REINVENT ENGINEERING

CENTRALE NANTES

Qualité Conception Modélisation

Introduction à la modélisation et aux phases de développement d'un projet

PLAN

- Des méthodes orientées objet
- De l'unification des méthodes à UML
- Le formalisme d'UML
  - Les vues
  - Les diagrammes
  - Les modèles d'éléments
- Les phases de développement d'un projet
- Diagrammes UML et phases de développement

DES MÉTHODES ORIENTÉES OBJET

Les méthodes orientées objet ont été influencées par le développement d'Ada et des langages de programmation basés sur les objets C++

Elles abordent l'étude d'un problèmes suivant :

- L'aspect statique : identifie les propriétés des objets et leurs liaisons avec les autres objets
- L'aspect dynamique : définit le cycle de vie des objets (comportement des objets, les différents états par lesquels ils passent, les évènements déclenchant ces changements d'états)
- L'aspect fonctionnel : précise les fonctions réalisées par les objets par l'intermédiaire des méthodes)

DES MÉTHODES ORIENTÉES OBJET [2]

Les méthodes orientées objet les plus connues :

- OMT (Object Modeling Technique - Rumbaugh et al. 1991)
- BOOCH (Booch 1991)
- Objectory-OOSE (Object Oriented Software Engineering –Jacobson et al. 1992)
- OOA (Object Oriented Analysis - Shlaer et Mellor 1992)
- OOA (Object Oriented Analysis - Coad et Yourdon 1992)
- …

DE L'UNIFICATION DES MÉTHODES À UML [1]

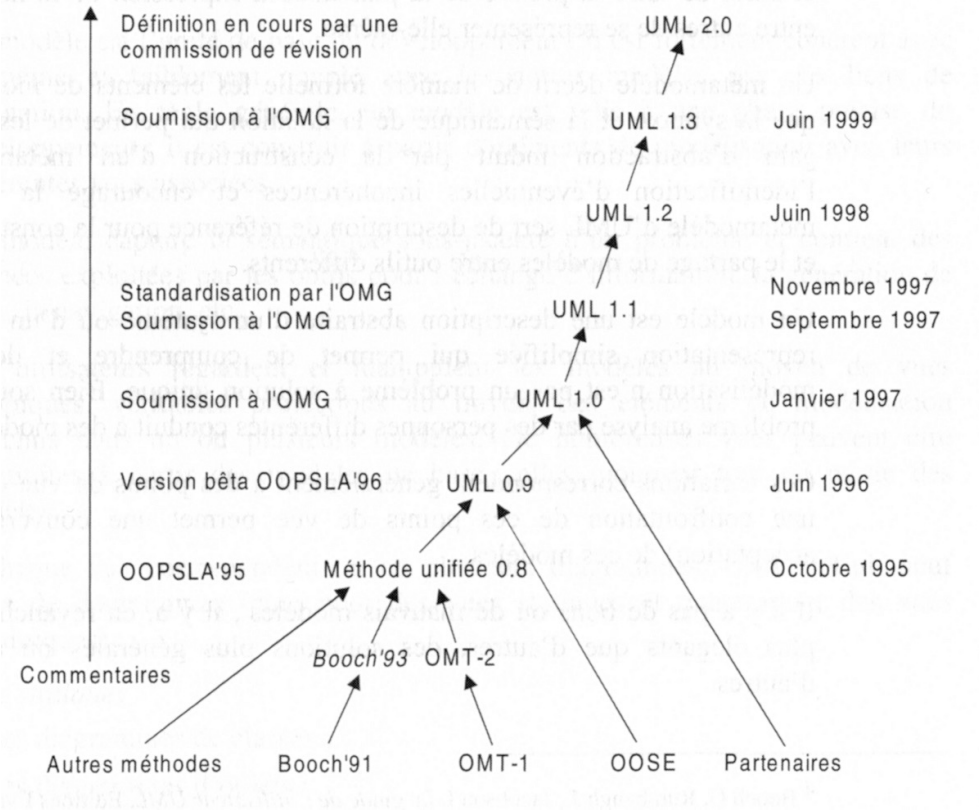
En 1994, Grady Booch (Booch) et James Rumbaugh (OMT) unifient leur travaux pour proposer la méthode unifiée (unified method)

En 1995, Ivar Jacobson (OOSE, use cases) les rejoint

Ils se fixent quatre objectifs :

- Représenter des systèmes entiers par des concepts objets
- Établir un couplage explicite entre les concepts et les artéfacts exécutables
- Prendre en compte les facteurs d'échelle inhérents aux systèmes complexes et critiques
- Créer un langage de modélisation utilisable à la fois par les humains et les machines

DE L'UNIFICATION DES MÉTHODES À UML [1]



LE FORMALISME D'UML

UML est un langage de modélisation visuel ayant un ensemble de notations

UML n'est pas une méthode d'analyse ou de conception

Le formalisme UML est composé de 13 types de diagrammes (9 en UML 1.3)

UML se décompose en plusieurs sous-ensembles :

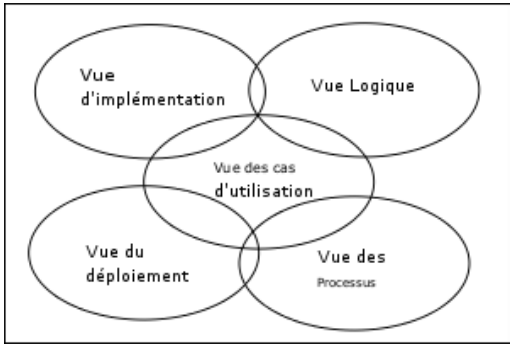
- Les vues : Les vues sont les observables du système.
- Les diagrammes : Les diagrammes sont des éléments graphiques.
- Les modèles d'éléments : Les modèles d'éléments sont les briques des diagrammes UML, ces modèles sont utilisés dans plusieurs types de diagramme.

LE FORMALISME D'UML - LES VUES

Vue des cas d'utilisation : c'est la description du système "vue" par les acteurs du système. Elle correspond aux besoins attendus par chaque acteur (c'est le QUOI et le QUI).

Vue logique : C'est la définition du système vue de l'intérieur. Elle explique comment peuvent être satisfait les besoins des acteurs (c'est le COMMENT).

Vue d'implémentation : Cette vue définit les dépendances entre les modules.
Vue des processus : C'est la vue temporelle et technique, qui met en œuvre les notions de tâches concurrentes, stimuli, contrôle, synchronisation, etc..



