Méthodes de conception orientées objets et agiles

- Introduction
- Découpage d'un projet
- Cycles de vie logiciel
- Le processus unifié.

Maîtrise d'ouvrage et maîtrise d'oeuvre

- Maîtrise d'ouvrage (MOA)
  - Le maître d'ouvrage est le demandeur du livrable
  - Représenté par un responsable de projet ou par une direction de projet.
  - Le maître d'ouvrage peut faire appel à un intermédiaire pour l'expression des besoins du projet appelé maître d'ouvrage délégué.
- Le maître d'œuvre (MOE) doit réalisé le livrable demandé selon les conditions (délai, qualité coûts) définies dans le contrat.
  - Le MOE doit désigner un chef de projet (pour la conduite du projet)
  - Le MOE peut faire appel à des sous traitant.

Découpage d'un projet

- Découpage du projet: on décomposer un projet en plusieurs étapes
  - Caractéristiques d'une étape
    - Chaque étape conduit à un résultat défini
    - Chaque étape nécessite des ressources définies
    - Contraintes d'enchaînement entre étapes (séquentiel, parallèle)
    - Une étape peut être décomposée en sous étapes
  - Le découpage peut être réalisé selon deux dimensions: temporelle (succession d'étapes, de phases et de tâches) ou structurelle (modularisation).

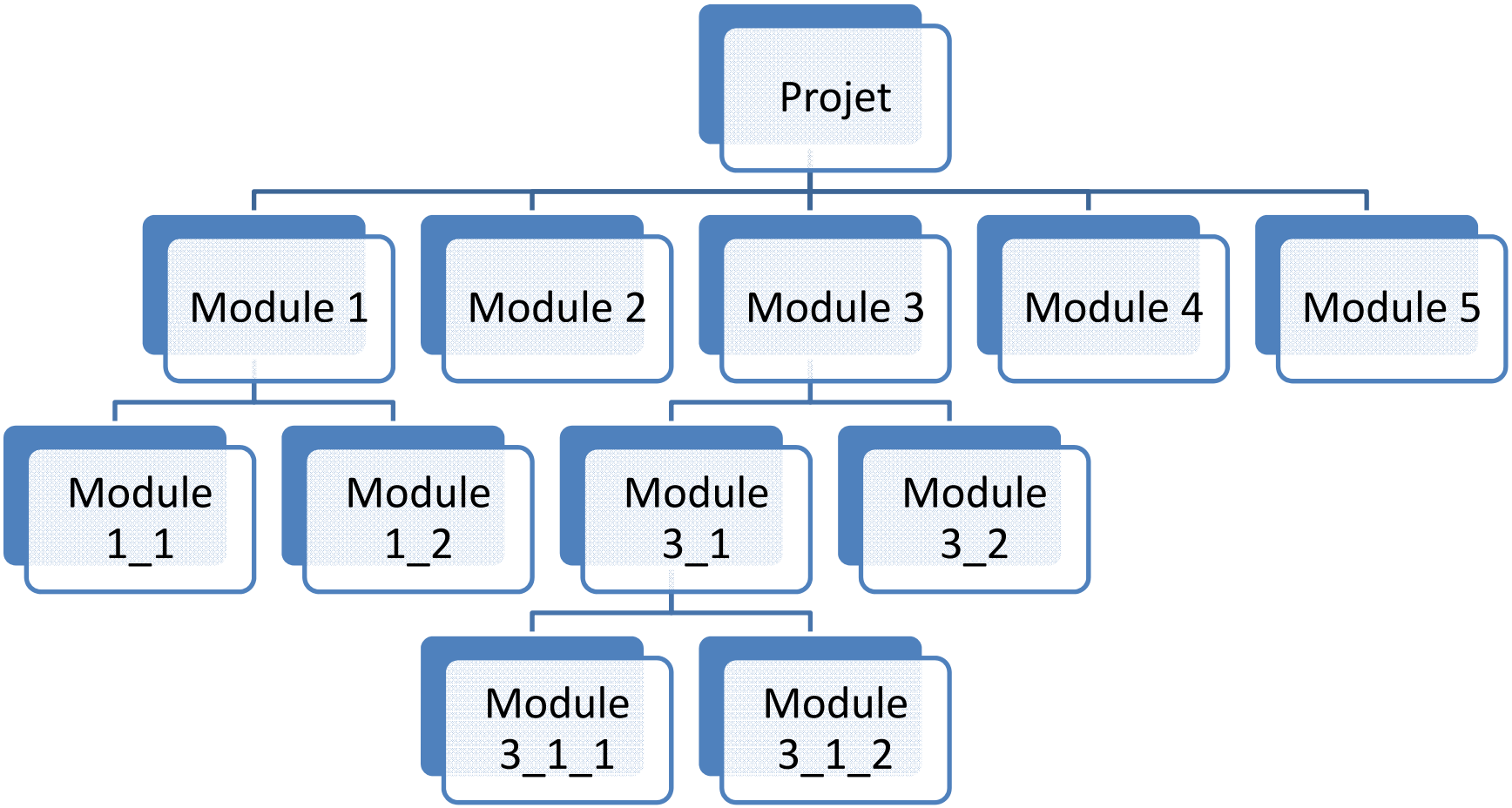
Objectifs du découpage

- Découpage structurel
  - Réduire la complexité du système
- Découpage temporel
  - Définition de jalons intermédiaires permettant de valider la conformité du résultat de l'étape par rapport aux exigences
  - Maîtriser les risques, les coûts, el les délais.

