

IFT6803:

Génie logiciel du commerce électronique

Chapitre 1: Introduction

Section 3: Processus de développement

Sommaire

Chapitre 1, Section 3

« Processus de développement »

1.3.1 Cycle de vie du logiciel

1.3.2 Processus de développement

1.3.2.1 Modèle en cascade

1.3.2.2 Modèle en V

1.3.2.3 Modèle par prototypage

1.3.2.4 Modèles évolutifs

1.3.2.5 Modèle transformationnel

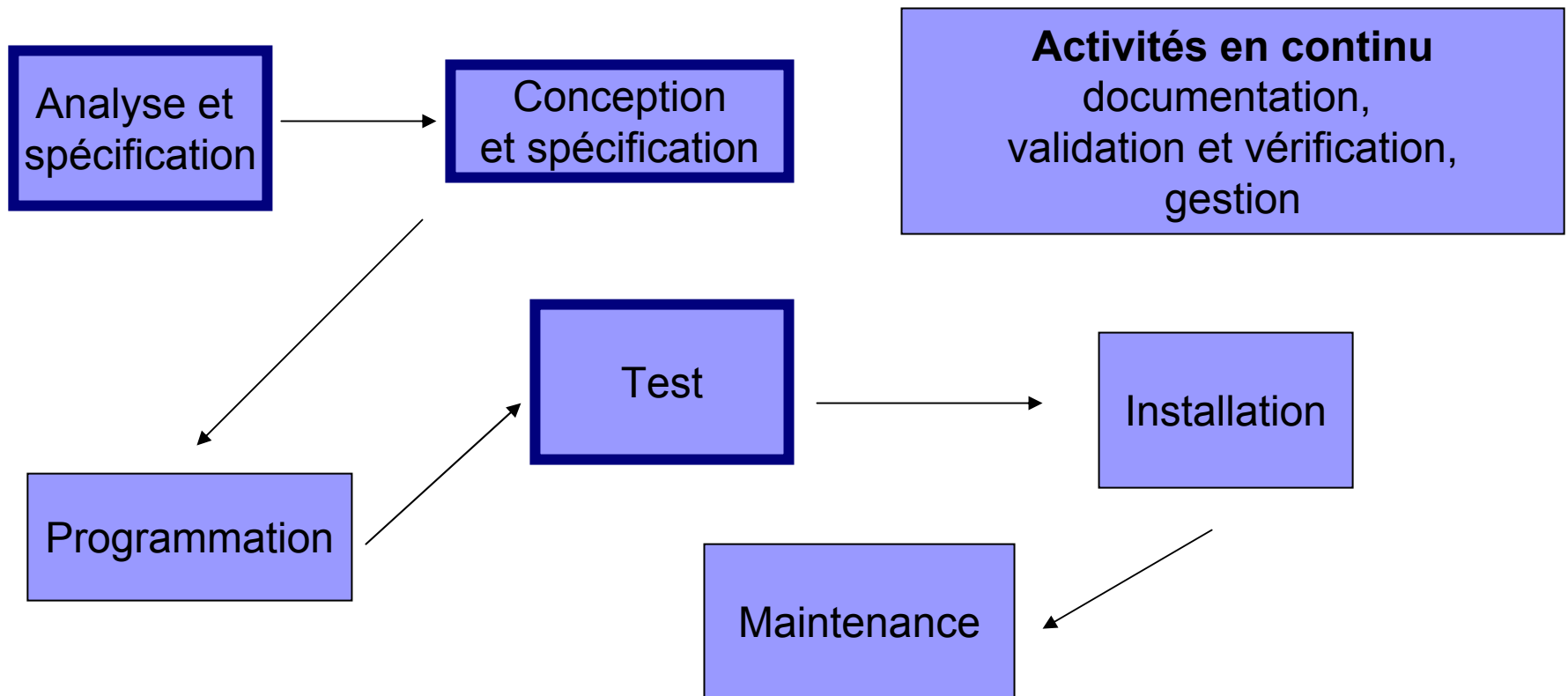
1.3.2.6 Modèle en spirale

1.3.2.7 Rapid Application Development (RAD)

1.3.3 Ingénierie système

1.3.1 Cycle de vie du logiciel

- Le logiciel est l'objet de plusieurs activités de développement au cours de sa vie...



Cycle de vie: analyse

■ Analyse

1. Etude préliminaire:

- définition globale du problème (diag. contexte)
- évaluation des stratégies possibles
- évaluation des ressources, coûts et délais
- Documents:** rapport d'analyse

2. Expression et analyse des besoins

- exigences fonctionnelles attendues: opérations, etc.
- qualités non fonctionnelles attendues: performance, portabilité, etc.
- Documents:** cahier des charges, document d'analyse (spécification) et plan de test du système



Quoi faire?

Cycle de vie: conception



Comment faire?

■ Conception

1. Conception architecturale: décomposition et organisation de l'application en composants plus simples (modules) définis par leurs interfaces (fonctions et services offerts).
 2. Conception détaillée: pour chaque module, description de la manière dont les services et fonctions sont réalisés:
 - algorithmes essentiels
 - structures de données utilisées, etc.
- **Documents**: document de conception (spécification), plan de test global et plan de test par module.

Cycle de vie: programmation

■ Programmation

- Traduction de la conception dans un langage de programmation
- **Documents:** dossiers de programmation, codes sources commentés.

Cycle de vie: test

■ Test

1. Tests unitaires:

- Tests avec les jeux d'essais par module selon le plan de test.


2. Tests d'intégration:

- Composition progressive des modules
- Tests des regroupement de modules

3. Test du système:

- Test en vraie grandeur du système complet selon le plan de test global.

- **Documents:** rapport de vérification par test.



Est-ce bien fait? Le programme répond-t-il à la spécification?

Cycle de vie: installation et maintenance

■ Installation:

- Mise en fonctionnement opérationnel chez les utilisateurs.
- Parfois restreint (dans un 1er temps) à des utilisateurs sélectionnés.

■ Maintenance:

- Maintenance corrective (corriger erreurs)
- Maintenance adaptative (modifications)
- Maintenance perfective (améliorations)
- *Aussi*: maintenance préventive (pour faciliter les opérations de maintenance à venir).

Cycle de vie: autres activités

■ Activités exécutées en continu

□ Validation

- Le produit répond-il aux besoins du client? «Construit-on le bon produit?»
- S'assurer de répondre aux exigences du client.

