

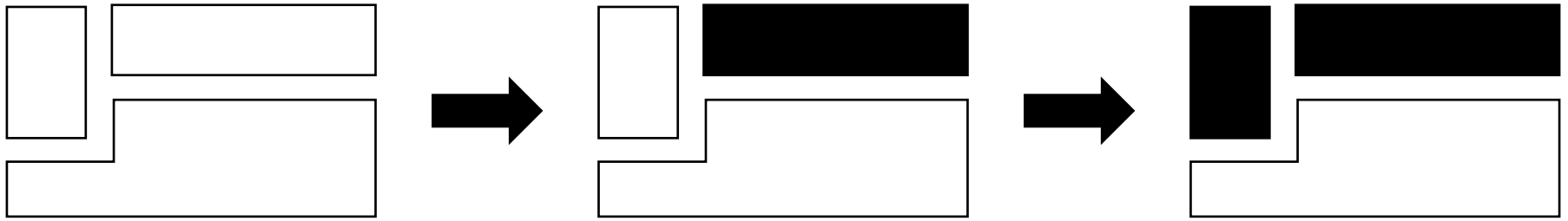
Rouge: maintenance après livraison

Modèle en V


Opération, Maintenance
Analyse des exigences
Validation
Conception architecturale
Vérification système
Conception détaillée
Intégration, Tests, Vérification
Implémentation

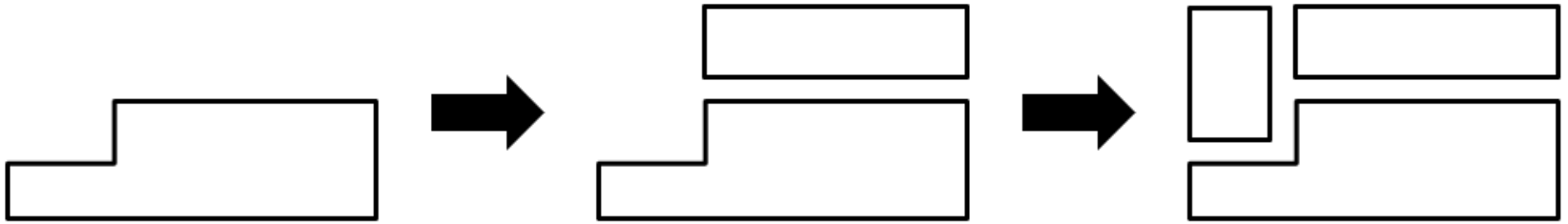
Développement itératif

- Repose sur la spécification et la mise en œuvre de parties individuelles du logiciel.
- Architecte la coquille du produit complet, puis améliore chaque composant dans l'itération suivante et ainsi de suite.