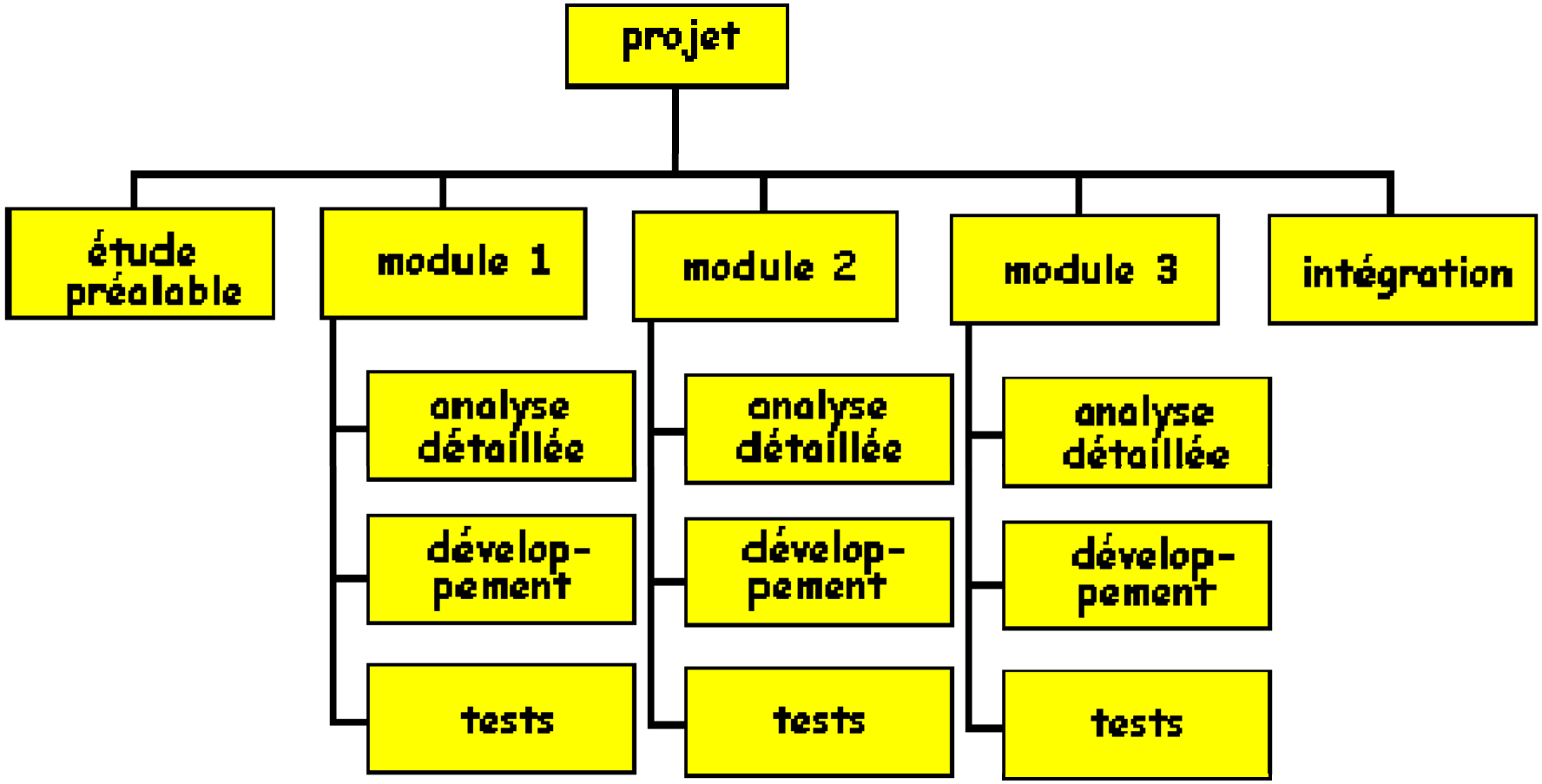
Méthodes

- Découpage structurel
  - PBS (Product Breakdown Structure)
    - Décomposition structurelle du produit
  - Work Breakdown Structure
    - Décomposition structurelle des tâches
  - OBS (Organization Breakdown Structure): affectation d'un responsable pour chaque unité

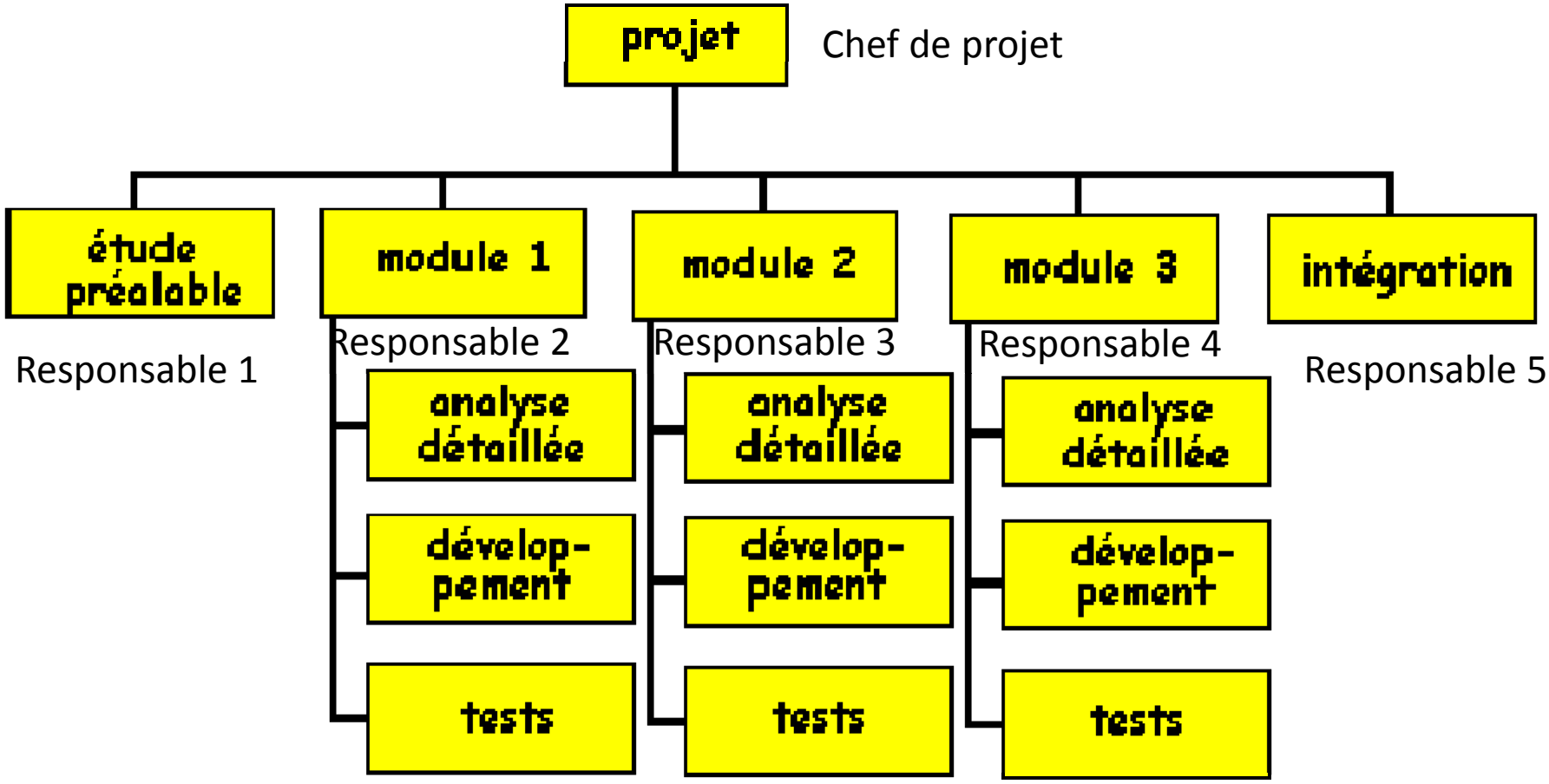
PBS


Projet
Module 1
Module 2
Module 3
Module 4
Module 5
Module 1_1
Module 1_2
Module 3_1
Module 3_2
Module 3_1_1
Module 3_1_2