□ Vérification

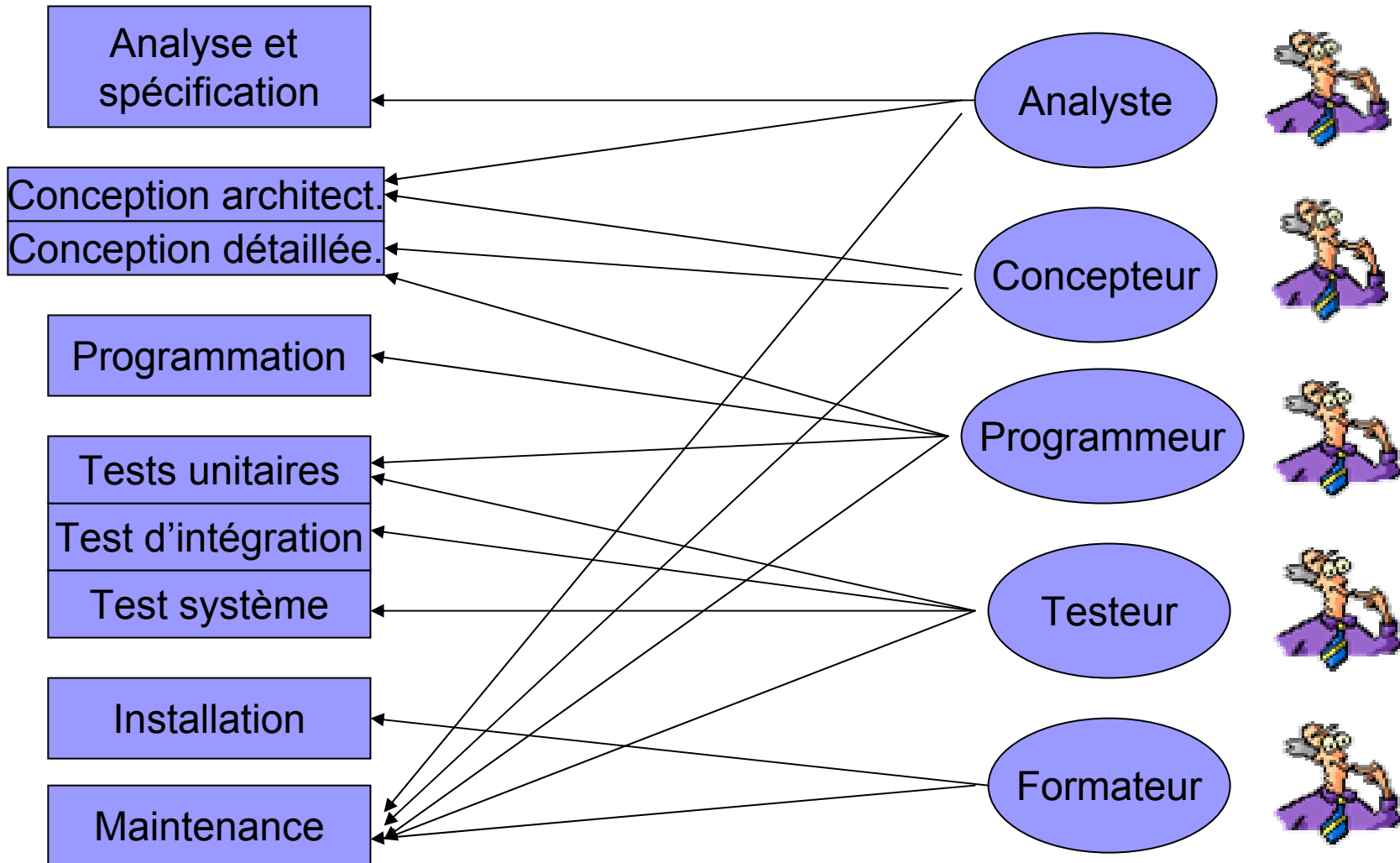
- Le produit est-t-il correct (par rapport à la spécification)? «Construit-on le produit comme il faut?»
- S'assurer de la qualité du produit (révisions et inspections)
- S'assurer de satisfaire la spécification.

□ Gestion

- Du processus de développement (suivi de projet, révision, etc.)
- De la configuration: politique de gestion des versions, des documents, politique de réutilisation
- Des ressources humaines
- Du risque

□ Documentation

L'équipe de développement: le rôle des membres dans le cycle de vie



1.3.2 Processus de développement

Processus : Description abstraite et idéalisée des différentes manières d'organiser les activités du développement d'un logiciel).

- Décrit un **ensemble de tâches ordonnées** impliquant
 - des **activités** (celles du cycle de vie)
 - des **contraintes** (sur le temps, ressources, etc.)
 - des **ressources** (humaines, matérielles, etc.)
- Doit être **«personnalisé»** pour l'entreprise de façon à définir l'ordonnancement idéal des activités.
 - **spécifier les artefacts** à produire (types de documents, format, échéancier).
 - **attribuer activités & artefacts** aux développeurs (cf. membres de l'équipe de développement)
 - proposer des **critères pour superviser** l'évolution du projet, ses résultats et prévoir plans futurs (vérification, validation, documentation, etc.)
 - proposer une **méthodologie pour gérer les changements** tant dans le processus et que le logiciel.

Processus de développement

- Le processus de développement est à définir en équipe, de préférence, car ce travail permet:
 - Compréhension commune des activités, ressources et contraintes.
 - Identification des incohérences, redondances et omissions dans le processus.
 - Identification des activités permettant de réaliser les objectifs de qualité.
 - Adaptation du processus aux besoins spécifiques du projet et compréhension de ses particularités.

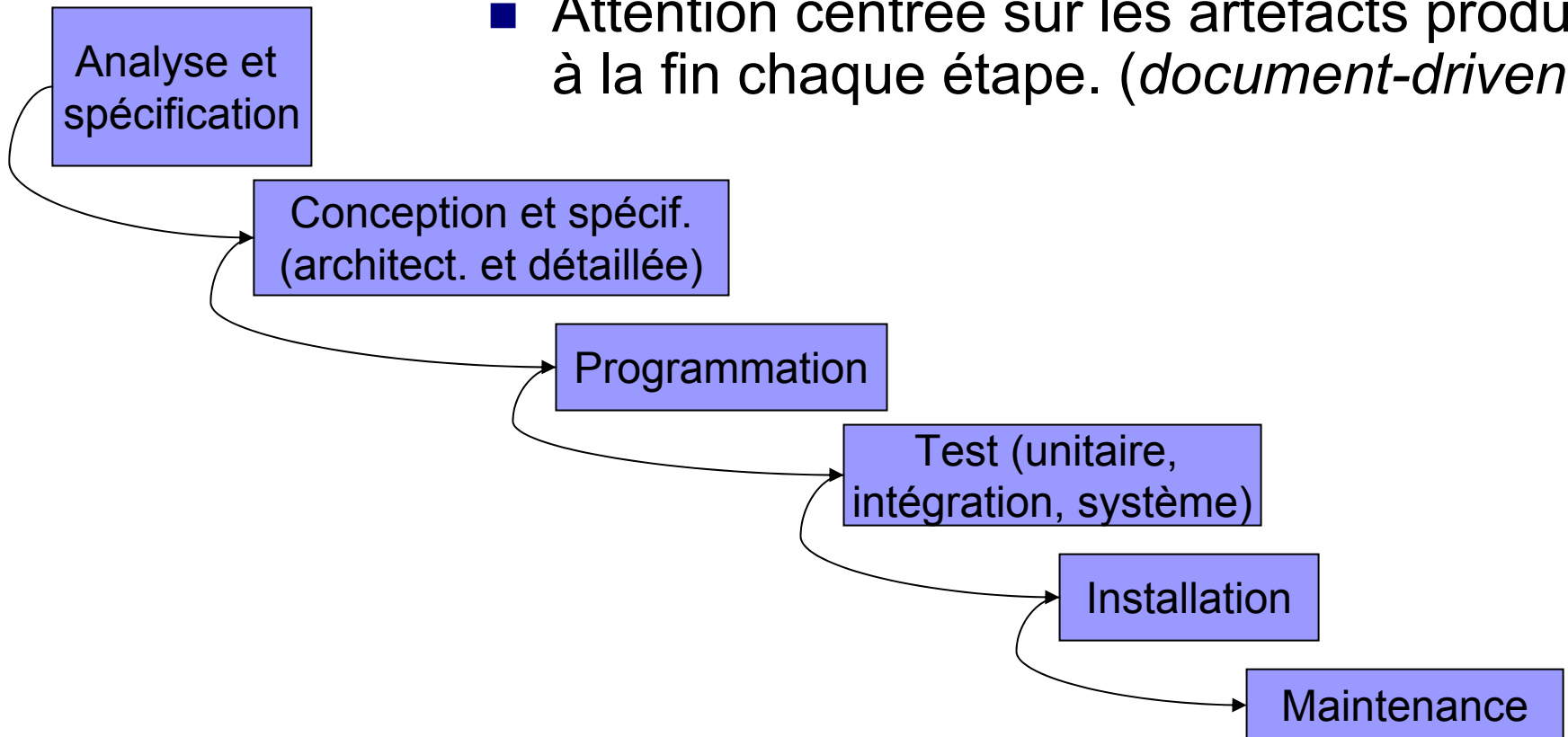
Processus de développement

Quelques modèles existants
... à personnaliser

- Modèle en cascade
- Modèle en V
- Modèle par prototypage
- Modèles évolutifs: incrémental et itératif
- Modèle transformationnel
- Modèle en spirale
- Modèle RAD