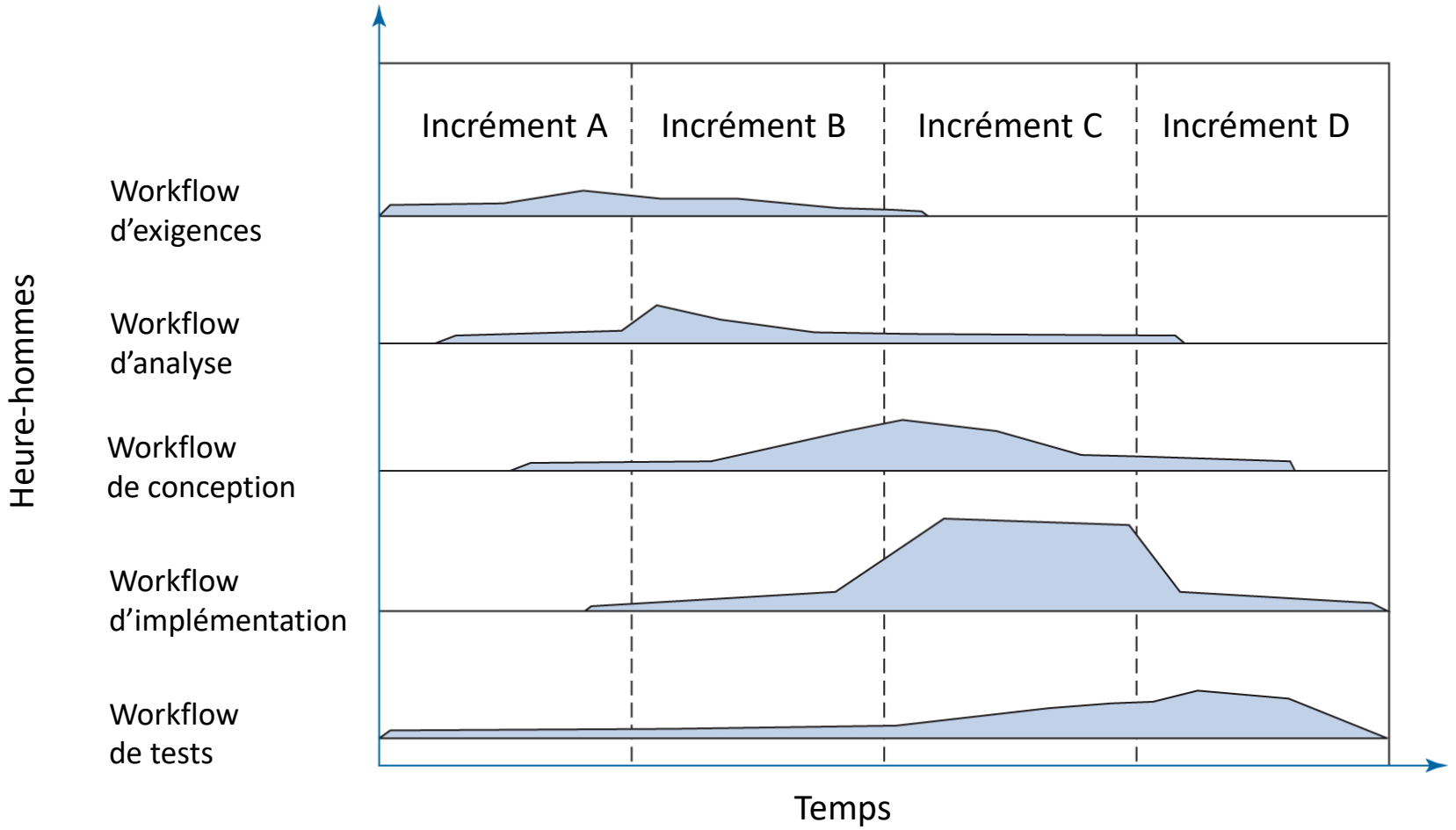


Développement incrémental

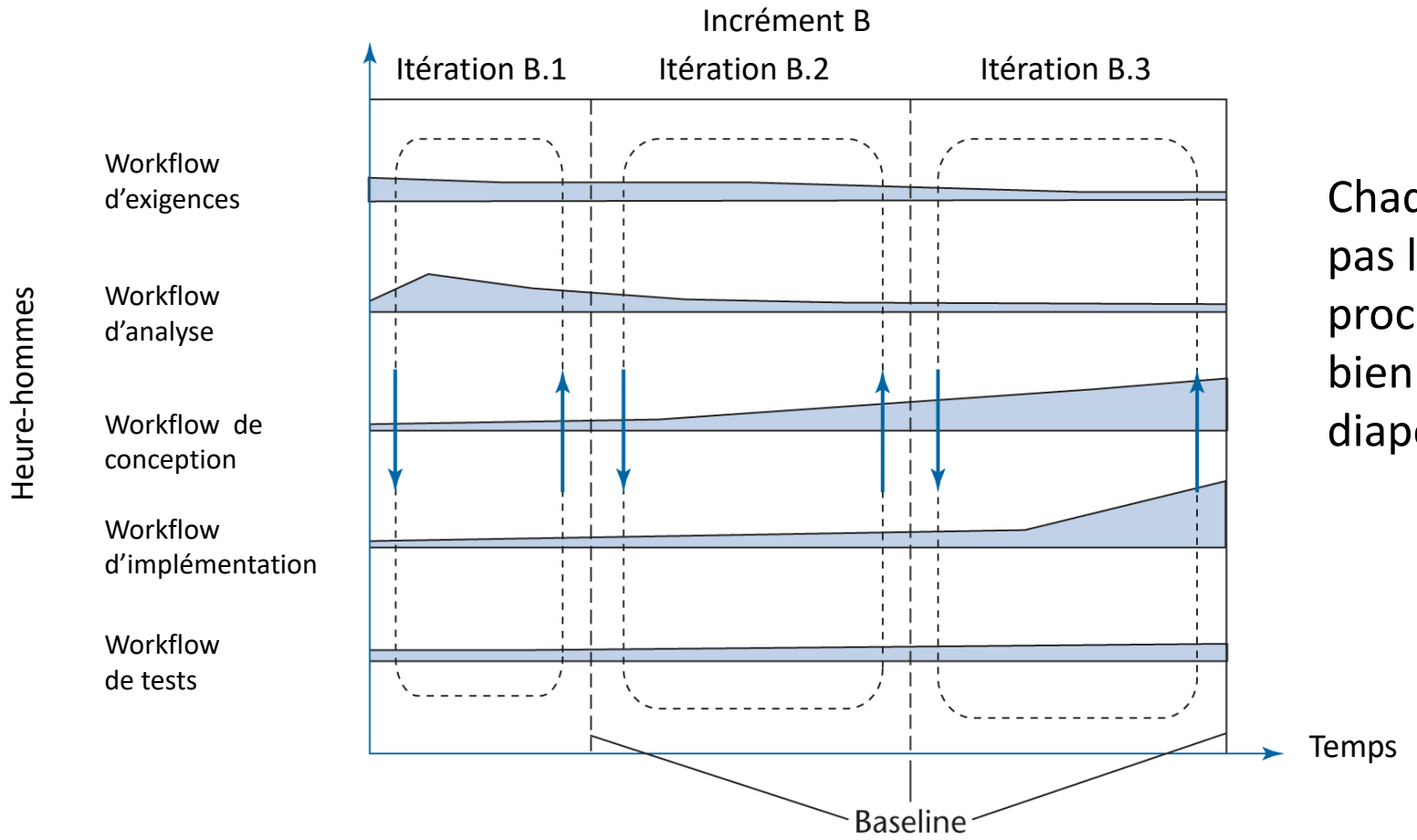
- Pour gérer de plus grandes information, utiliser le **raffinement par étapes *(stepwise refinement)***
  - Se concentrer sur les aspects les plus importants à ce moment
  - Laisser les moins critiques pour plus tard
  - Chaque aspect sera géré, dans l'ordre d'importance actuelle
- Chaque incrément abouti à une livraison
  - Voir une fonctionnalité complétée tôt dans le processus
  - Chaque incrément est le prototype du suivant
- Plus facile d'évaluer le progrès