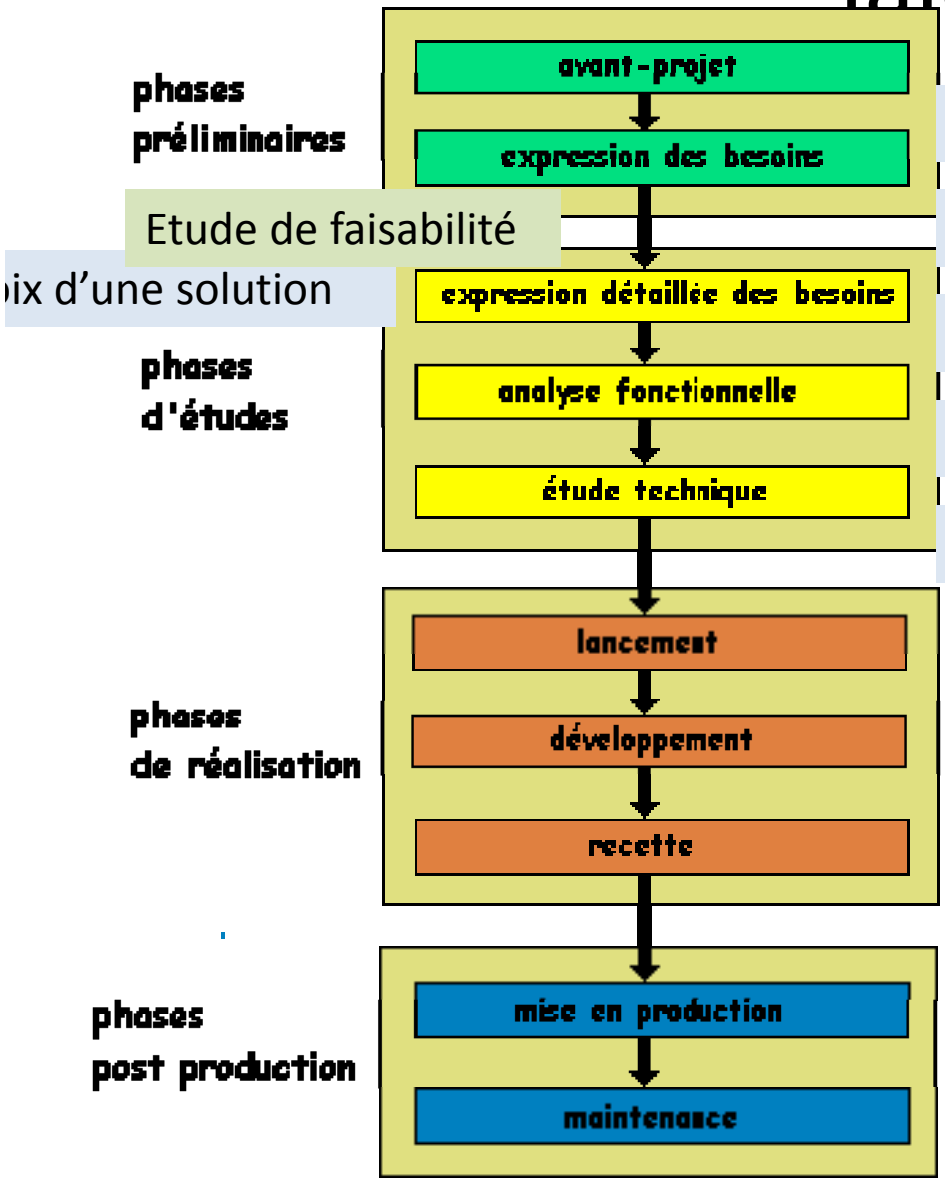
WBS


projet
étude préalable
module 1
module 2
module 3
intégration
analyse détaillée
dévelop-pement
tests
analyse détaillée
dévelop-pement
tests
analyse détaillée
dévelop-pement
tests

OBS


projet
Chef de projet
étude préalable
Responsable 1
module 1
Responsable 2
analyse détaillée
dévelop-pement
tests
module 2
Responsable 3
analyse détaillée
dévelop-pement
tests
module 3
Responsable 4
analyse détaillée
dévelop-pement
tests
intégration
Responsable 5

Découpage Temporel: Approche par jalons



Jalon de lancement du projet

Jalon de validation des besoins

production d'un document.

cahier des charges fonctionnel

cahier des charges techniques

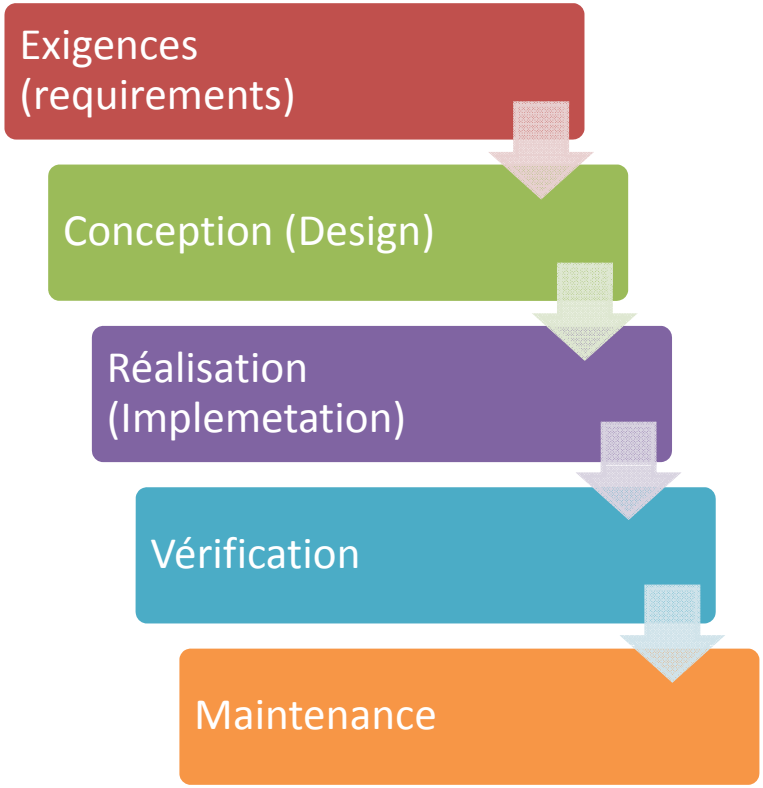
documentation projet

Jalons

Cycles de vie logiciels

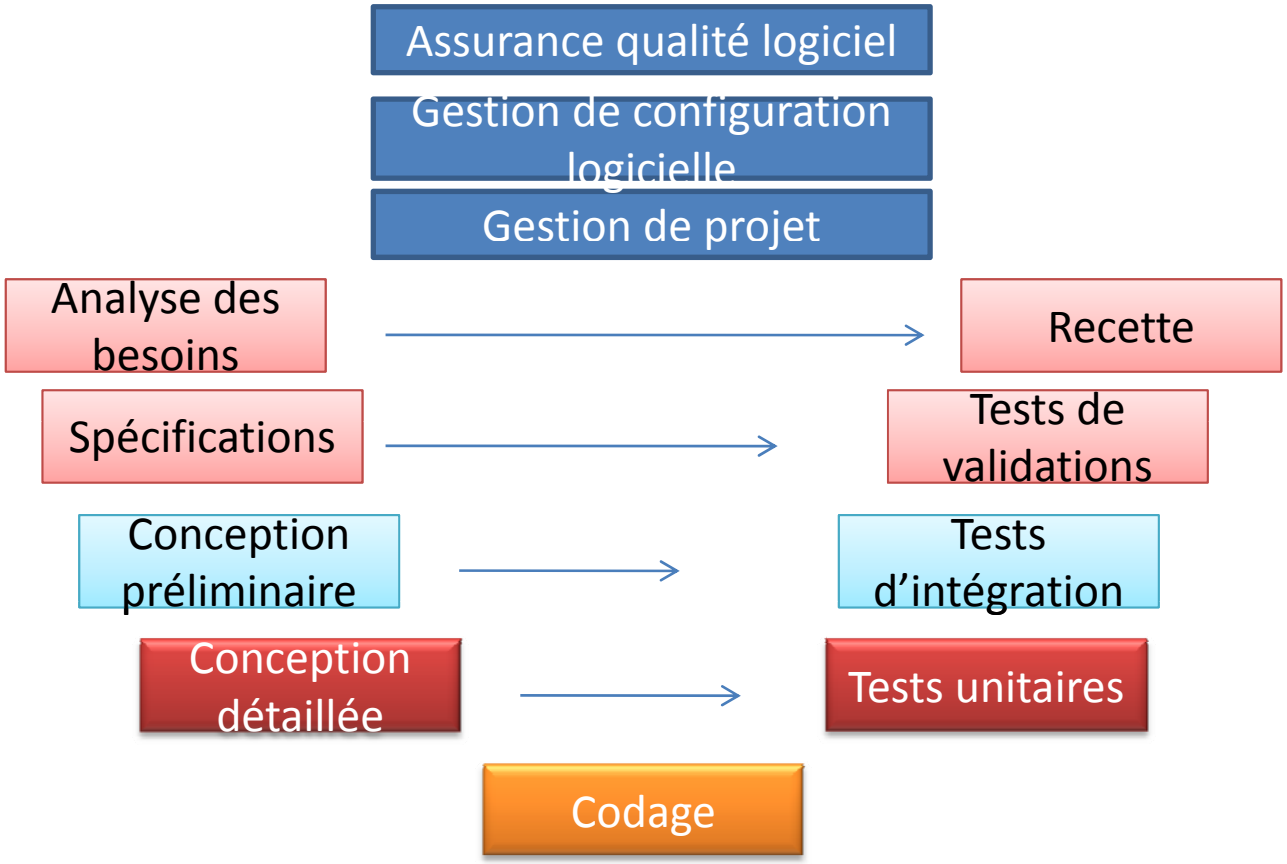
- Définition: Description d'un processus modélisant les étapes de l'élaboration d'un livrable logiciel depuis l'expression des besoins jusqu'à la fin de vie du produit.
- Solutions ALM
  - Borland open ALM
  - Visual studio Team System
  - Eclipse Process Framework