Vue de déploiement : Cette vue décrit la position géographique et l'architecture physique de chaque élément du système (c'est le OU).

LE FORMALISME D’ UML - LES DIAGRAMMES

Les 13 diagrammes UML sont dépendants hiérarchiquement et se complètent

Diagrammes Structurels ou Diagrammes statiques (Structure Diagram)
Diagrammes Comportementaux ou Diagrammes dynamiques (Behavior Diagram)
Diagramme d'interactions (Interaction Diagram)

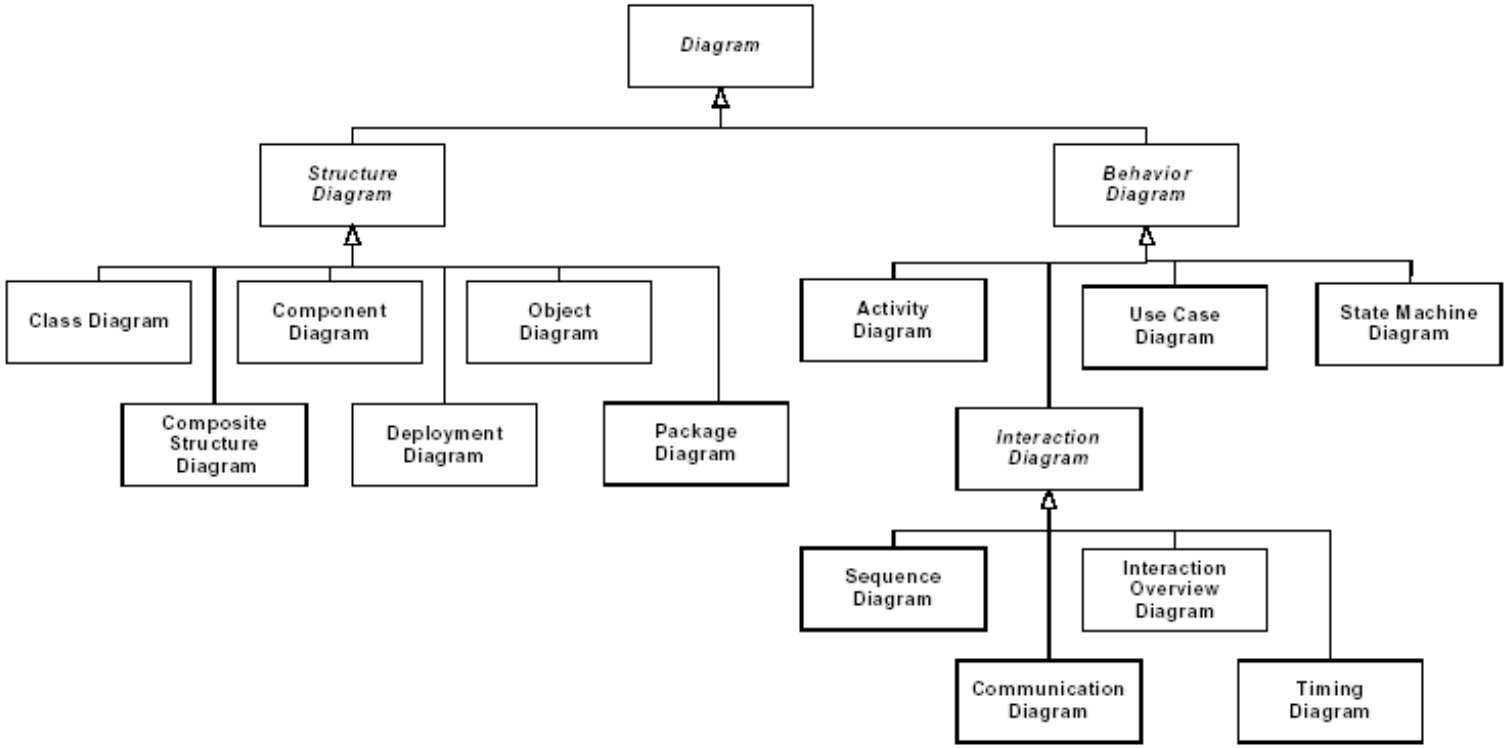


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
Sequence Diagram
Interaction Overview Diagram
Communication Diagram
Timing Diagram

LE FORMALISME D'UML - LES DIAGRAMMES

Diagrammes Structurels ou Diagrammes statiques (Structure Diagram)

- Diagramme de classes (Class diagram) : il représente les classes intervenant dans le système.
- Diagramme d'objets (Object diagram) : il sert à représenter les instances de classes (objets) utilisées dans le système.
- Diagramme de composants (Component diagram) : il permet de montrer les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...)

LE FORMALISME D'UML - LES DIAGRAMMES

Diagrammes Structurels ou Diagrammes statiques (Structure Diagram)

- Diagramme de déploiement (Deployment diagram) : il sert à représenter les éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent avec eux.
- Diagramme des paquetages (Package Diagram) : représente les packages du système
- Diagramme de structures composites (Composite Structure Diagram) : montre la structure interne d'une classe avec ses points d'interaction avec les autres parties du système

LE FORMALISME D'UML - LES DIAGRAMMES

Diagrammes Comportementaux ou Diagrammes dynamiques (Behavior Diagram)