Processus incrémentaux



Itération et incrémentation (I&I)



Chaque incrément n'est pas le résultat d'un processus linéaire, mais bien itératif (détails diapo 25)

Chaque incrément est le résultat de plusieurs itérations

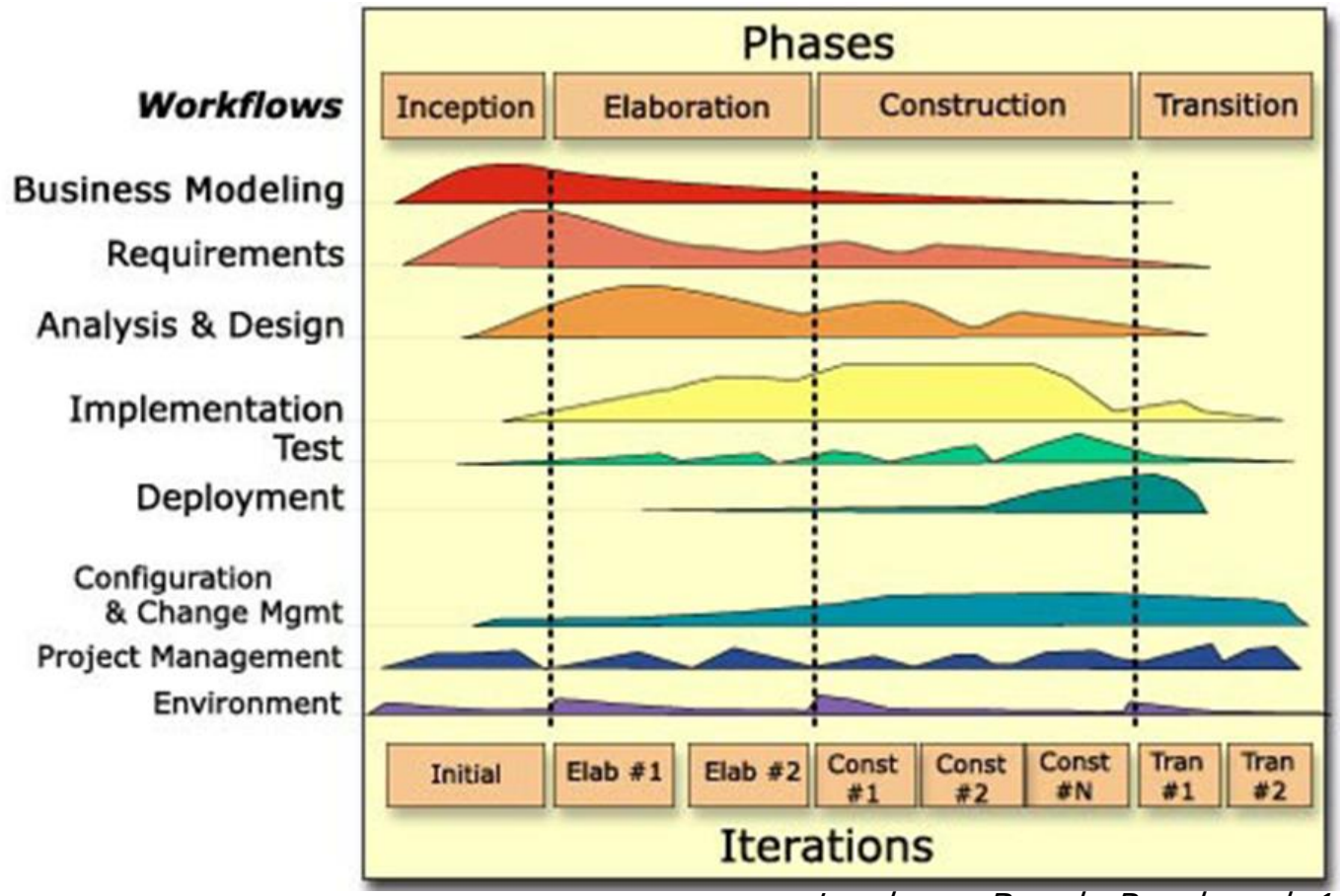
Avantages des processus I&I

- Tous les flux d'activités *(workflow)* sont impliqués dans chaque incrément, mais certains vont **dominer** plus.
- Plusieurs opportunités de **tester**, recevoir du retour du client et s'ajuster :
  - La vérification du produit est faite à plusieurs reprises.
  - Test workflow à chaque itération. Fautes détectées et corrigées très tôt.
- **Robustesse** de **l'architecture** peut être déterminée **tôt** dans le développement :
  - Architecture : ensemble des différents modules qui composent le système et leur intégration.
  - Robustesse : propriété de permettre des extensions et changements sans se détériorer
- Plusieurs modèles : UP, méthodes agiles…