1.3.2.1 Modèle en cascade

- Etapes réalisées de façon strictement séquentielle.
- Attention centrée sur les artefacts produits à la fin chaque étape. (*document-driven*)



Modèle en cascade

Avantages

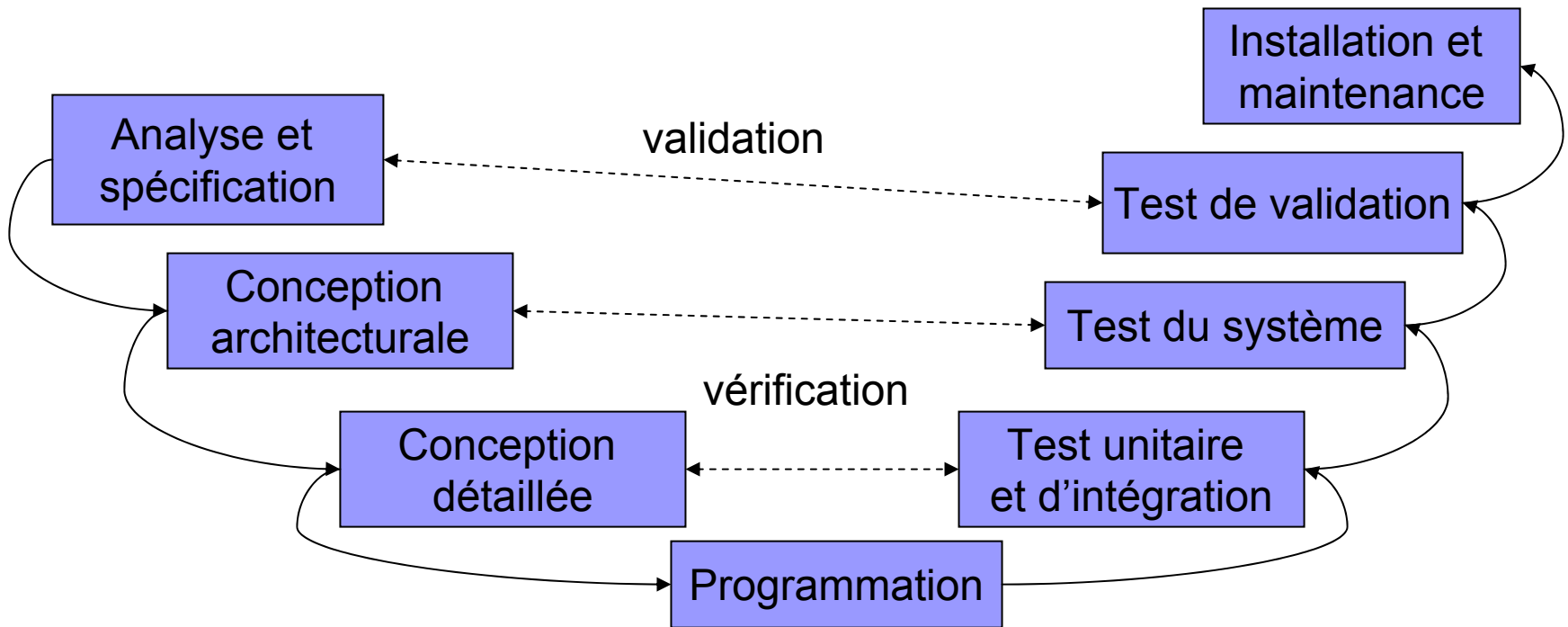
- Plan **simple** de ce qu'il faut faire.
- **Facile** à comprendre
- Cette première définition a permis la normalisation des cadres conceptuels et terminologiques des différentes activités.

Inconvénients

- Hypothèses souvent **irréalistes** que l'on peut dès le départ définir complètement et en détail
 - ce qu'on veut réaliser
 - les résultats intermédiaires obtenus
- Ne reflète pas la façon dont le code est réellement développé.
- Trop **rigide**, manque de flexibilité pour imprévus.
- **Pas de «feedback»** avant la livraison au client....

1.3.2.2 Modèle en V

- Variation du modèle en cascade
- Attention centrée sur la correction: vérification et validation



Modèle en V

Avantages

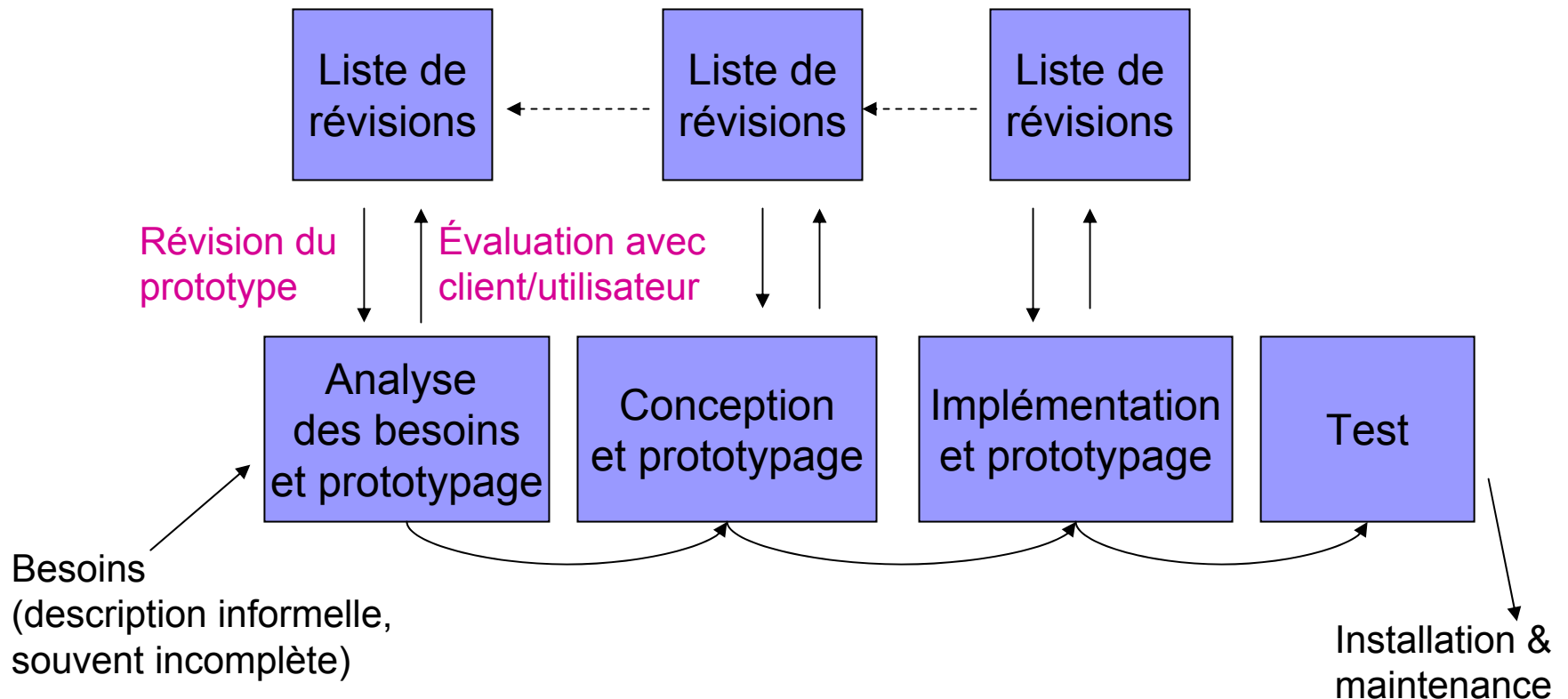
- Montre comment les activités de «test» **sont liées** à celles d'«analyse et de conception».