cycle en cascade



- Le plus ancien (hérité du bâtiment)
- Toute étape doit être achevée avant de passer à la suivante
- Toute modification en amont du cycle a un impact majeur sur les coûts en aval.

Cycle en V



Inconvénients :

- I1) l'utilisateur, ou client intervenant uniquement au niveau 1, doit attendre les tests de validation (appelés aussi tests d'acceptation ou de qualification) pour s'assurer que ses exigences ou besoins ont été pris en compte de manière satisfaisante

Avantages :

- A1) modèle normalisé
- A2) montre l'interaction non seulement entre les phases successives mais aussi entre les phases de même niveau
- A3) le cycle de vie du logiciel est intégré dans le cycle de vie du système

Les méthodes itératives et agiles

- Le manifeste agile en 2001 (mis au point par 17 experts)
  - 4 valeurs fondatrices
    - Individus et interactions plutôt que processus et outils
    - Développement logiciel plutôt que documentation exhaustive
    - Collaboration avec le client plutôt que négociation contractuelle
    - Ouverture au changement plutôt que suivi d'un plan rigide

Les 12 principes de l'agilité

- première priorité: satisfaire le client en livrant tôt et régulièrement des logiciels utiles.
- Le changement est tolérable, même dans les étapes avancées du développement
- Livraisons fréquentes d'applications fonctionnelles (2 semaines à deux mois)
- Artistes et développeurs doivent collaborer quotidiennement au projet.
- Bâtissez le projet autour de personnes motivées. croyez en leur capacité à faire le travail.
- La méthode la plus efficace de transmettre l'information est une conversation en face à face.
- Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet.
- Les processus agiles permettent un rythme de développement soutenable..
- technique et qualité de la conception excellentes améliorent l'agilité
- Simplicité
- Equipes auto-organisées.
- Brainstorming À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens.

Méthodes itératives et incrémentales

- RAD (Rapid Application Development: années 80)
  - Cycle court (120 jours maximum)
  - Composé de cinq phases
    - Initialisation: préparation de l'organisation et communication (6% du projet)
    - Cadrage: analyse et expression des exigences (9% du projet)
    - Design: conception et modélisation (23% du projet)
    - Construction: réalisation et prototypage (50% du projet)
    - Finalisation: recette et déploiement (12% du projet)
  - Les 3 premières phases sont séquentielles, au terme du design l'architecture fonctionnelle et générale ayant été définie, il est possible de réaliser un découpage du projet en terme de fonctionnalités (services), et de niveau au sein de ces fonctionnalités.
  - La phase de construction est itérative, module par module on applique le cycle en v (conception détaillée, développement et recette)

- UP (unified Process):Objectifs: intégrer les préceptes UML
  - Le projet est construit par itérations successives, fonctionnalité par fonctionnalité (prototypage)
- XP: Recueil de meilleurs pratiques
  - Programmation en binôme
  - Adaptée aux petites équipes
  - Conception et test continus

- SCRUM: (le terme désigne mêlée dans le rugby).
  - Scrum ne couvre aucune technique d'ingénierie du logiciel nécessite une méthode complémentaire comme XP.

- DSDM: dynamic software development method.
  - Inspirée de la méthode RAD.

Le processus unifié

- Objectifs
  - Spécifier les différentes phases d'un projet.
  - Définir les rôles de chacun des intervenants
  - Contrôle des coûts, délais et qualité de l’application

Caractéristiques

- **Développement itératif du logiciel** : montrer fréquemment au futur utilisateur, ou au marketing, des versions intermédiaires du logiciel en construction, ceci afin de mieux maîtriser les risques inhérents au développement en les levant au plus tôt.
- A**rchitecture logicielle par assemblage de composants** : privilégier une construction du logiciel par assemblage de composants (sur étagère ou produits par le projet), pour développer plus vite et tester plus finement.
- G**estion des exigences** : distinguer et organiser les exigences de l'utilisateur, ou du marketing, et assurer leur traçabilité jusque dans le code, ceci afin d'être capable de démontrer la complète prise en compte des besoins exprimés.
- G**estion des demandes de changement** : enregistrer chaque demande de changement du projet et des caractéristiques du logiciel afin de maîtriser les changements tout en les acceptant.
- V**érification de la qualité en continu** : faire des validations et des recettes fréquentes de versions intermédiaires du logiciel en construction, ceci afin d'habituer l'utilisateur au logiciel à venir et d'assurer progressivement la bonne prise en compte des besoins.
- M**odélisation graphique des exigences** : utiliser les diagrammes UML par exemple, afin de communiquer mieux et plus rigoureusement entre développeurs et utilisateurs.

Phases de développement

- Une approche incrémentale par une succession d'affinements ou d'itérations
- Modèle de développement orienté activité
- Phases:
  - Inception (appelée aussi phase de lancement ou d'initialisation du projet.):
    - Définit les objectifs du projets, sa faisabilité et sa portée
    - Identification des cas d'utilisation principaux
    -
  - Elaboration
    - Définition et construction de l'architecture de base du système
    - Identification et description de la majorité des besoins de l'utilisateur à l'aide de cas d'utilisation

Phase de développement

- L'architecture doit être exprimée sous vue de chacun des modèles

Remarque

À l'issue de cette phase l'étude commerciale est montée, le chef de projet doit être en mesure d'estimer les ressources nécessaires à l'achèvement du projet.