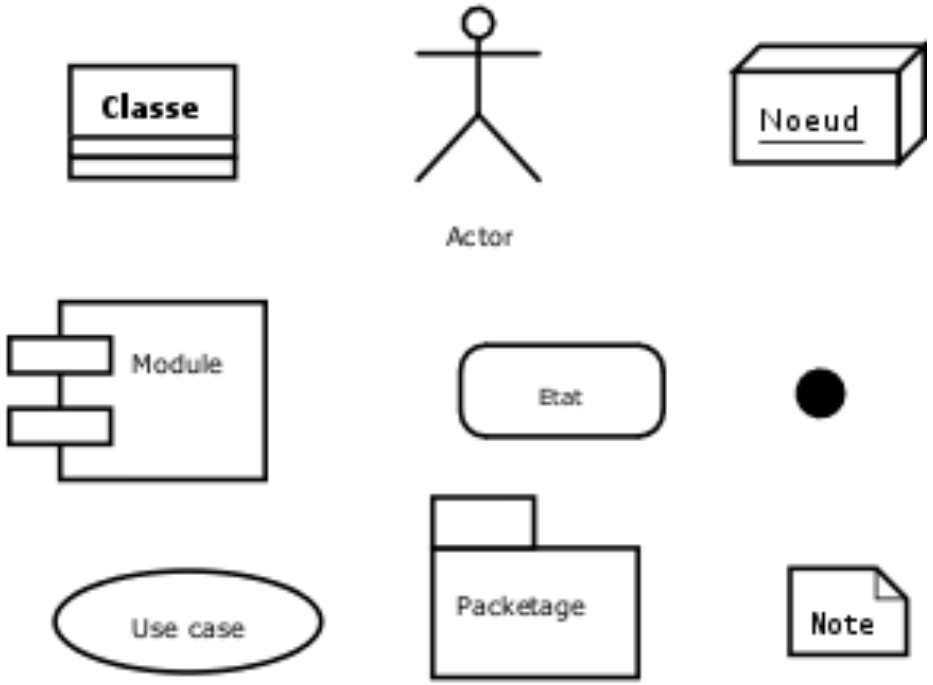
- Diagramme des cas d'utilisation (use-cases) (Use case diagram): il décrit les possibilités d'interaction entre le système et les acteurs, c'est-à-dire toutes les fonctionnalités que doit fournir le système.
- Diagramme États-Transitions (State Machine Diagram) : il montre la manière dont l'état du système (ou de sous-parties) est modifié en fonction des événements du système.
- Diagramme d'activité (Activity Diagram) : variante du diagramme d'états-transitions, il permet de représenter le déclenchement d'événements en fonction des états du système et de modéliser des comportements parallélisables (multi-threads ou multi-processus).

LE FORMALISME D'UML - LES DIAGRAMMES

Diagramme d'interactions (Interaction Diagram) :

- Diagramme de séquences (Sequence Diagram) : représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou des acteurs.
- Diagramme de communication (Communication Diagram) : représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.
- Diagramme global d'interaction (Interaction Overview Diagram) : variante du diagramme d'activité où les nœuds sont des interactions.
- Diagramme de temps (Timing Diagram) : représentation des interactions où l'aspect temporel est mis en valeur.

LE FORMALISME D' UML - LES MODÈLES D' ÉLÉMENTS



LES PHASES DE DÉVELOPPEMENT D'UN PROJET

MODÈLE EN CASCADE (WATERFALL MODEL) [6]

Le cycle de développement en cascade se décompose en phases discrètes, chacune ayant un résultat et un critère de terminaison définis

Le modèle classique du cycle de développement est séquentiel et comprend 5 phases :

- Analyse
- Conception
- Implémentation
- Test
- Installation

LES PHASES DE DÉVELOPPEMENT D'UN PROJET

MODÈLE EN CASCADE (WATERFALL MODEL) [6]

Chaque phase a des entrées et des sorties, qui sont généralement des documents, parfois des produits

Toute sortie d'une phase servira d'entrée à une phase ultérieure

Certaines activités déployées pendant une phase sont spécifiques à la phase, d'autres préparent les phases suivantes

LES PHASES DE DÉVELOPPEMENT D'UN PROJET

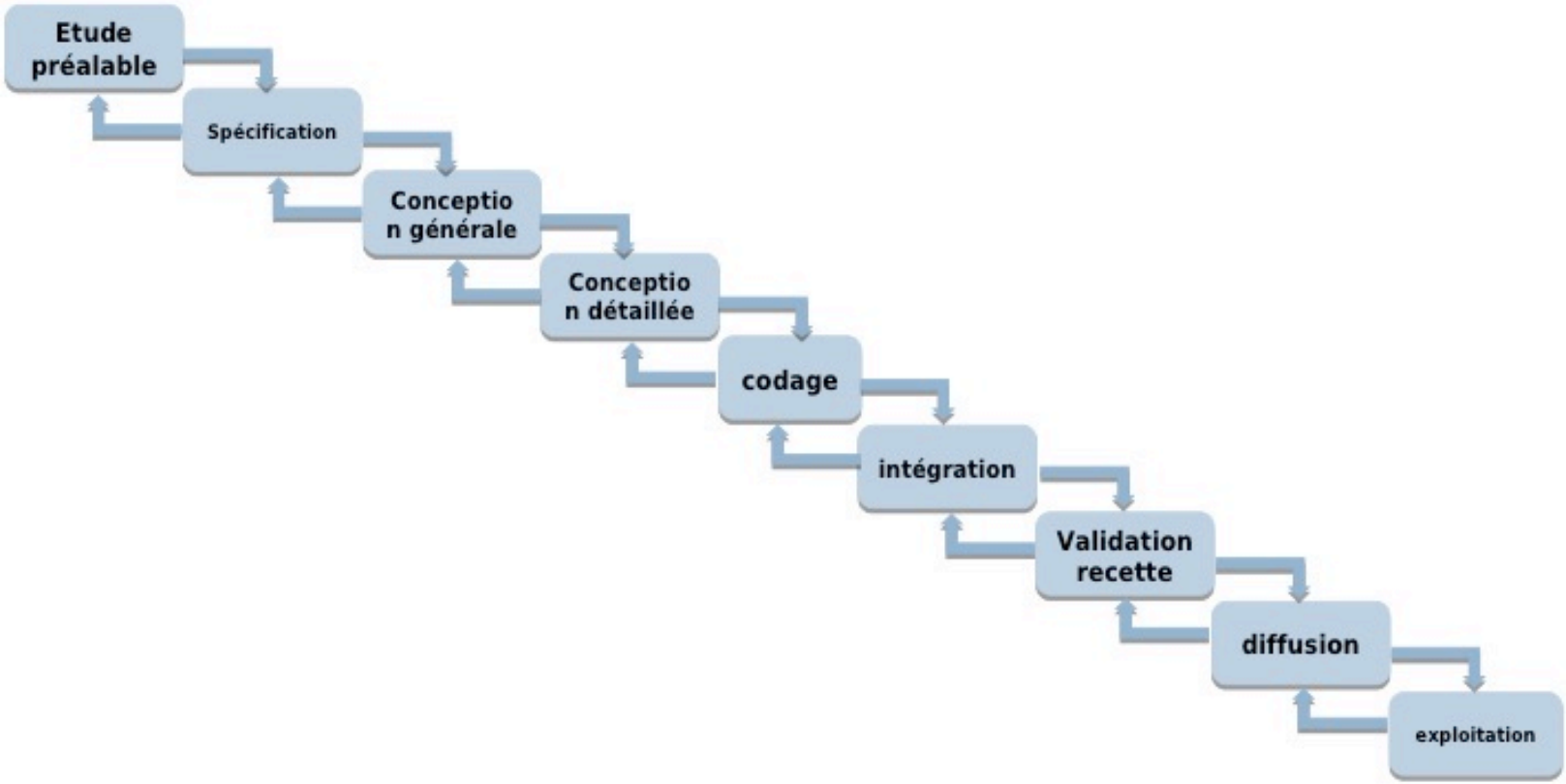
MODÈLE EN CASCADE (WATERFALL MODEL) [6] MODÈLE EN CASCADE (WATERFALL MODEL) [6]