Inconvénients

- **Manque de « feedback ».**
Pas de résultats intermédiaires dont on peut discuter avec le client.

1.3.2.3 Modèle par prototypage

- À chaque étape, un ou plusieurs prototypes sont soumis au client pour évaluation/révision.
- Permet d'examiner et d'explorer certains aspects du système pour évaluer et choisir les meilleures stratégies/solutions.



Modèle par prototypage

Types de prototypes

- Prototype jetable: maquette exploratoire développée pour mieux comprendre les besoins du client, évaluer différentes solutions, etc. Développé rapidement et jeté ensuite.
- Prototype évolutif (ou réutilisable): maquette destinée à être complétée/optimisée dans les prochaines étapes du développement jusqu'à l'obtention du produit final.

Modèle par prototypage

Avantages

- Améliore la compréhension mutuelle du problème entre client, développeur et utilisateur.
- Favorise les activités de vérification et de validation intermédiaires.

Inconvénients

- Tentation d'abrégier le processus et de se contenter d'un prototype incomplet...

1.3.2.4 Modèles évolutifs

- Le processus de développement d'un logiciel peut être très long (mois, années...)
- Le logiciel est un produit de nature évolutive...
- Pour être compétitif, il faut **réduire le temps de mise en marché** et pouvoir
 - produire versions partielles du système
 - ajouter graduellement de qualités et fonctionnalités.

Solution:

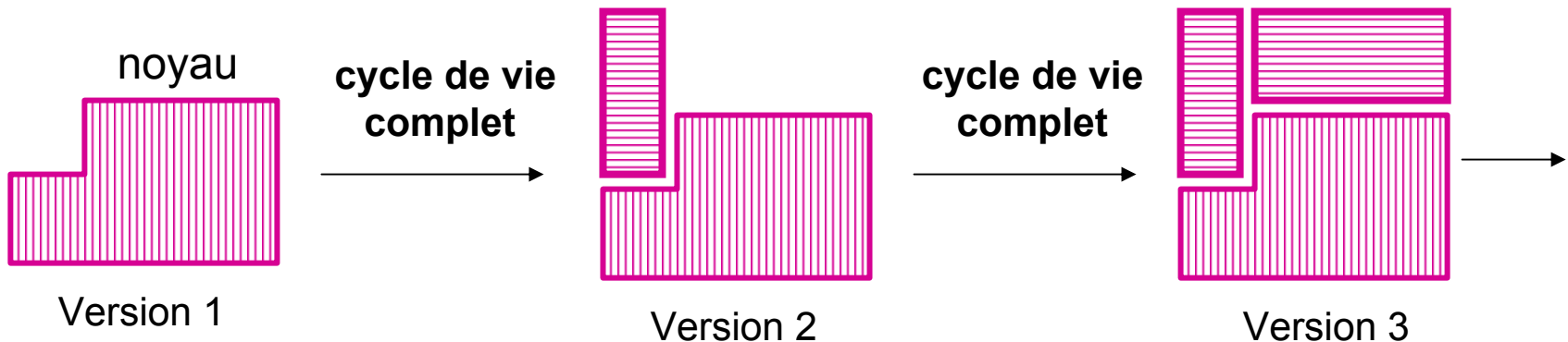
Modèle évolutif: développement au cours duquel des versions du logiciel de plus en plus complexes et détaillées sont développées.

- Deux versions du logiciel existent alors en parallèle
 - logiciel opérationnel (version utilisée actuellement)
 - logiciel en développement (prochaine version)

Modèle évolutif - incrémental

■ Approche incrémentale

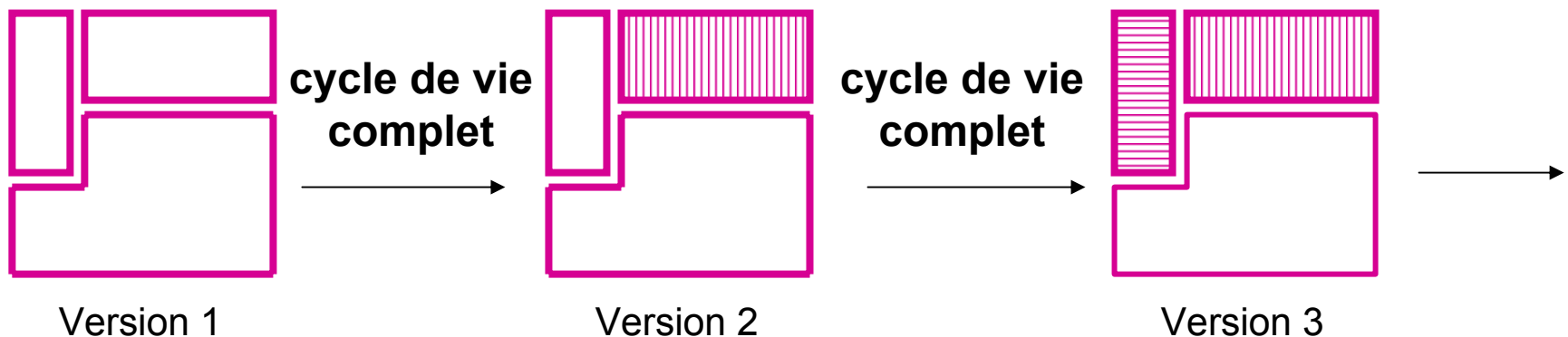
- division du système en sous-systèmes (un par fonctionnalité).
- 1re version: système partiel.
- Chaque nouvelle version (incrément) ajoute une nouvelle fonctionnalité.



Modèle évolutif - itératif

■ Approche itérative

- division du système en sous-systèmes (un par fonctionnalité).
- 1re version: coquille complète du système.
- Chaque nouvelle version apporte une modification/amélioration à une fonctionnalité.