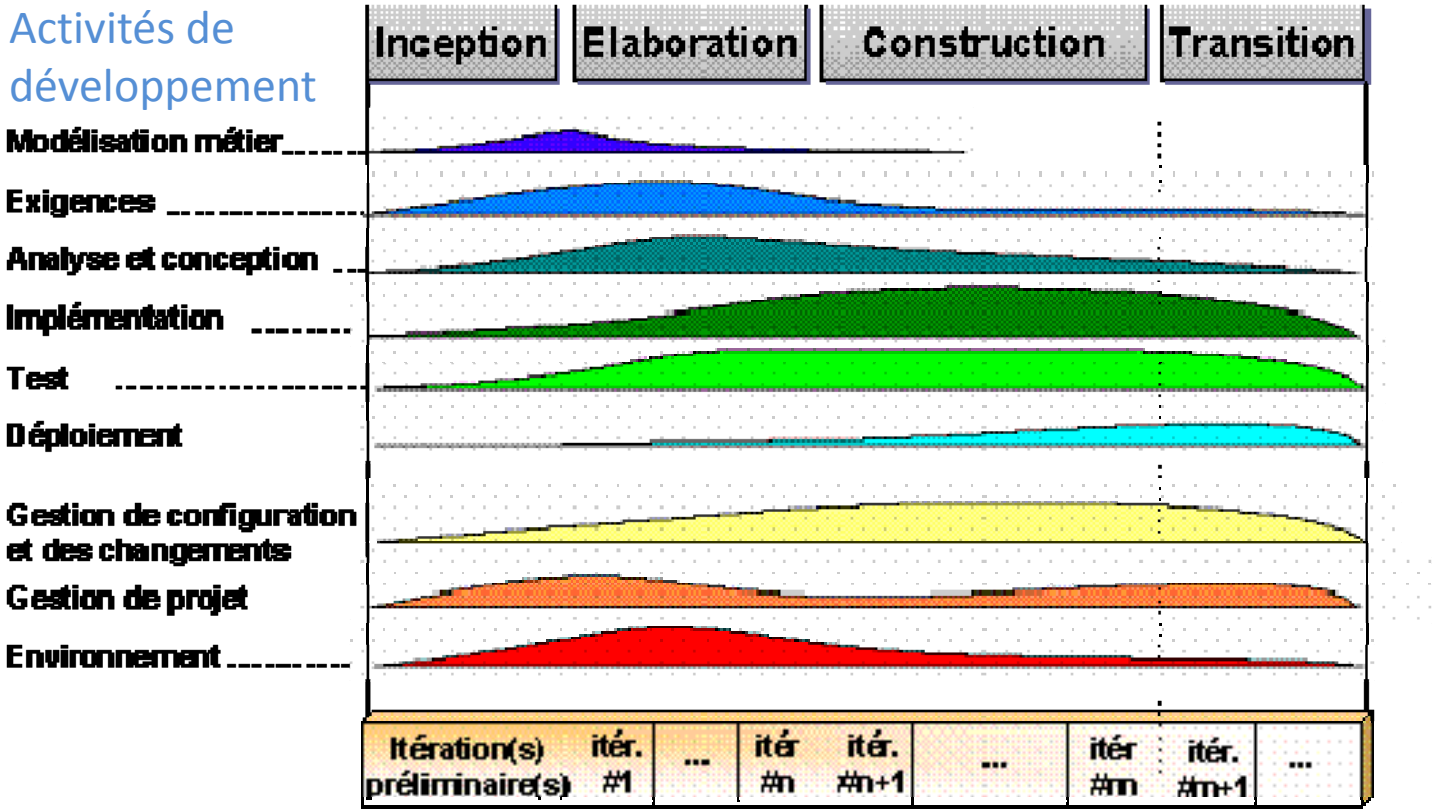
– Construction
  - Conception et implémentation de l'ensemble des cas d'utilisation de la phase d'élaboration
– Transition
  - Passage de l'application des développeurs aux utilisateurs finaux
    – Tests beta
    – Formation des utilisateurs
    – déploiement

Phase de développement

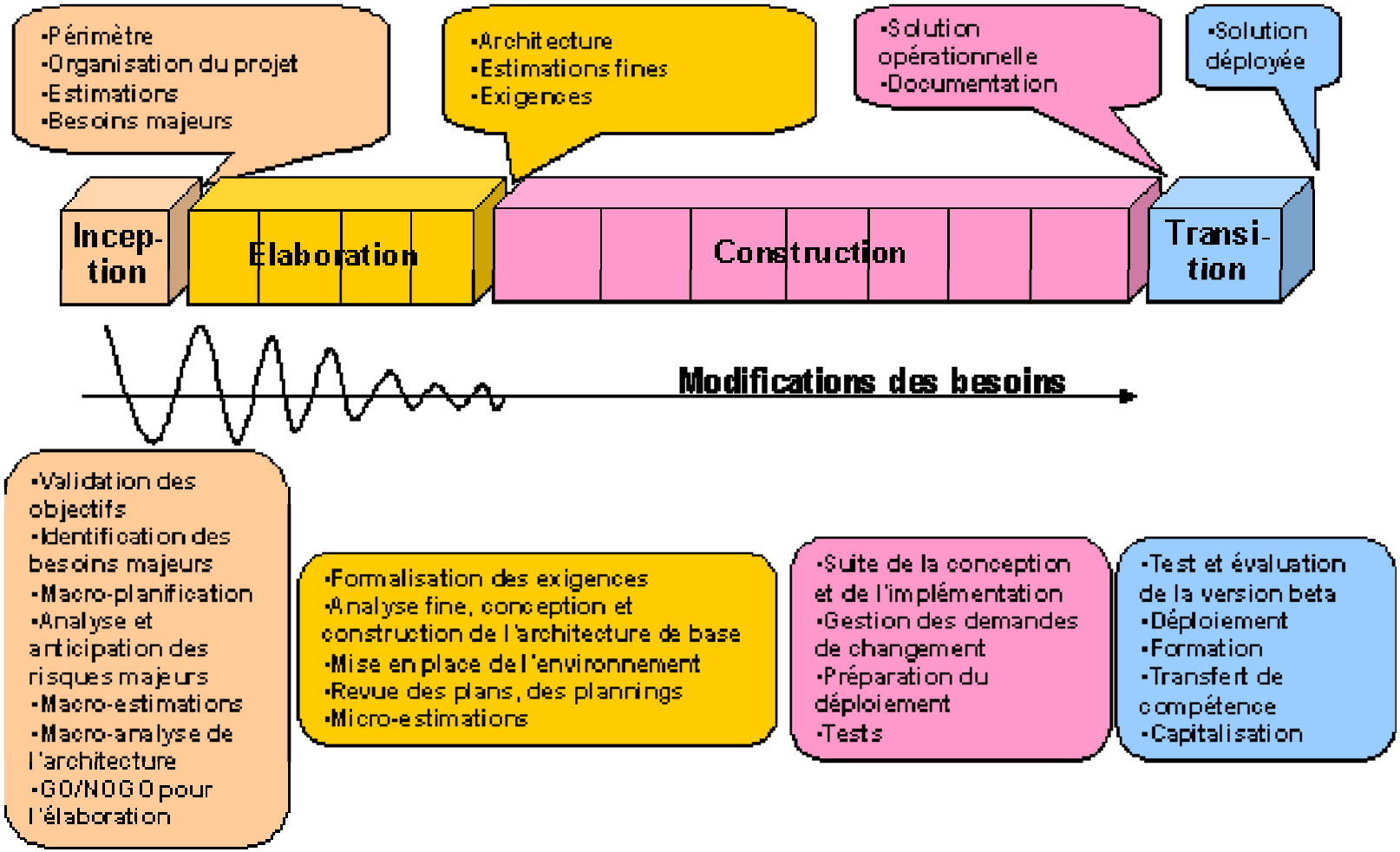
Remarques

- Chaque phase est composée séquentiellement en itérations limitées dans le temps (entre 2 et 4 semaines) le résultat de chacune d'elles est un système testé, intégré et exécutable.
- À la fin de chaque itération, les différents modèles définis précédemment sont affinés et améliorés par des ajouts successifs
- Le système croît avec le temps d'une manière incrémentale, itération par itération

Phases de développement



Activités de développement



Activités de développement

- capture des exigences
- analyse et conception
- Implémentation
- test
- déploiement.
- La modélisation métier est une activité amont optionnelle et transverse aux projets.
- Activités de support
  - gestion de projet
  - gestion du changement et de la configuration
  - ainsi que la mise à disposition d'un environnement complet de développement incluant aussi bien des outils informatiques que des documents et des guides méthodologiques.