Avantages des processus I&I

- **Livrables spécifiques** pour chaque incrément et chaque workflow
- Le projet complet peut-être vu comme un ensemble de mini projets (incréments)
- On peut atténuer et résoudre les **risques** plus tôt
  - Il y a toujours des risques impliqués dans le développement et la maintenance d'un logiciel

Modèle du Processus Unifié (*Unified Process*)



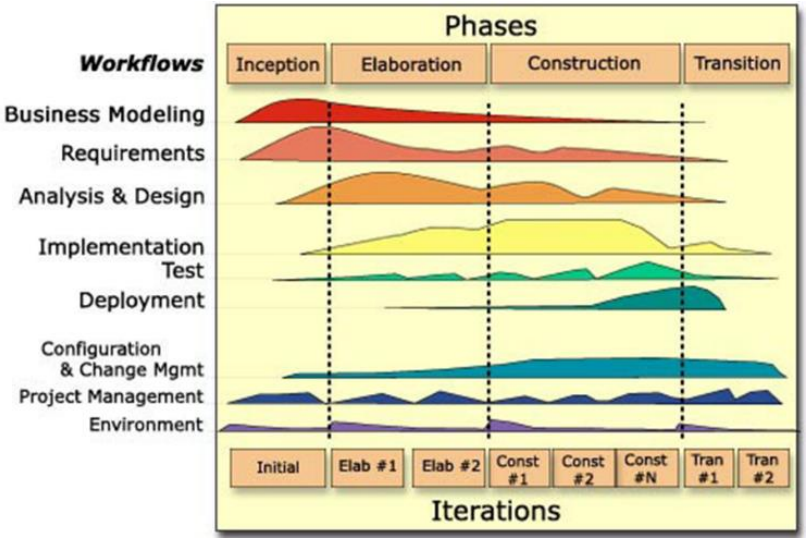
Jacobson, Booch, Rumbaugh 1999

- À partir duquel les processus I&I ont été élaborés
- RUP ’99 par les "Three Amigos" (Jacobson, Booch, Rumbaugh) IBM, pères de l’UML
- Il n’y a pas un modèle qui fonctionne sur tous les projets.
- Modèle adaptable, à modifier pour chaque logiciel à développer

Modèle du Processus Unifié

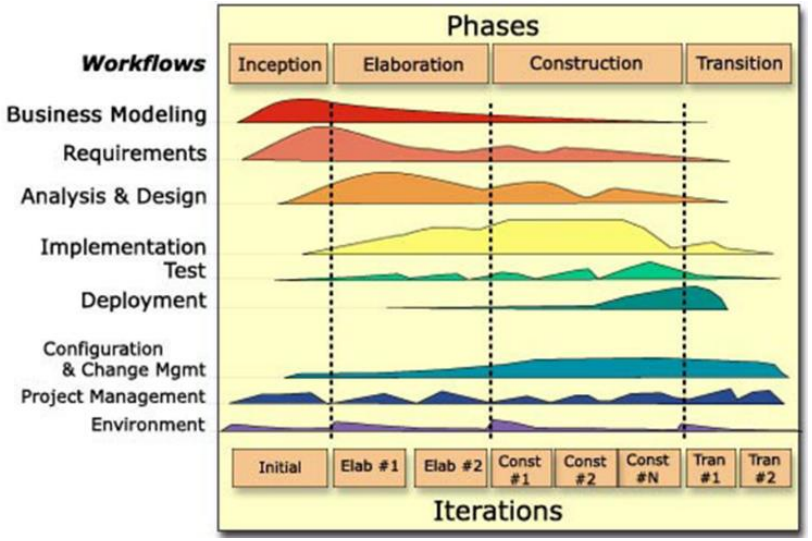
- **Phases** = Incréments, livraison
  - **Itérations** : chaque version du logiciel aux différentes étapes du développement.
- **Création (inception)** : cadre, cas d'utilisation
- **Élaboration** : conception de l'architecture initiale, estimés de coût & ressources
- **Construction** : développe les composants, livraisons, critère d'acceptation
- **Transition** : déploiement