Modèles évolutifs

Avantages

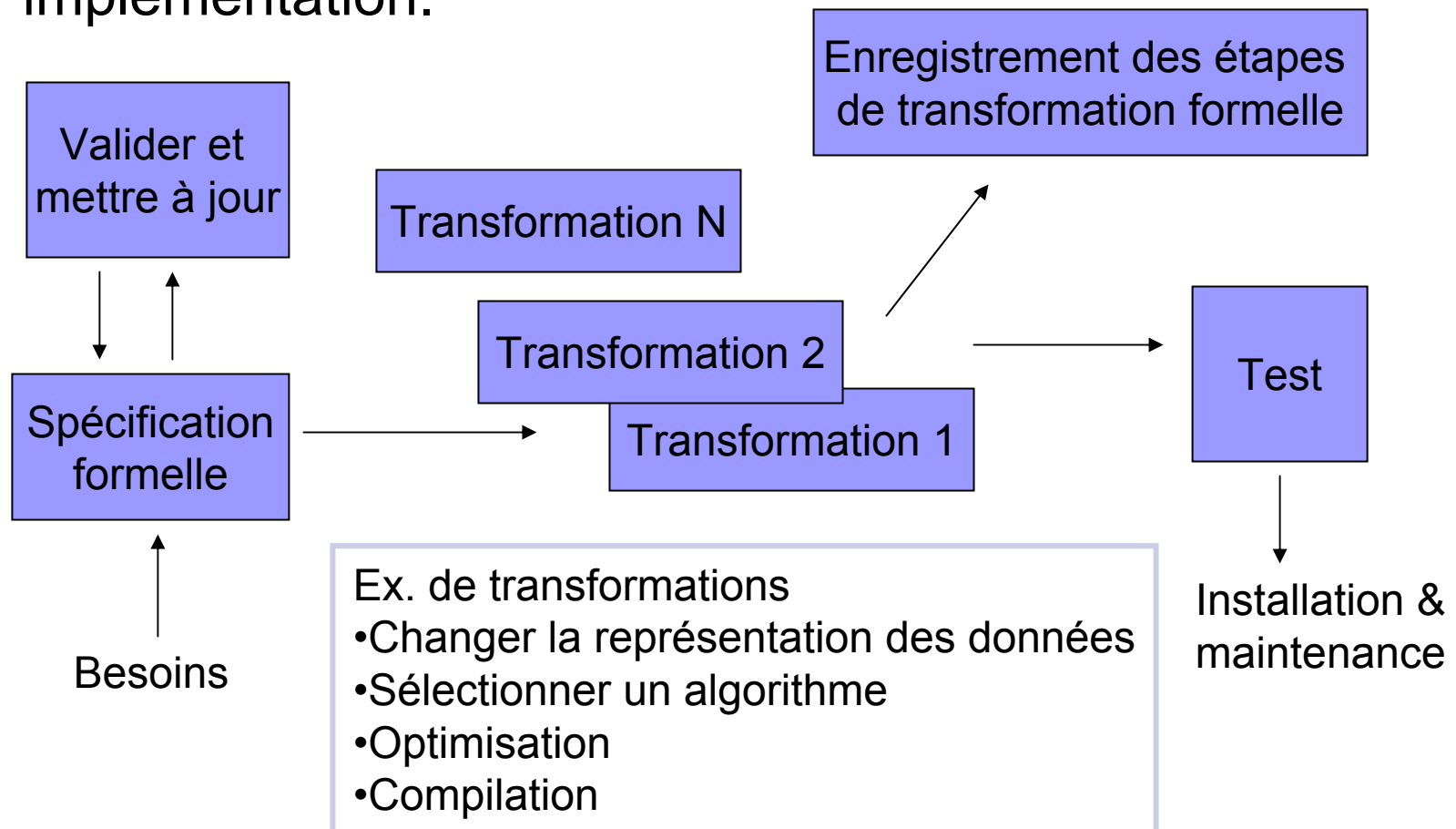
- Formation précoce des utilisateurs. Réponse rapide possible.
- Création précoce de nouveaux marchés pour nouvelles fonctionnalités.
- Focus sur nouveau domaine d'expertise à chaque étape (version).
- Détection précoces des problèmes imprévus (correction immédiate du système en développement).

Inconvénients

- Risque de la remise en cause du noyau (fonctionnalités de base) au cours du développement.

1.3.4.5 Modèle transformationnel

- Développement consistant en une séquence d'étapes qui transforment graduellement une spécification en implémentation.



Modèle transformationnel

Avantages

- Chaque transformation est **prouvée formellement**.
Preuves gardées dans un registre.
 - Favorise le développement de programme correct et documenté formellement.
- Modifications répercutées dans le code & spécification.
Pas de divergences.
- Environnement et outils
 - Développer des composants réutilisables.
 - Outils intelligent d'aide à la preuve.

Inconvénients

- Demeure une approche expérimentale et de recherche.
- Exige une certaine **expertise en méthodes formelles** pour le développement de systèmes complexes.

1.3.2.6 Modèle en spirale (Boehm 88)

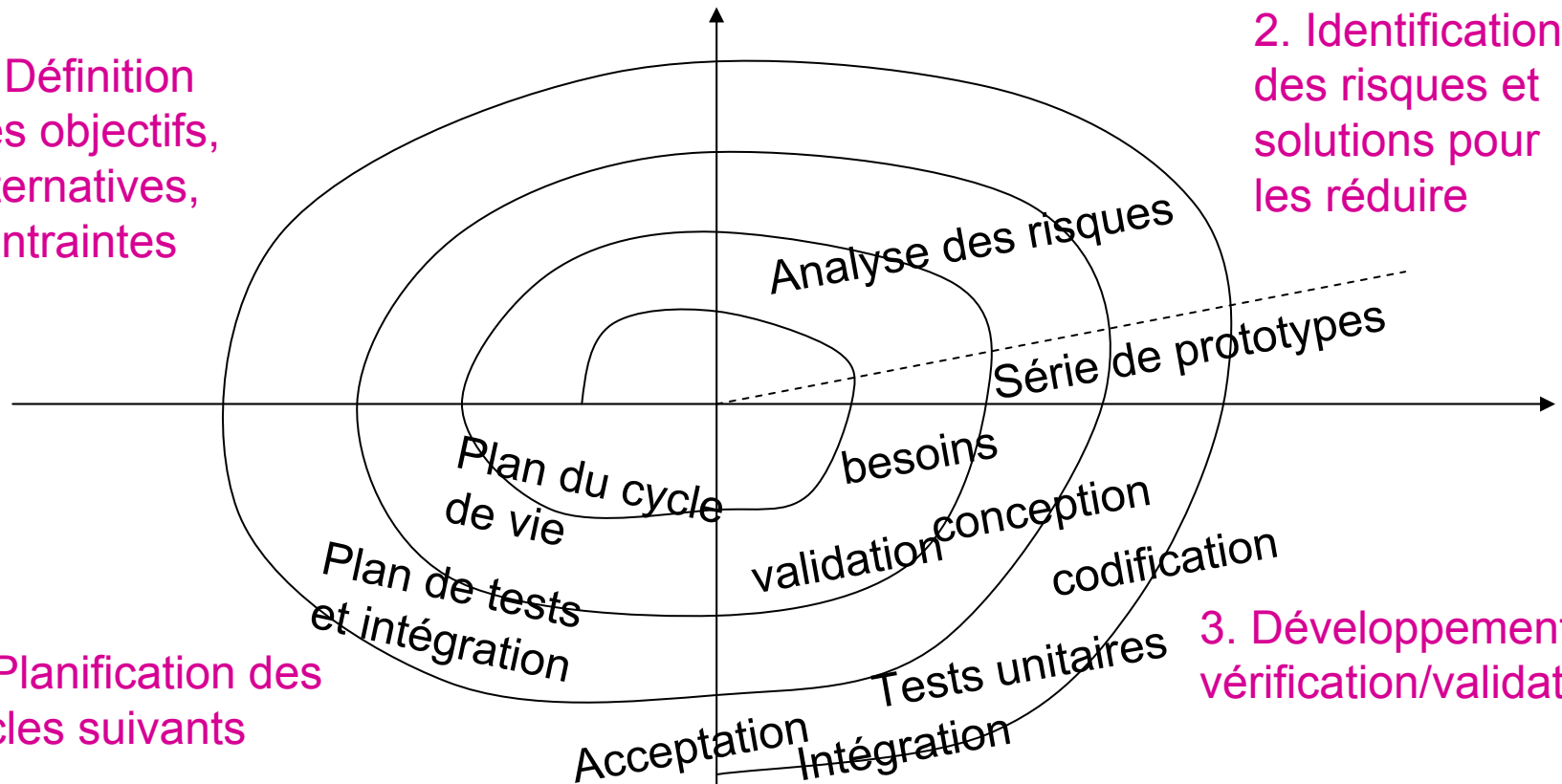
- Modèle cyclique. Chaque cycle est composé de 4 étapes
- A chaque cycle:
 - spécification de nouveaux besoins
 - critères de robustesse deviennent des critères de correction