L'application stricte du modèle en phases prescrit qu'une phase soit entièrement complétée avant de passer à la suivante

Dans la pratique, il arrive cependant qu'au cours d'une phase on découvre des erreurs commises dans une phase antérieure ou même l'absence d'éléments essentiels qui auraient dû être fournis par une phase antérieure

Il peut donc être nécessaire de revenir sur une phase précédente : dans ce cas il faut parcourir à nouveau toutes les phases à partir de la phase révisée pour répercuter partout les modifications

MODÈLE EN CASCADE (WATERFALL MODEL) [7]



MODÈLE EN CASCADE (WATERFALL MODEL) [7]

Etude préalable *(feasibility)*

Phase exploratoire : Y-a-t-il lieu de réaliser le logiciel ?

>Fixer les conditions générales

>Débouche sur une phase conceptuelle

> Cahier des charges et plan de projet

Spécification *(requirements)*

Description informelle vers définition précise

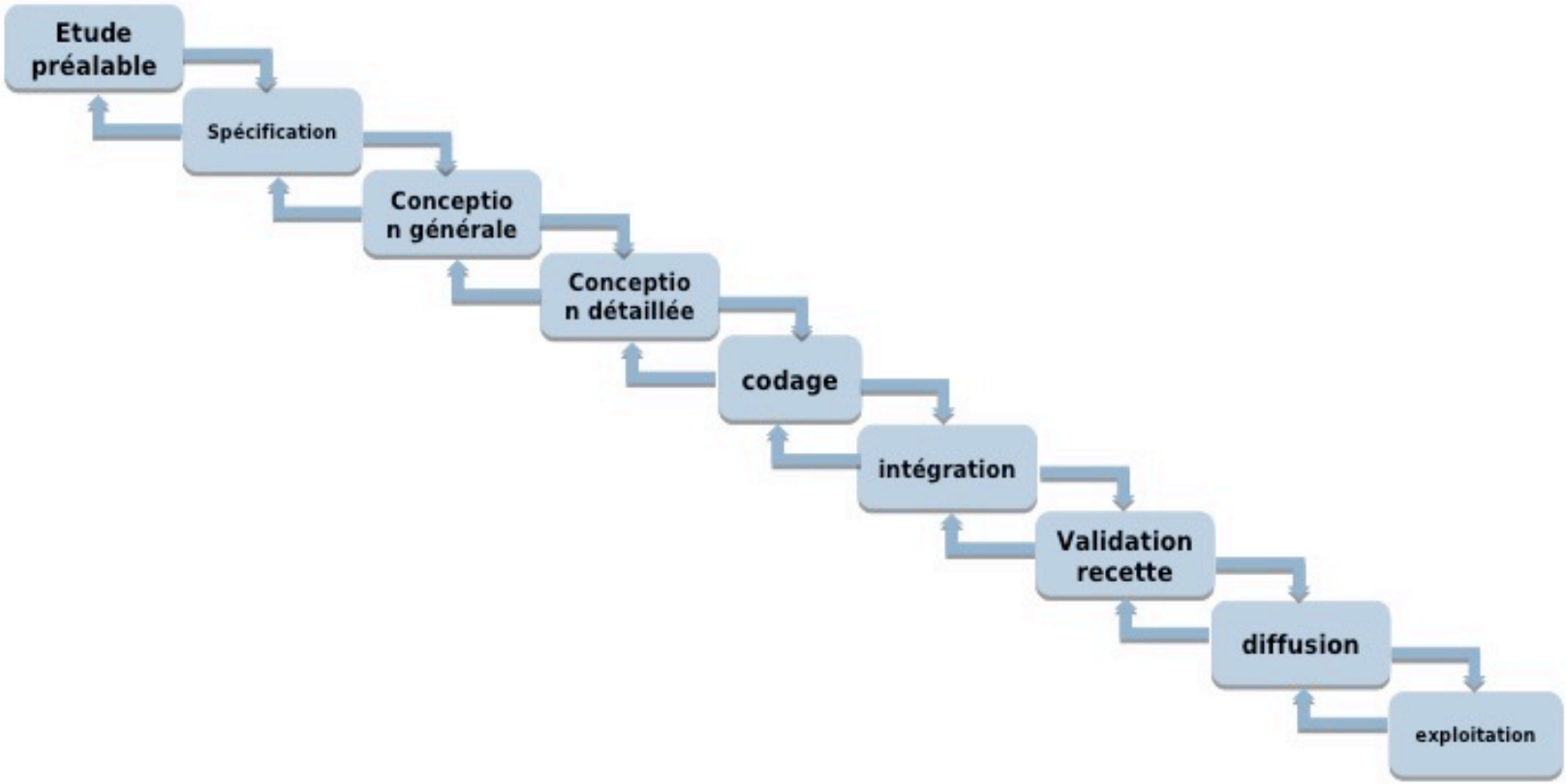
>Des objets manipulés

>Des tâches à effectuer sur ces objets

>Des contraintes de performance

Planification détaillée des étapes suivantes

MODÈLE EN CASCADE (WATERFALL MODEL) [7]



MODÈLE EN CASCADE (WATERFALL MODEL) [7]

Conception générale (product design)