- **Workflows** : activités tout au long de la vie du logiciel



Modèle du Processus Unifié

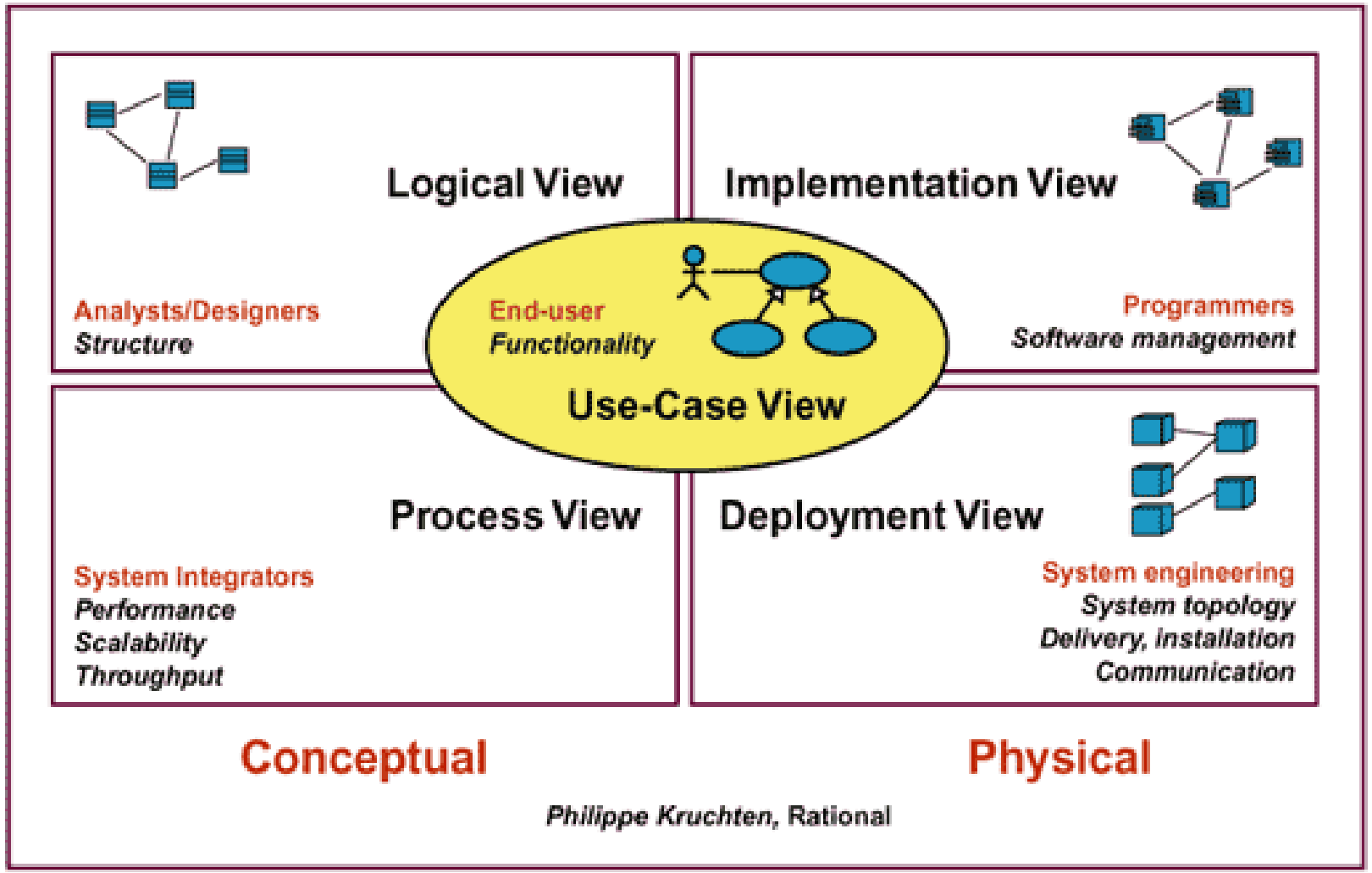
- **Exigences**: analyse du domaine de l'appli & création des artéfacts des exigences
- **Analyse & conception:** création de la solution & des artéfacts de conception
- **Implémentation**: création du code
- **Test**: évaluation du processus et des artéfacts
- **Déploiement**: transition du système à l'utilisateur.
- **Environnement**: maintenance (communication & gestion des configurations)



Résumé

- **Linéaire** (Cascade, V)
  - Petits projets, cadre bien défini, peu de risques, pas de changements de besoins en phase de tests, tout est déployé d'un coup
- **Itératif** (Spirale)
  - Exigences pas encore clairs, risque élevé de changements dans les exigences, prototype initial nécessaire, chaque itération améliore et converge vers le produit final suite au retour de la précédente
- **Incrémental** (Phases)
  - Livraison des composants par ordre de priorité pour réduire risques et augmenter la qualité tout en incorporant les changements tout au long du projet, risques de conception et techniques éliminés tôt, pour de gros projet car les tests commencent tôt
- **Itératif et incrémental** (Unifié, agile)
  - Chaque composant est développé dès le 1er incrément, ils sont améliorés dans les incréments suivants, peut rapidement intégrer les changements et changer de priorités

UP et le modèle « 4+1 » vues



Le modèle d 'architecture de *Kruchten*, dit *modèle des 4 + 1 vues*, est adopté dans l'*Unified Process*. La vue des cas d'utilisation établit le lien et dirige la création des autres diagrammes d'architecture.

Fortement couplé à UML.

* Philippe B. Kruchten, *The 4+1 View Model of Architecture*, IEEE Software, novembre 1995.

Le modèle « 4 + 1 vues » Ph. Kruchten

Vue logique : principal souvent unique ption architectu

Sta iagrammes de classes et *objets.*

Dynamique : *Diagrammes d'interactions (séquences, communication)* et *d'états.*

Abstraction Encapsulation

Vue déve ent (réalisation): v ation de l'organisation des c ants. *Diagrammes de co nts*.

Organisation statique

Vue des cas d'utilisation : analyse du système p nces des utilisa

Diagramme de cas d'utilisation

Cohérence globale

Vue des processus : interactions entre diff rents processus et allocati biets (fil d'exécution, tâches, paralle).

Diagrammes d'activ de séquences, diagrammes de communication ou *d'états-transitions*.