Activités de développement

- Le Processus Unifié ne considère pas que les activités sont purement séquentielles. En fait, une itération comporte une certaine quantité de travail dans la plupart des activités. Mais la répartition de l'effort relatif entre celles-ci change avec le temps. Les premières itérations ont tendance à mettre plus l'accent sur les exigences et la conception, les autres moins, à mesure que les besoins et l'architecture se stabilisent grâce au processus de feed-back et d'adaptation. Le processus unifié doit donc être compris comme une trame commune des meilleures pratiques de développement, et non comme l'ultime tentative d'élaborer un processus universel.

Modèle de processus: itératif et incrémental

- Dans le processus unifié, on ne développe pas tout le projet en une fois, on peut découper le travail en plusieurs parties qui sont autant de mini projets. Chacun d'entre eux représentant une itération qui donne lieu à un incrément (une itération désigne la succession des étapes de l'enchaînement d'activités, tandis qu'un incrément correspond à une avancée dans les différents stades de développement, ce dernier constitue souvent un additif à ce qui a été fait précédemment). Chacune de ces itérations possède les caractéristiques d'un projet de développement logiciel : planification, analyse des besoins, conception, implémentation, tests, livraison… Cependant, une itération n'est pas une entité indépendante mais réellement une étape dans le projet.

Modèle de processus: itératif et incrémental

- La stratégie de ce processus est donc de développer un produit logiciel en plusieurs petites étapes : Planifier un peu ; Spécifier, Concevoir et Implémenter un peu ; Intégrer, Tester et Exécuter chaque itération un peu.
- Si le passage d'une étape vous convient, alors vous pouvez passer à la suivante.
- Entre chaque étape vous recevez des retours utilisateurs qui vous permettent de réajuster le développement pour l'étape ultérieure. Ensuite vous passez à l'étape suivante puis à l'étape suivante, etc.
- Lorsque vous avez exécuté toutes les étapes planifiées, vous pouvez livrer le produit aux utilisateurs. Les itérations exécutées dans le premières phases du projet correspondent surtout à l'étude générale du projet, au calcul des risques et à l'élaboration de l'architecture générale. Puis, au fur et à mesure de la création du projet et de la création des composants, les itérations deviennent des incréments.

Avantages

- Prise en compte des changements d'exigence
- Accélère le rythme de développement grâce à des objectifs clairs et à court terme
- Les éléments sont intégrés progressivement
- Le processus de développement peut être amélioré et raffiné en cours de réalisation, l'évaluation à la fin d'une itération permet d'identifier ce qui devrait être changé dans l'organisation et dans le processus pour l'améliorer dans les itérations suivantes

Remarques

- Le nombre, la durée et les objectifs des itérations doivent être soigneusement étudiés
- Les tâches et les responsabilités des participants doivent être bien définies
- Le choix de ce qui doit être implémenté au cours d'une itération repose sur deux facteurs :
  - une itération prend en compte un certain nombre de cas d'utilisation qui ensemble, améliorent l'utilisation du produit à un certain stade de développement
  - l'itération traite en priorité les risques majeurs.

Remarques

- A chaque itération, les développeurs identifient et spécifient les cas d'utilisations pertinents, créent une conception en se laissant guider par l'architecture choisie, implémentent cette conception sous forme de composants et vérifie que ceux ci sont conformes aux cas d'utilisation.
- Dés qu'une itération répond aux objectifs fixés le développement passe à l'itération suivante. Pour rentabiliser le développement il faut sélectionner les itérations nécessaires pour atteindre les objectifs du projet. Ces itérations devront se succéder dans un ordre logique. Un projet réussi suivra un déroulement direct, établi dés le début par les développeurs et dont ils ne s'éloigneront que de façon très marginale. L'élimination des problèmes imprévus fait partie des objectifs de réduction des risques.

Architecture du système

- Dès le démarrage du processus, on aura une vue sur l'architecture à mettre en place.
- L'architecture d'un système logiciel peut être décrite comme les différentes vues du système qui doit être construit.
- L'architecture logicielle équivaut aux aspects statiques et dynamiques les plus significatifs du système.
- L'architecture émerge des besoins de l'entreprise, tels qu'ils sont exprimés par les utilisateurs et autres intervenants et tels qu'ils sont reflétés par les cas d'utilisation.

Architecture du système

- Elle subit également l'influence d'autres facteurs :
  - la plate-forme sur laquelle devra s'exécuter le système.
  - les briques de bases réutilisables disponibles pour le développement.
  - les considérations de déploiement.
  - les systèmes existants
  - les besoins non fonctionnels (performance, fiabilité..).

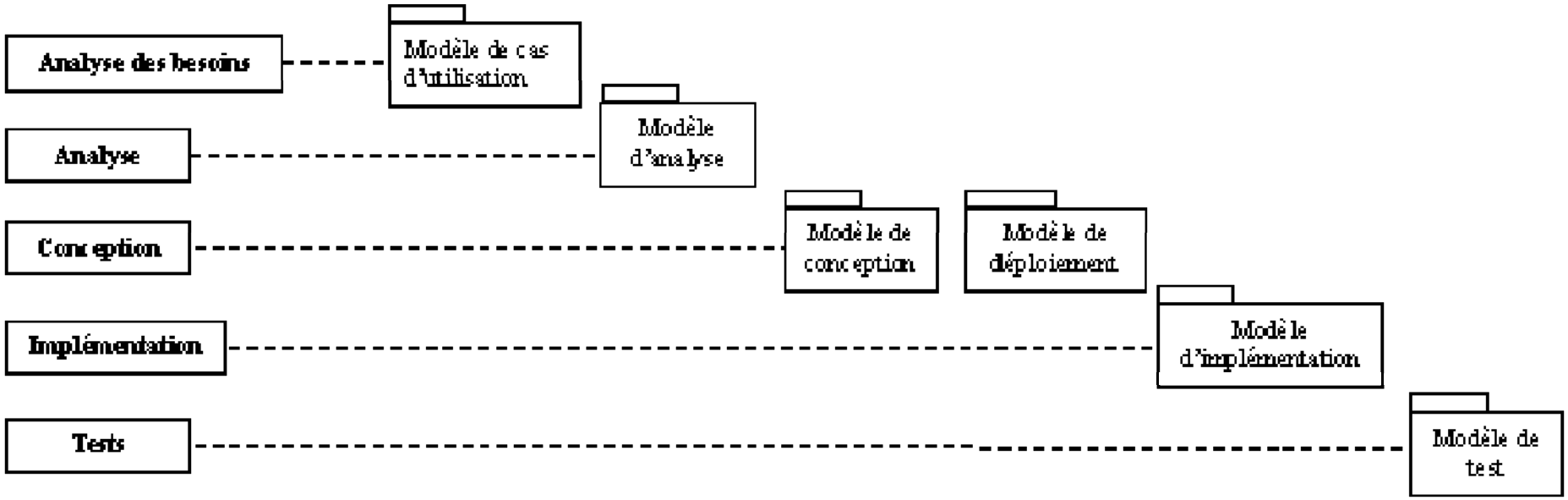
Architecture du système

- Le processus unifié répète un certain nombre de fois une série de cycles. Tout cycle se conclut par la livraison d'une version du produit aux clients et s'articule en 4 phases : création, élaboration, construction et transition, chacune d'entre elles se subdivisant à son tour en itérations.
- Chaque cycle se traduit par une nouvelle version du système. Ce produit se compose d'un corps de code source réparti sur plusieurs composants pouvant être compilés et exécutés et s'accompagne de manuels et de produits associés. Pour mener efficacement le cycle, les développeurs ont besoin de construire toutes les représentations du produit logiciel, c'est à dire l'ensemble des modèles qui décrivent le système.