1. Définition des objectifs, alternatives, contraintes

2. Identification des risques et solutions pour les réduire

3. Développement et vérification/validation

4. Planification des cycles suivants



Modèle en spirale

Attention centrée sur l'évaluation des risques

- RISQUE = ce qui met en péril processus et qualité.
 - défaillance de ressources humaines
 - calendriers et budgets irréalistes
 - développement de fonctionnalités inappropriées
 - développement d'interfaces inadéquates
 - produit «plaqué or» (production non rentable)
 - volatilité des besoins (viser stabilité, noyau)
 - problèmes de performances exigences démesurées par rapport à la technologie.

Modèle en spirale

Avantages

- Permet d'établir le modèle de développement le plus adéquat en regard du risque.
- **Méta-modèle**: tous les autres modèles de développement constituent une variante de la spirale!

Inconvénients

- Comparables à ceux des autres modèles...

1.3.2.7 Rapid Application Development (James Martin 91)

Définition

Approche de construction de SI qui combine l'utilisation d'outils CASE, du prototypage incrémental et l'observation d'échéanciers rigoureux dans une méthodologie efficace visant à réduire les coûts de développement tout en assurant la qualité du produit.

Principes à la base de ce processus

Dans certaines situations...

- Une solution satisfaisant **4/5 des exigences** peut être produite dans le **1/4 du temps** requis pour développer une solution complète.
- Les **exigences de rentabilité** d'un système peuvent être entièrement satisfaites même si certaines exigences fonctionnelles ne le sont pas encore tout à fait.
- **L'acceptabilité d'un système** peut être déterminée sur la base convenue d'un **sous-ensemble minimum d'exigences utiles** plutôt que sur toutes les exigences définies.

Rapid Application Development

Problèmes auxquels s'attaque le RAD

Avec les méthodes conventionnelles...

- le client doit attendre longtemps avant de pouvoir juger des premiers résultats.
- le développement est parfois tellement long que les exigences de rentabilité exprimées par le client ont tout à fait changé une fois le système prêt.
- Rien n'est livré avant que le processus de développement ne soit complètement terminé.

Rapid Application Development

Pourquoi utiliser le RAD ?

■ Mauvaises raisons:

- Pour éviter le dépassement des coûts estimés
- (NON: l'équipe doit déjà être formée à la gestion des coûts)
- Pour éviter le dépassement des échéanciers.
- (NON: l'équipe doit déjà être sensibilisée à la gestion du temps).

■ Bonnes raisons:

- Pour converger rapidement vers une solution conceptuelle acceptable pour le client et réalisable par les développeurs.
- Pour limiter les menaces de changements tardifs auxquelles le projet s'expose.
- Pour réduire les temps de développement (quitte à faire certains compromis sur les coûts et les exigences de qualité).

Rapid Application Development

Temps vs coûts vs qualité

- Les compromis détermineront le rythme du développement
 - Développement efficace: équilibre entre coût, temps et qualité.
 - RAD normal: compromis au niveau des coûts et de la qualité pour favoriser la réduction du temps de développement.
 - RAD extrême: Codage intensif!! (coûts plus élevés, qualité moindre...)

Rapid Application Development

Temps vs coûts vs qualité

- Il faut que le client soit prêt à négocier les coûts ou certaines exigences de qualité...
 - Bonnes chances de succès du processus si coûts **OU** qualités négociables.
 - Meilleures chances de succès si coût **ET** qualité négociables.
 - N.b. Négocier la qualité ne signifie pas accepter un produit qui contient des erreurs (!) ... mais tolérer quelques compromis au niveau de la performance, de la convivialité, de la portabilité, etc.
- Certains objectifs ne seront peut-être pas réalisable: assurer le *minimum* d'erreurs, assurer la *meilleure* satisfaction du client, assurer les coûts *les plus bas*...

Rapid Application Development

Contraintes

- Pour être accepté, chaque livrable (version du logiciel, documents, etc.) doit répondre à un besoin de l'entreprise.
- Toute partie pouvant influencer la définition des exigences doit être représentée dans l'équipe de développement tout au long du processus.
- Les clients, les développeurs et gestionnaires doivent accepter des documents informels
- L'équipe de développement doit avoir l'autorisation de prendre des décisions généralement réservées aux gestionnaires.
- Le projet ne doit pas durer plus de 6 mois.
- Le développement itératif doit converger vers une solution rentable et acceptable pour l'entreprise.

Rapid Application Development

Caractéristiques

A. **Équipes hybrides**

- Environ 6 personnes incluant les développeurs, les utilisateurs du système ainsi que toute personne impliquée dans la définition des exigences du projet (client)
- Les développeurs choisis devront être flexibles et polyvalents de façon à pouvoir assumer à la fois les rôles d'analyse, de concepteurs et de programmeurs.

B. **Utilisation d'outils spécialisés supportant**

- Développement visuel, création de prototypes (jetables et incrémentales), langages multiples, planification, travail d'équipe et en collaboration, utilisation de composants réutilisables, utilisation d'API, gestion de versions.

C. **Encadrement stricte du temps (timeboxing)**

- Les aspects secondaires sont éliminés (définitivement ou temporairement) afin de respecter des échéanciers fixes.

Rapid Application Development

Caractéristiques

D. Développement par prototypage itératif et incrémental

1. Réunion JAD (Joint Application Development)

Les clients/utilisateurs et les concepteurs/développeurs se rencontrent pour une session de remue-méninges au cours de laquelle on dressera la liste générale des besoins.

- Les concepteurs/développeurs discutent et écoutent.
- Les clients/utilisateurs discutent et écoutent.

Infos supplémentaires ([ici html](#)).

Rapid Application Development

Caractéristiques

2. Répéter jusqu'à ce que le système soit terminé:

- Les développeurs complètent le prototype selon les exigences actuelles.
- Les concepteurs révisent le prototype.
- Les clients/utilisateurs essaient le prototype et font la mise à jour de leurs exigences.
- Réunion du groupe de mise au point:
 - Les clients et développeurs se rencontrent pour réviser le produit ensemble, raffiner les exigences et faire la liste des demandes de modifications.
 - Les développeurs écoutent.
 - Les client parlent.
- Les demandes de changement et nouvelles exigences sont cadrées dans l'horaire.
 - Les changements et exigences secondaires qui ne peuvent être cadrés dans l'horaire doivent être éliminés.