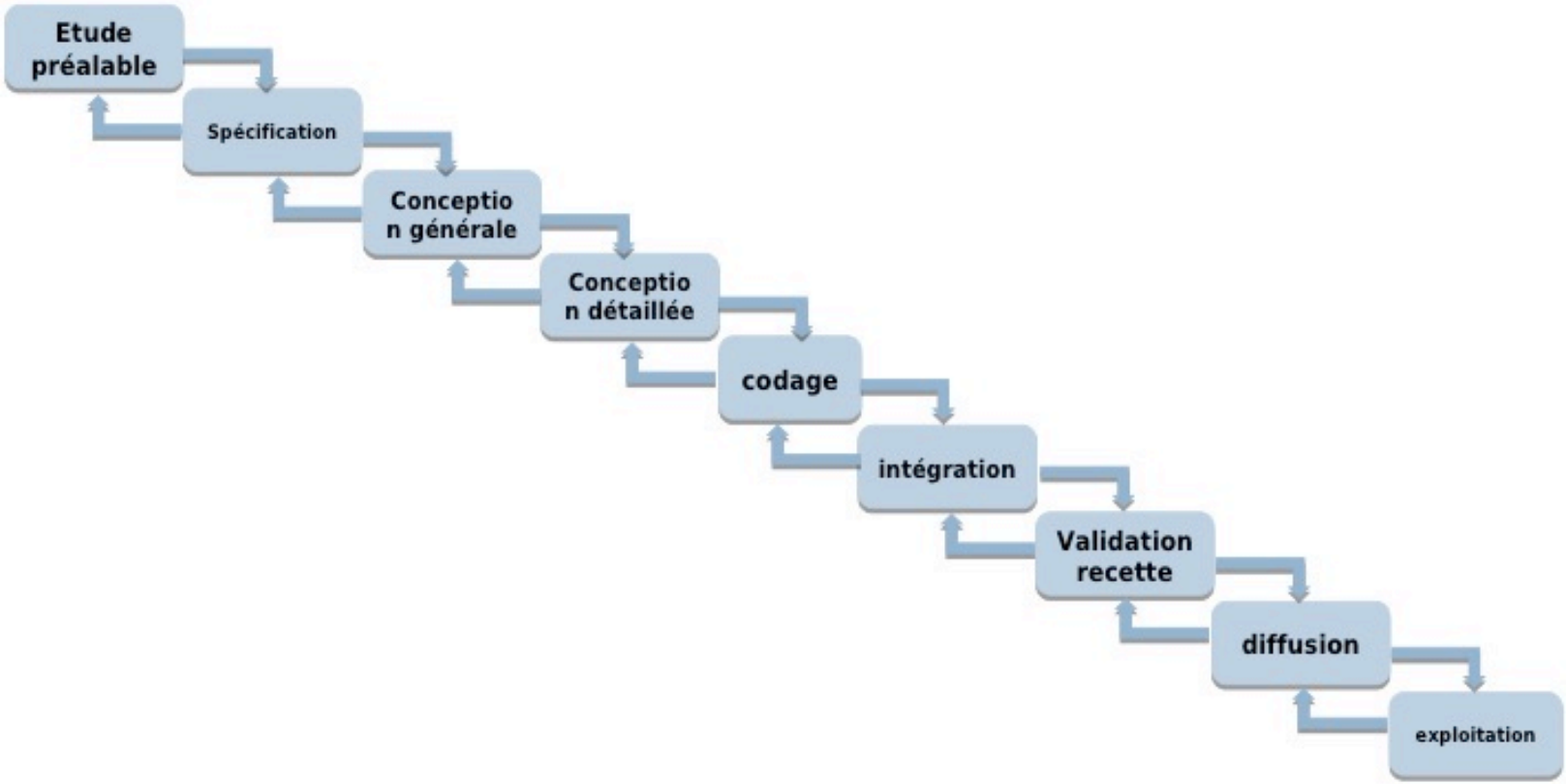
de la définition vers la réalisation

- Architecture du système
  - Principales structures de données
  - Décomposition du système en modules

Conception détaillée (detailed design)

Raffinement des éléments précédents jusqu'à l'obtention d'une forme permettant d'écrire immédiatement les programmes

MODÈLE EN CASCADE (WATERFALL MODEL) [7]



MODÈLE EN CASCADE (WATERFALL MODEL) [7]

Codage (coding)
- écriture des textes des programmes

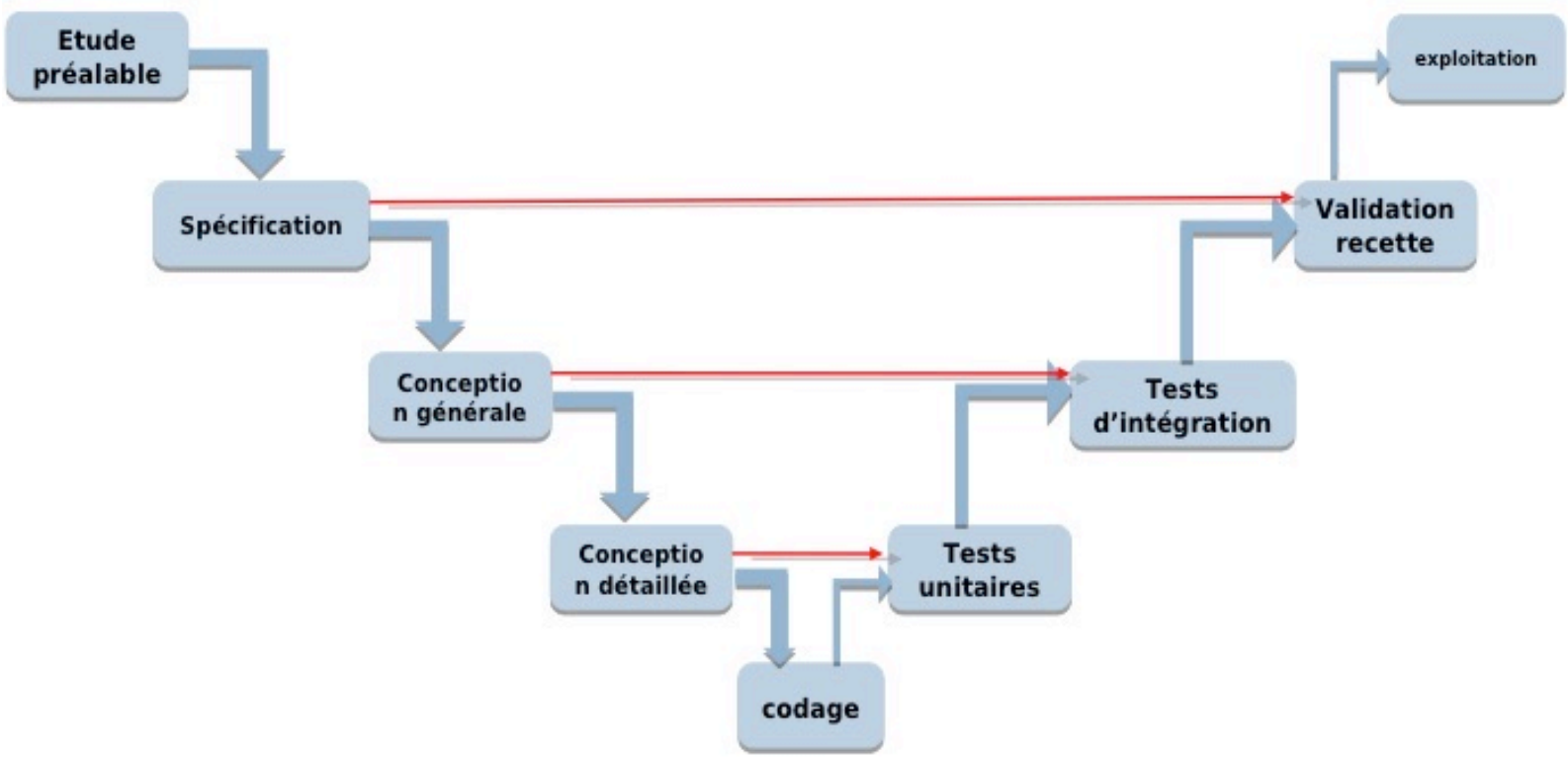
Intégration
- Regroupement des divers modules
- Construction de l'architecture générale