Activités et Modèles associés

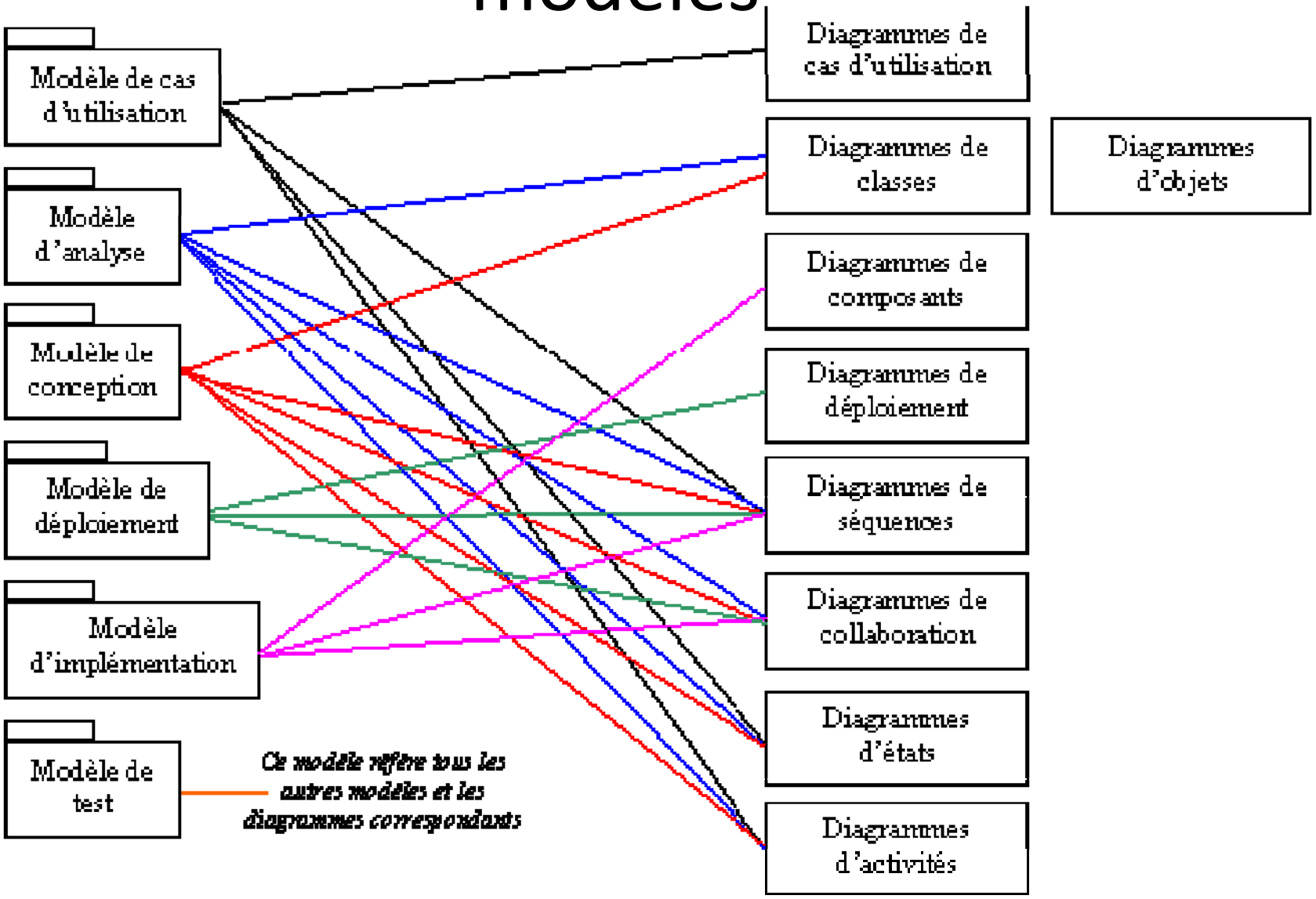
- Généralement, l'artefact principal d'une activité est la mise au point de son modèle. L'une des entrées d'une activité est le modèle issu d'une activité précédente. Lors de chaque phase, plusieurs modèles pourront être créés/enrichis selon l'implication de l'activité dans la phase.
- B:

Activités et Modèles associés



- **Modèle des cas d'utilisation :** Expose les cas d'utilisation et leurs relations avec les utilisateurs.
- **Modèle d'analyse :** Détaille les cas d'utilisation et procède à une première répartition du comportement du système entre divers objets.
- **Modèle de conception :** Définit la structure statique du système sous forme de sous système, classes et interfaces ; Définit les cas d'utilisation réalisés sous forme de collaborations entre les sous systèmes les classes et les interfaces
- **Modèle d'implémentation :** Intègre les composants (code source) et la correspondance entre les classes et les composants
- **Modèle de déploiement :** Définit les nœuds physiques des ordinateurs et l'affectation de ces composants sur ces nœuds.
- **Modèle de test :** Décrit les cas de test vérifiant les cas d'utilisation.
- **Représentation de l'architecture :** Description de l'architecture.

Diagrammes de représentation des modèles



Diagrammes de représentation des modèles

- Tous ces modèles sont liés. Ensemble, ils représentent le système comme un tout. Ils permettent de visualiser, spécifier, construire et documenter l'architecture du système. Les éléments de chacun des modèles présentent des dépendances de traçabilité ; ce qui facilite la compréhension et les modifications ultérieures.

Gestion des exigences –cas d'utilisation

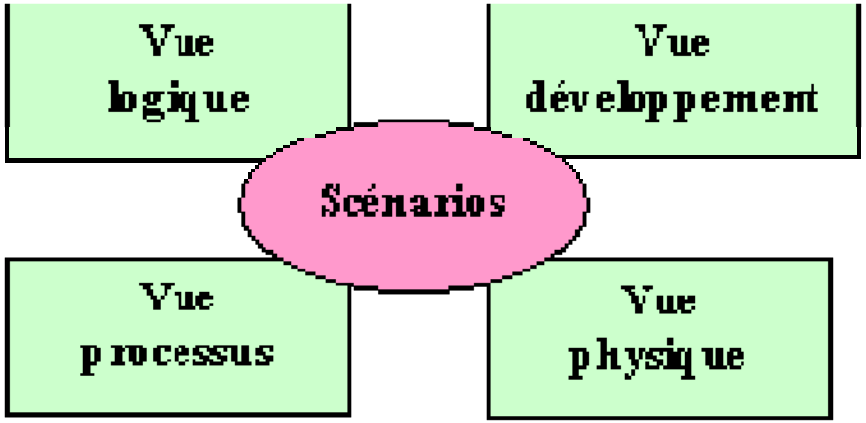
- Une exigence est une condition à laquelle le système doit satisfaire ou une capacité qu'il doit avoir.
  - On distingue les exigences fonctionnelles (qui formulent ce que le système est chargé de faire)
  - et les exigences non fonctionnelles (qui décrivent la qualité des services attendus du système, comme la performance, la sécurité…).
- Les cas d'utilisation font apparaître les besoins fonctionnels des utilisateurs et leur ensemble constitue le modèle des cas d'utilisation qui décrit les fonctionnalités complètes du système.