Rapid Application Development

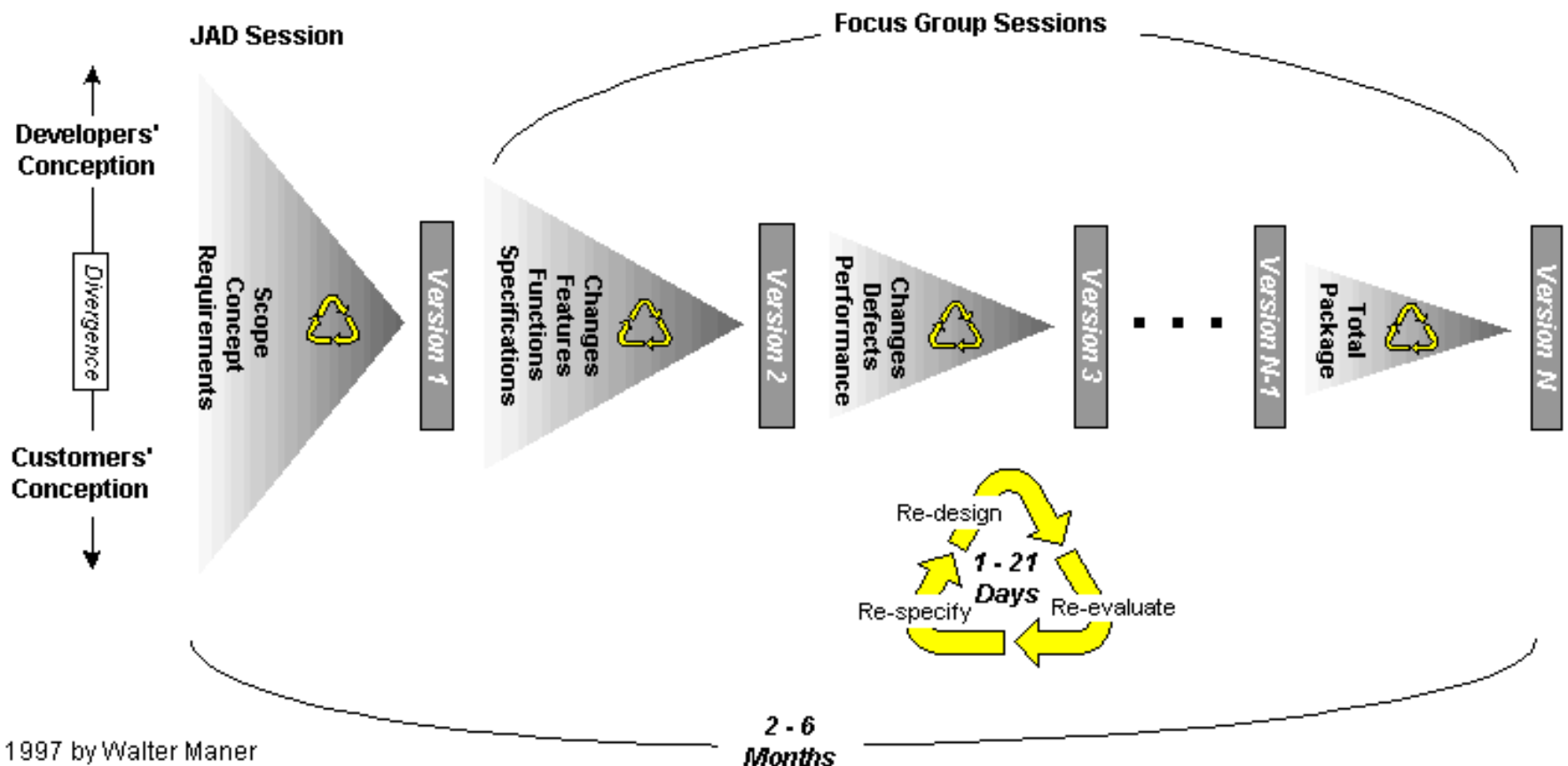
Caractéristiques

Remarques

- La durée de chaque itération varie entre 1 journée et 3 semaines.
- Les prototypes exploratoires peuvent se transformer en prototypes opérationnels.
- Session du groupe de mise au point:
 - Dure environ 2 heures.
 - Est dirigé par un modérateur d'expérience qui incite le groupe à se centrer sur les points importants qui nécessitent d'être éclaircis.
 - Fait l'objet d'un rapport rédigé par le modérateur.

Rapid Application Development

RAPID APPLICATION DEVELOPMENT USING ITERATIVE PROTOTYPING



(c) 1997 by Walter Maner

Rapid Application Development

Le RAD est approprié lorsque...

- Pour les applications autonomes.
- Les librairies standards (API) peuvent être exploitées.
- La performance et la fiabilité ne sont pas des facteurs critiques.
- La portée du projet est limitée (maintenance légère).
- Le système peut être décomposé en plusieurs modules indépendants.
- Le produit est destiné au marché hautement spécialisé des systèmes d'information (mise en marché rapide).
- Le projet a des contraintes horaires bien fixées (timeboxing).
- La technologie requise date de plus d'une année.

Rapid Application Development

Le RAD n'est pas approprié...

- Pour les applications nécessitant d'interagir avec d'autres.
- Pour les applications critiques.
- Lorsqu'une performance optimale est requise.
- Lorsqu'une grande fiabilité est exigée.
- Lorsque la technologie requise n'est pas bien maîtrisée...

Rapid Application Development

Avantages

- Les applications sont parfois plus facilement portables (utilisation d'abstractions de haut niveau, scripts, etc.)
- Visibilité précoce et meilleure rétroaction (feedback)
- Plus grande flexibilité de modification.
- Réduction du codage "manuel" (réutilisation, générateur de code, etc.)
- Implique davantage les utilisateurs (qui participent aux réunions)
- Possiblement moins d'erreurs (utilisation d'outils CASE)
- Coûts possiblement réduits (le temps c'est de l'argent...)
- Cycle de développement plus court.
- Aspect et convivialité standardisé (API et composants réutilisables offrent une apparence cohérente).

Rapid Application Development

Désavantages

- Coûts supplémentaires pour les outils utilisés.
- Plus difficile de mesurer les progrès du développement
- Moins efficace (code généré...)
- Perte de précision scientifique (pas de méthodes formelles)
- Fiabilité moindre, plus d'erreurs (si le code a été écrit à un rythme d'enfer!)
- Fonctionnalités et facilités restreintes (à cause du temps réduit)
- Recours aux composants
 - Ajout de fonctionnalités non nécessaires
 - Problèmes de droits d'auteurs
- Expérience réussie difficile à répéter (il n'y en a pas deux pareilles!)