Validation globale/recette

Diffusion
- Préparation et distribution des différentes versions

Exploitation
- Mise en place du système dans son environnement opérationnel

MODÈLE EN V [7]



Variation du modèle en cascade

Attention centrée sur la correction : vérification et validation

MODÈLE EN V

Avantages

Montre comment les activités de tests sont liées à celles d'analyse et de conception

Oblige à une réflexion et à des retours sur la description en cours

Meilleure préparation de la branche droite du V

Inconvénients

Manque de retours : pas de résultats intermédiaires dont on peut discuter avec le client

Avant la phase d'intégration, seuls des documents sont produits

MÉTHODES AGILES [8]

Création aux milieu des 90's par un groupe d'experts en cycles de vie logiciels voulant proposer de nouveaux modèles

Modèles plus « légers » : moins de documentation et moins de contrôle sur le procédé

Pour des projets petits ou moyens et des équipes de taille réduite

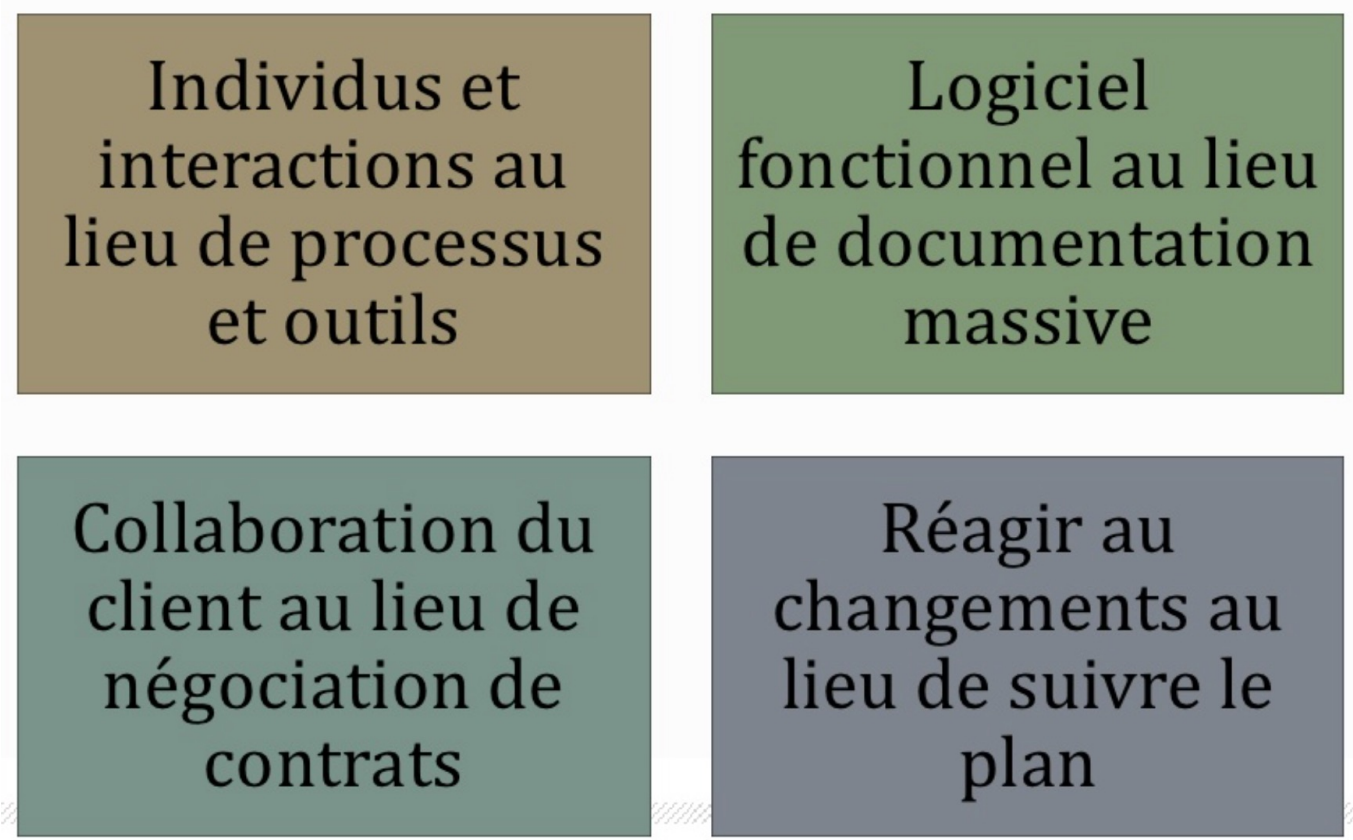
Permettent de s'ajuster rapidement à des changements de spécification tout en assurant des livraisons fréquentes

MÉTHODES AGILES

Valeurs des méthodes Agiles :

- L'équipe
- L'application
- La collaboration
- L'acceptation du changement

MÉTHODES AGILES



Extrait de [8]

MÉTHODES AGILES [extrait de 8]

INDIVIDUS ET INTERACTIONS AU LIEU DE PROCESSUS ET OUTILS

- Les collaborateurs sont la clé du succès
- Les « seniors » échoueront s'ils ne collaborent pas en tant qu'équipe
- Un bon collaborateur n'est pas un forcément un bon programmeur. C'est quelqu'un qui travaille bien en équipe
- Une surabondance d'outils est aussi mauvaise que le manque d'outils
- Démarrer petit et investir peu au démarrage
- Construire l'équipe c'est plus important que construire l'environnement

MÉTHODES AGILES [extrait de 8]