Gestion des exigences –cas d'utilisation

- Cependant, les cas d'utilisation ne sont pas un simple outil de spécification des besoins du système. Ils vont complètement guider le processus de développement à travers l'utilisation de modèles basés sur l'utilisation du langage UML.
- A partir du modèle des cas d'utilisation, les développeurs créent une série de modèles de conception et d'implémentation réalisant les cas d'utilisation.
- Chacun des modèles successifs est ensuite révisé pour en contrôler la conformité par rapport au modèle des cas d'utilisation.
- Enfin, les testeurs testent l'implémentation pour s'assurer que les composants du modèle d'implémentation mettent correctement en œuvre les cas d'utilisation.

Gestion des exigences –cas d'utilisation

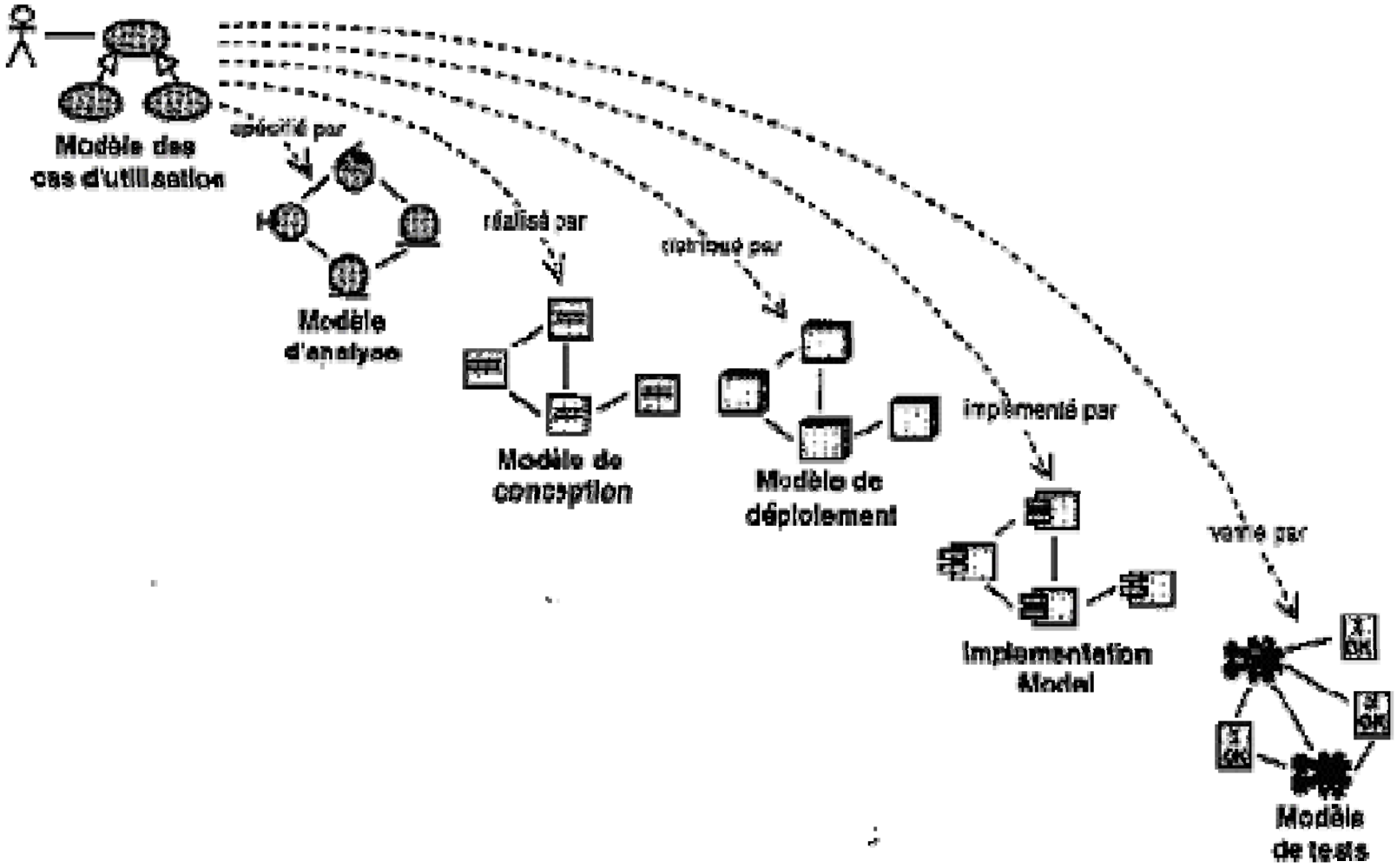
Toutes les itérations effectuées doivent être le résultat de l'analyse d'un ou de plusieurs cas d'utilisation. Selon [Kruchten 95], quelle que soit la vue que l'on utilise pour étudier un logiciel (logique, physique, développement ou processus), les scénarii et cas d'utilisation y ont une influence décisive.



Gestion des exigences –cas d'utilisation

- Les cas d'utilisation garantissent la cohérence du processus de développement du système (figure suivante).
- Ils permettent de guider les activités de développement par la création et la validation de l'architecture du système ; par la définition des cas et des procédures de tests ; par la planification des itérations ; par la création de la documentation utilisateur et par le déploiement du système.
- Ils permettent également de synchroniser le contenu des différents modèles. S'il est vrai que les cas d'utilisation guident le processus de développement, ils ne sont pas sélectionnés de façon isolée, mais doivent absolument être développés "en tandem" avec l'architecture du système.

Gestion des exigences –cas d'utilisation



Gestion des exigences –cas d'utilisation

- Le projet est mené en tenant compte des besoins et des exigences des utilisateurs. Les cas d'utilisation du futur système sont identifiés, décrits avec précision et on leur affecte un ordre de priorité.
- La gestion des exigences ne signifie pas forcement avoir des exigences correctes dès le démarrage du projet mais simplement ne pas être négligent, recueillir efficacement les exigences, les organiser et considérer les changements de manières formelles.

Analyse des risques et évaluation continue de la qualité

- Un risque est un événement redouté dont l'occurrence est plus ou moins prévisible et provoquant, lorsqu'il se produit, des dommages sur le projet. Le processus unifié permet de construire le logiciel en analysant les risques dès les premières étapes du développement. Il faut non seulement penser aux risques techniques, mais également à ceux liés au client, liés au domaine applicatif et à l'organisation du projet.
- Le pilotage par les risques, c'est :
- analyser les risques potentiels le plus tôt possible (dès les premières itérations),
- les hiérarchiser,
- commencer par travailler sur les éléments les plus exposés.
- La gestion des risques est une démarche proactive, alors que la gestion des problèmes est une démarche réactive.
- Le processus unifié permet également d'effectuer une gestion continue de la qualité. En effet, il n'est vraiment pas recommandé de faire tous les tests et analyse de la qualité à la fin du développement. Ce travail doit se faire en continu pour éviter les surcoûts et les retards de livraison.

Analyse des risques et évaluation continue de la qualité

- **Quelques chiffres :**
  - Corriger une anomalie plus tard coûte 10 à 100 fois plus que de la corriger à son origine [Boehm 81].
  - Les produits avec le moins d'anomalies ont les délais les plus courts [Jones 91].
  - La mauvaise qualité est la raison la plus courante de dépassement des délais [Jones 94].
  - La correction des anomalies consomme 40-50% du coût total [Boehm 87].
  - 60% des anomalies existent au moment de la conception [Gilb 88].
  - Le processus unifié permet d'effectuer des vérification de l'adéquation de la solution aux besoins, de faire des tests systématiques et périodiques ainsi que des actions qualité. Cela permet d'identifier de manière précoce les possibles disfonctionnements, d'avoir une meilleure réactivité par rapport aux déviations constatées et de maîtriser les risques de dérapage.