Flux d'exécution

Vue de déploiement ysique) : déploiement, nœ modules et différentes contra (géographiques, bandes p antes, QoS…)

Diagra de composants et *diagra mes de déploiement*.

Ressources matérielles

Flux de travail *(Workflows)*

Workflow des exigences

- But: déterminer ce dont le client a **besoin**

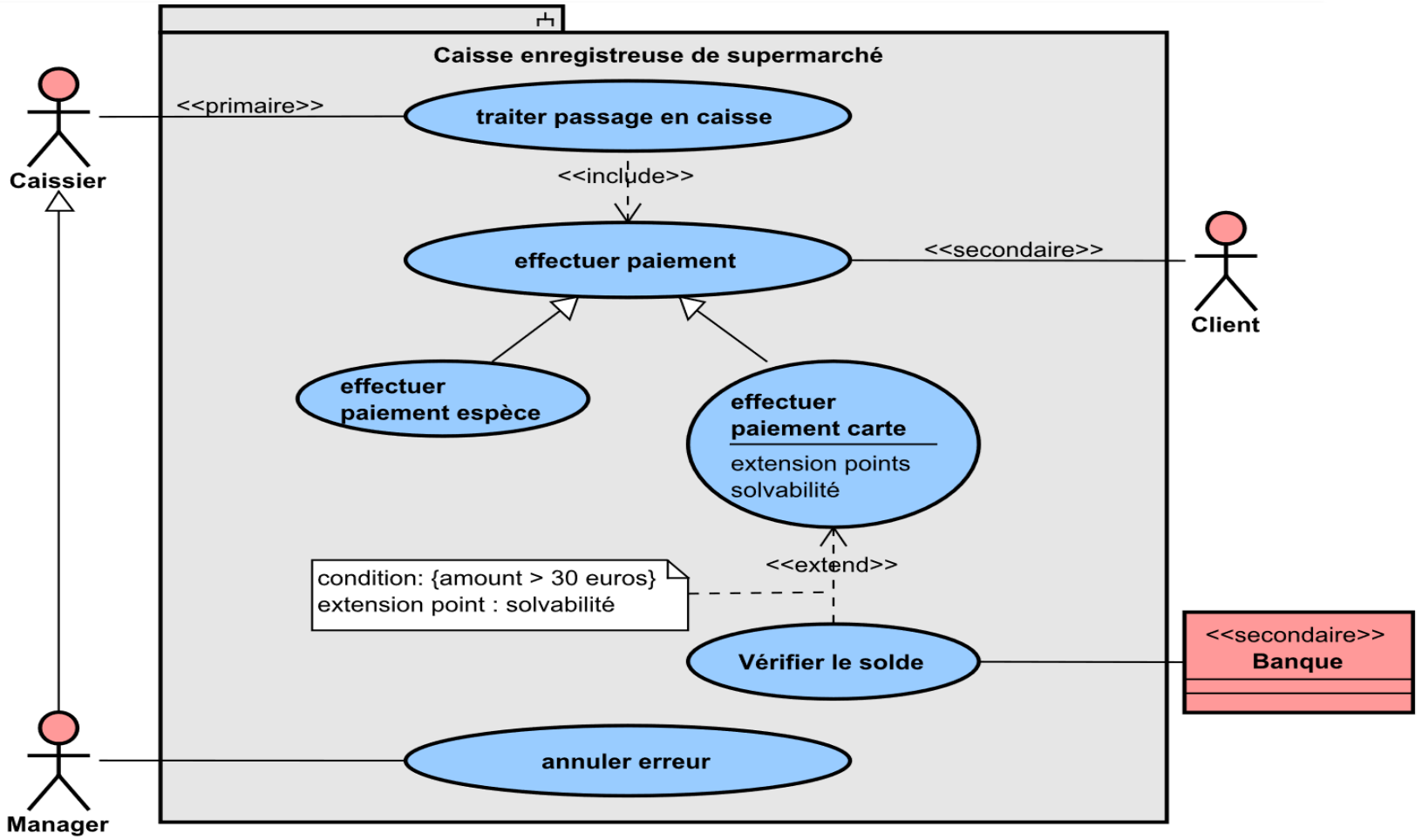
1. Comprendre le **domaine** d'application
   - Environnement spécifique au métier où le logiciel sera exploité.
   - Quel est le modèle d'affaires (vente de produits, fournir services…)
2. Construire un **modèle d'affaire**
   - Utiliser UML pour décrire le processus d'affaire du client (cas d'utilisation).
   - À n'importe quel moment, si le client pense que le coût n'en vaut pas la peine, on met immédiatement fin au développement
3. Définir les **exigences** du système
   - Découvrir quelles sont les **contraintes**

Workflow d'analyse

- But: analyser et **affiner** les exigences
- *Pourquoi ne pas le faire dans le workflow des exigences ?*
  - Exigences doivent être entièrement compréhensibles par le client
  - Exprimés en langage naturelle, donc imprécises/ambigües
  - Artéfacts d'analyse doivent être assez précis et complets pour l'équipe de développement
- Document (artefacts) de **spécifications**
  - Constitue un contrat, décrit ce que le logiciel doit faire
  - Précis et non ambigüe
- Spécifications doivent être complètes et correctes
  - Base pour les tests et la maintenance
  - Une fois approuvé, planification détaillée, cahier des charges, estimation du temps et du coût