LOGICIEL FONCTIONNEL AU LIEU DE DOCUMENTATION MASSIVE

- Un code sans documentation est un désastre
- Trop de documents est pire que pas de documents
- Difficulté à produire et à synchroniser avec le code
- Souvent les documents sont des « mensonges » formels
- Le code ne ment jamais sur lui-même
- Produire toujours des documents aussi courts que possible

MÉTHODES AGILES [extrait de 8]

COLLABORATION DU CLIENT AU LIEU DE LA NÉGOCIATION DE CONTRATS

- Très difficile de décrire la totalité du logiciel depuis le début
- Les projets réussis impliquent les clients d'une manière fréquente et régulière
- Le client doit avoir un contact direct avec l'équipe de développement

MÉTHODES AGILES [extrait de 8]

RÉAGIR AUX CHANGEMENTS AU LIEU DE SUIVRE UN PLAN

- Un logiciel ne peut pas être planifié très loin dans le futur
- Tout change : technologie, environnement et surtout les besoins
- Les chefs de projets classiques fonctionnent sur la base de GANTT et le système de tâches
- Avec le temps, les diagrammes se dégradent car des tâches s'ajoutent et d'autres deviennent non nécessaires
- Une meilleure stratégie est de planifier très court (02 semaines à 01 mois)
- Plannings détaillés pour la semaine à venir, rigoureux pour les trois mois et très vagues au-delà

MÉTHODES AGILES

12 principes AGILES

1.Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.

2.Accueillez positivement les changements de besoins, même tard dans le projet. Les processus agiles exploitent le changement pour donner un avantage compétitif au client.

3.Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.

4.Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.

5.Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont elles ont besoin et faites-leur confiance pour atteindre les objectifs fixés.

6.La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face.

Source : https://fr.wikipedia.org/wiki/Manifeste_agile#Les_12_principes

MÉTHODES AGILES

12 principes AGILES

7.Un logiciel opérationnel est la principale mesure d'avancement.

8.Les processus agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.

9.Une attention continue conduit à l'excellence technique et à une bonne conception renforce l'agilité.

10.La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle.

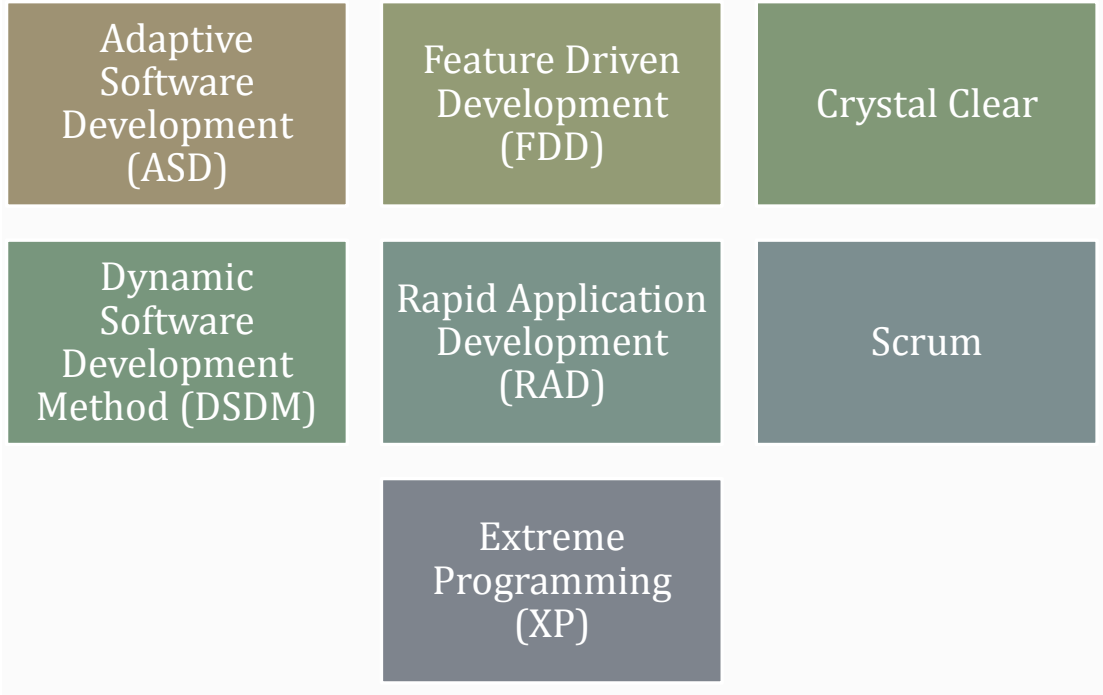
11.Les meilleures architectures, spécifications et conceptions émergent d'équipes auto-organisées.

12.À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

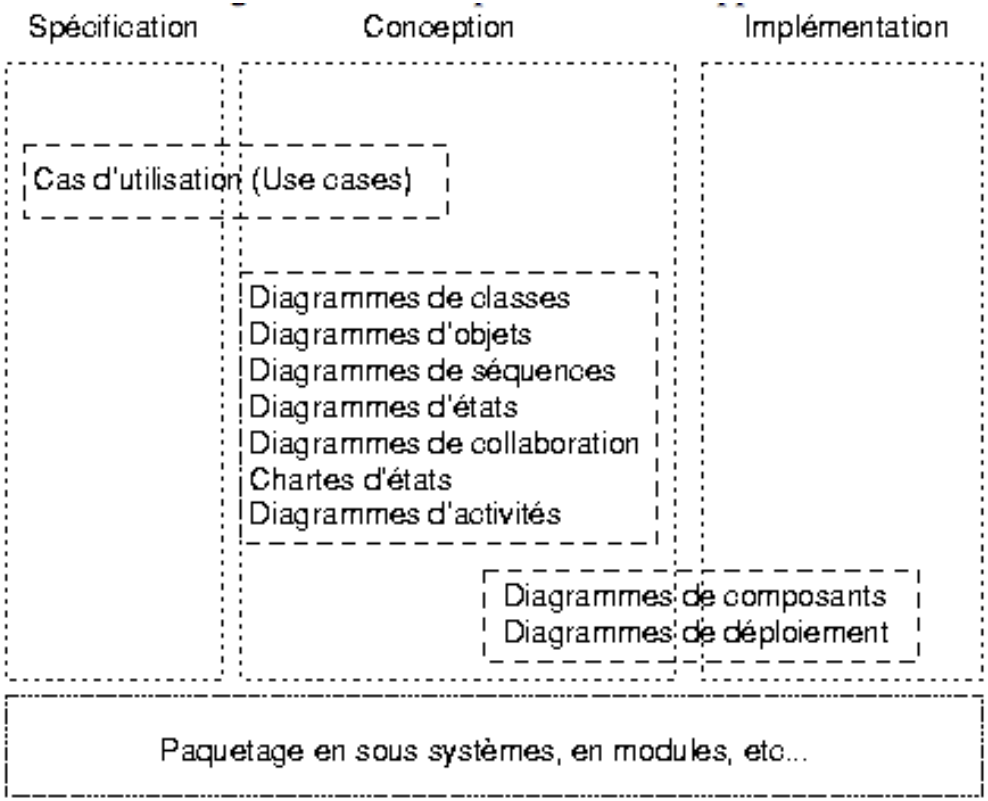
Source : https://fr.wikipedia.org/wiki/Manifeste_agile#Les_12_principes