Rapid Application Development

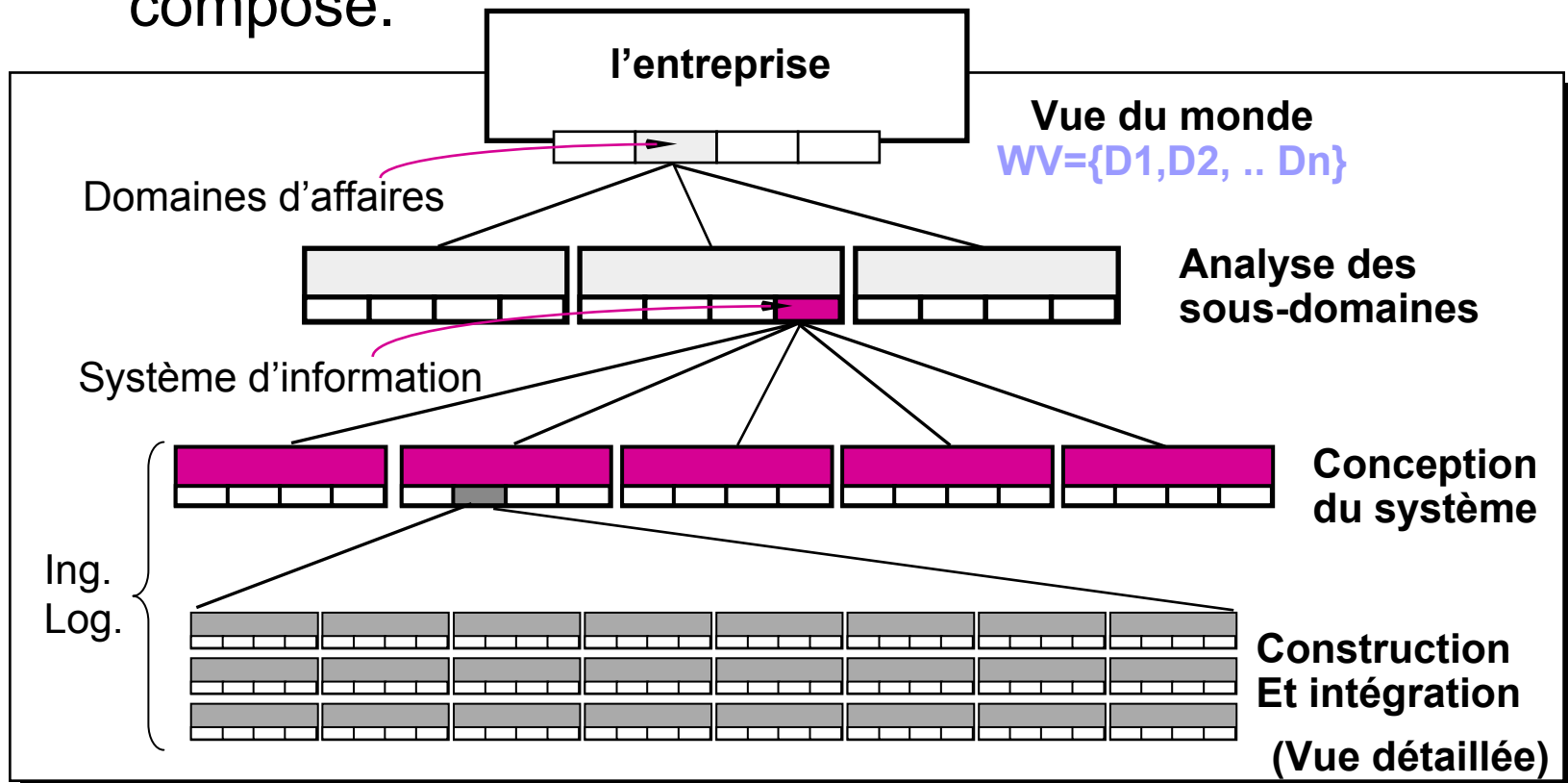
Références

- Présentation du processus RAD et d'un outil CASE ([pdf ici](http://www.casemaker.com/download/products/totem/rad_wp.pdf)): http://www.casemaker.com/download/products/totem/rad_wp.pdf
- Un autre processus de développement rapide:
 - Extreme Programming ([html ici](http://www.extremeprogramming.org/)): <http://www.extremeprogramming.org/>

1.3.3 Ingénierie système

Le logiciel à développer s'inscrit souvent dans un système général complexe.

- Situer le logiciel dans son contexte global
- Identifier la hiérarchie des éléments qui le compose.

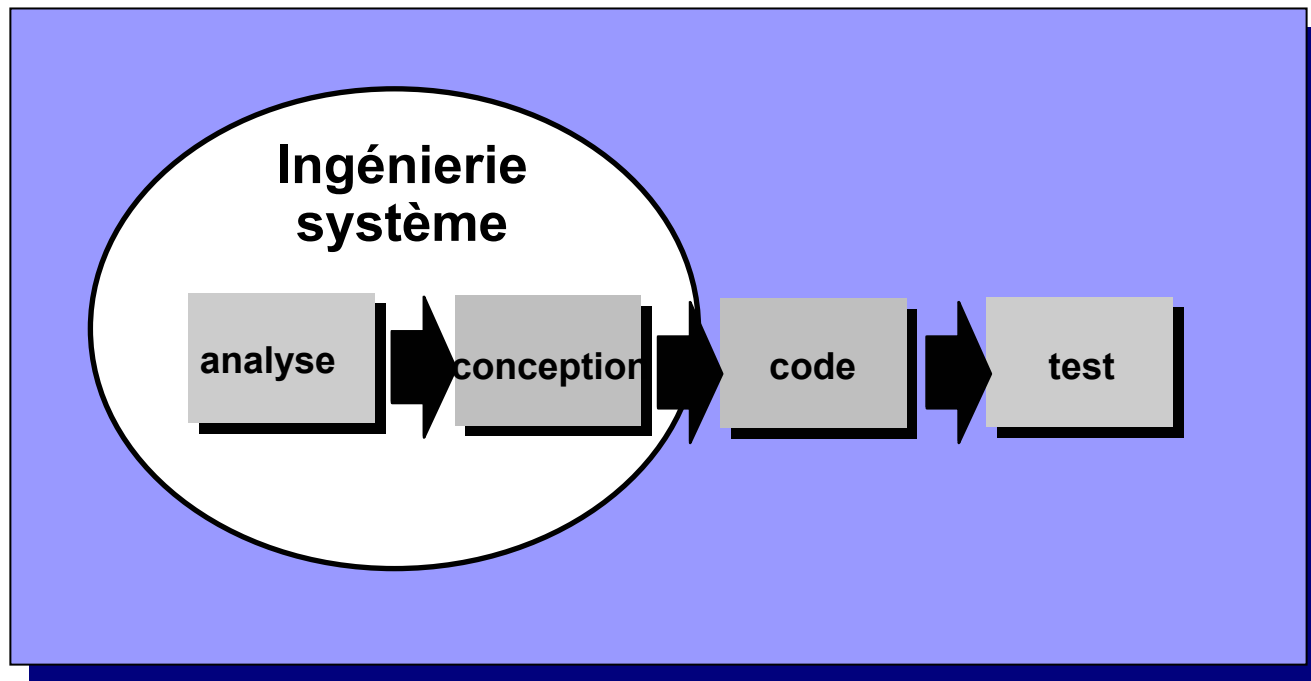


Ingénierie système

- Étude des **systèmes complexes** de type informatique (lois qui gouvernent les grands systèmes).
- Etablissement des **besoins relatifs** à tous les éléments du **système général** (matériel, bases de données, utilisateurs, etc.) dans lequel devra s'intégrer le logiciel.
Vue élargie des besoins.
- Discipline qui traite de la conception de systèmes informatiques, notamment de l'analyse des besoins, de la détermination des spécifications techniques et du type d'architecture du système à développer. (*Grand dictionnaire terminologique*)

Ingénierie système

- Les connaissances de l'ingénieur système sont principalement requises dans les phases d'**analyse** et de **conception** proprement dite du système, lesquelles phases précèdent celles de la réalisation.



Ingénierie système

Diagramme de contexte

On peut décrire le cadre du système dans lequel le logiciel s'inscrit par un diagramme de contexte.

Exercice: Donnez le diagramme de contexte d'un système de contrôle d'un ascenseur.