Quels moyens pour l'analyse ? Diagrammes

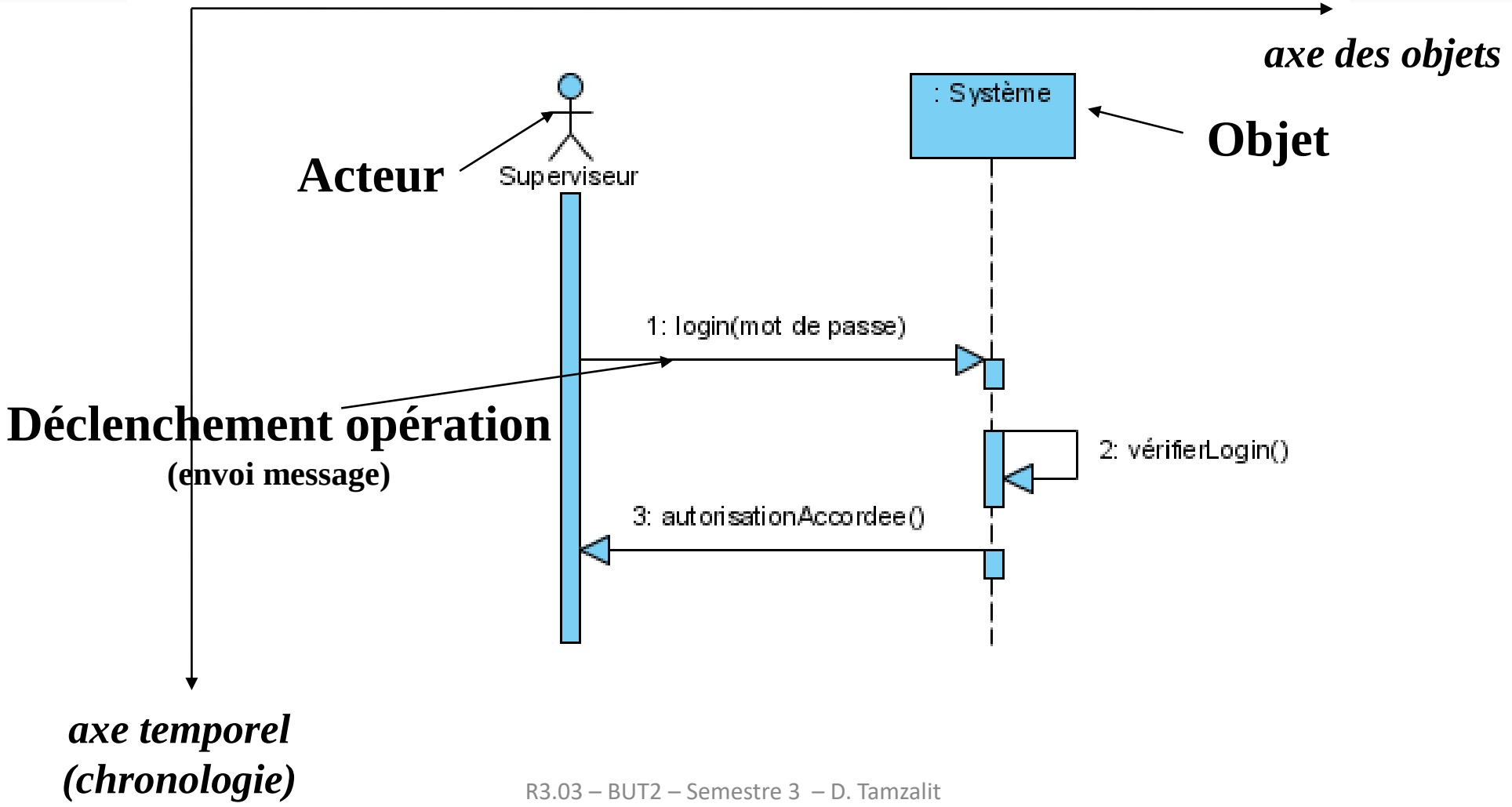
- Diagrammes de cas d'utilisation d'UML.
- Fonctionnement du système
- *Exemple* : cours de M. Mottu, en R2.10.
- Ne pas oublier les scénarios !



Diagrammes de séquences en phase d'analyse

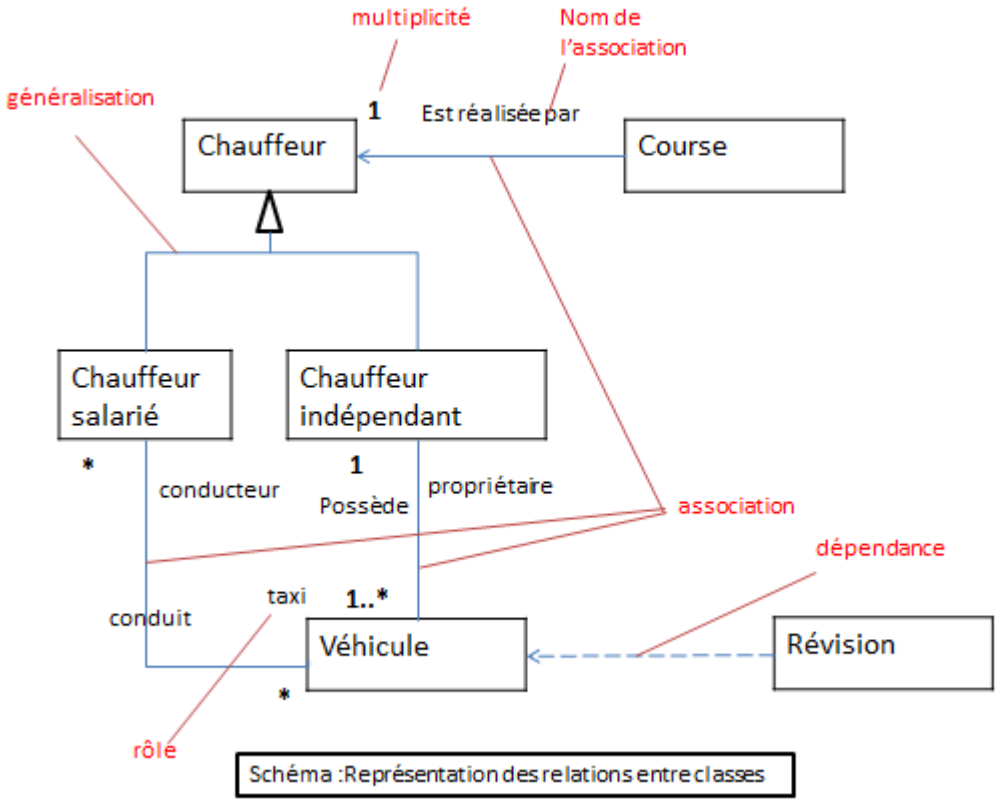
- Utilisation des diagrammes de séquences système (*boîte noire*).
- Généralement associé à un cas d'utilisation.
- Vue graphique d'un scénario (d'un cas d'utilisation) :
  - Met en évidence les interactions d'objets
  - Selon un point de vue temporel
- A deux dimensions :
  - verticale : représente le temps
  - horizontale : représente les différents objets.

Diagrammes de séquences en phase d'analyse



Quels moyens pour l'analyse ? Diagrammes

- Diagrammes de classes.
- Dimension statique du système : données et informations.
- Deux niveaux :
  - **Analyse**
  - Conception



https://fr.wikiversity.org/wiki/Mod%C3%A9lisation_UML/Le_diagramme_de_classes

Diagramme de classes d'analyse

- « On précise les entités métiers et leurs relations quelques méthodes-clefs pertinentes, certains types, etc. pour capturer le domaine métier » (confère ressource Intro Dev Objet en BUT1).
- Autre terminologie : conception architecturale, conception générale, analyse fonctionnelle
- Utilisation et utilité :
  - Outil de dialogues entre clients et utilisateurs
  - Comprendre les besoins métiers et les décrire
  - Extraire les modèles de l'expression des besoins
  - Uniquement les entités du domaine (métier)
  - Support pour la conception, l'implémentation et la maintenance
  - Description métier de ce que doit faire le système et comment il le fait sans la solution technique (trop tôt)

Diagramme de classes d'analyse

- Classes du domaine métier uniquement
- Use cases dans le langage du client mais diagramme d'analyse dans le langage du concepteur
- Objectif :
  - clarifier le domaine métier, se familiariser avec
  - pas de classes techniques
- Diagramme de classes d'analyse :
  - classes, attributs, associations
  - pas d 'opérations
- Démarche usuelle
  - diagramme de classes par use case
  - recouper les diagrammes de classes

Construction du DCA

- Recenser les objets métier dans :
  - les documents de définition des besoins
  - les documents d'analyse
- Noms ou groupes nominaux en classe, objet ou attribut.
- Doute entre attribut ou classe : préférer la classe puis affiner par la suite.
  - Exemple : adresse en attribut de Personne ou en classe associée à Personne ?
- Attention aux faux amis et doublons.
- Eviter les classes fonctionnelles qui font du traitement :
  - gestionnaire du planning, décodeur de message …

Construction du DCA

- Identifier les classes d'objets
  - puis garder les bonnes classes
  - Constitution du dictionnaire de données
- Identifier les associations
  - puis garder les bonnes associations
- Identifier les attributs
  - puis garder les bons attributs
- Raffiner au moyen de l'héritage
  - Généralisations et raffinages
- Itérer la modélisation
- Grouper les classes en modules/paquetages

Diagramme de classes d ’analyse

- Démarche
  - Dictionnaire des données/glossaire
  - Les classes (noms communs)
  - Les objets (noms propres ou nom référencés)
  - Les attributs (données simples qui qualifient une classe ou un objet)
  - Les éléments qui sont étrangers à notre problème ou qui n'y ont pas de rôle réel sont omis

Ressources

- Précédents cours DUT
- https://ca.insight.com/fr_CA/content-and-resources/2016/07152016-types-of-software-development-models.html
- https://membres-lig.imag.fr/dubousquet/docs/2.2_CyclesDeVie.pdf
- « GL – 2 Processus de développement, Cycles de vie », Lydie du Bousquet, IMAG.
- Support cours E. Syriani, Université de Montréal.