MÉTHODES AGILES

Principales méthodes Agile [extrait de 8]



DIAGRAMMES UML ET PHASES DE DÉVELOPPEMENT

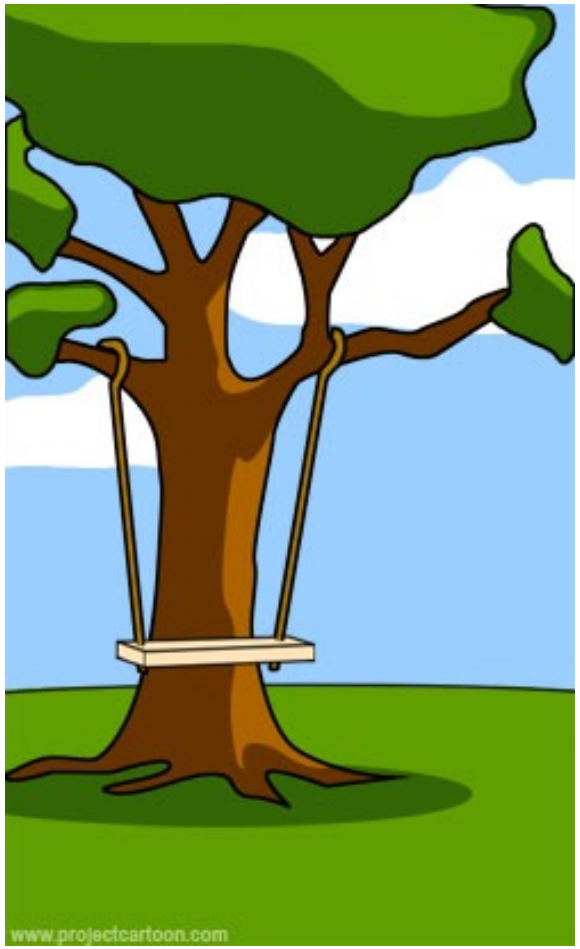


How IT Projects Really Work

Ce que le client
a expliqué

Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work



Ce qu'à compris
le chef de projet

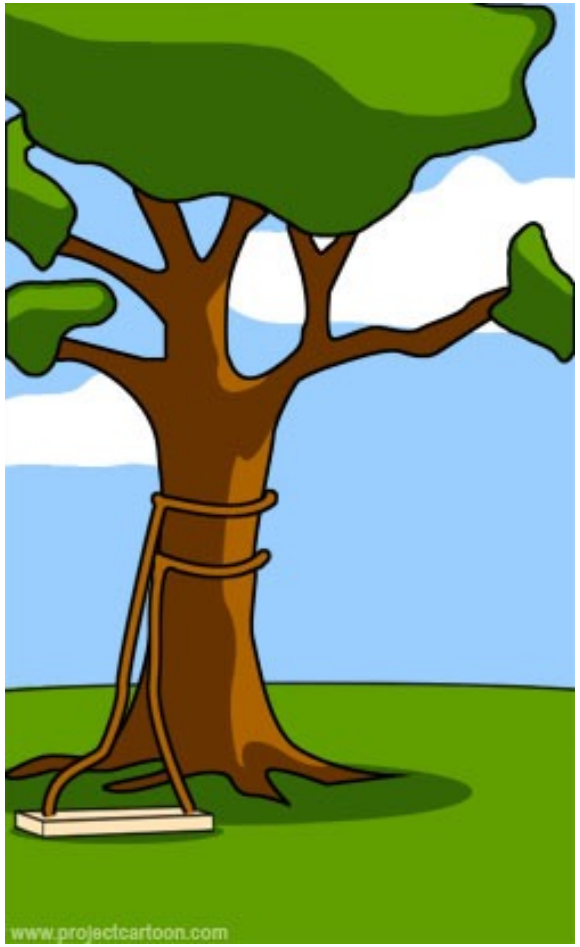
Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work

Ce que l'analyse à conçu

Source : http://www.projectcartoon.com/cartoon/42

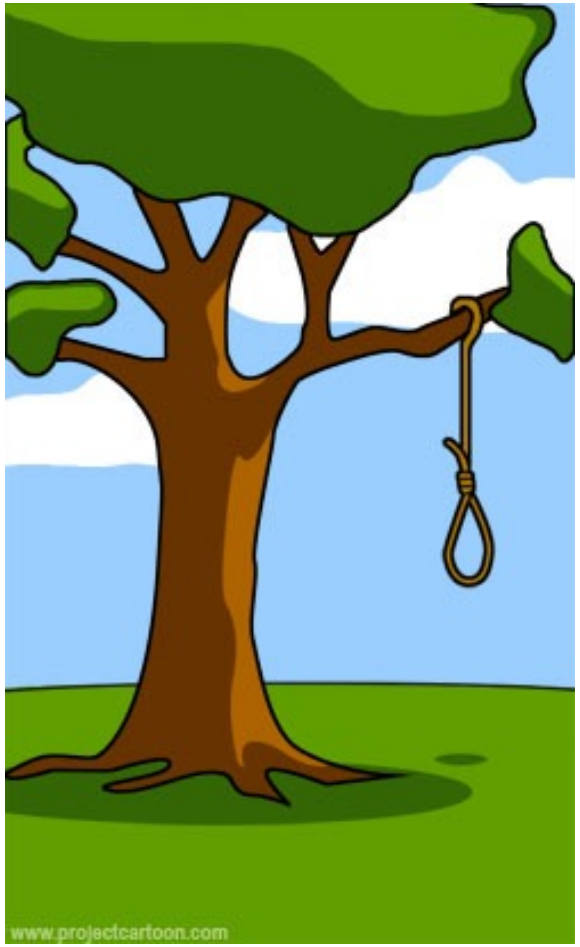
How IT Projects Really Work



Ce que le développeur a fait

Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work



Ce que les béta-testeurs ont reçu

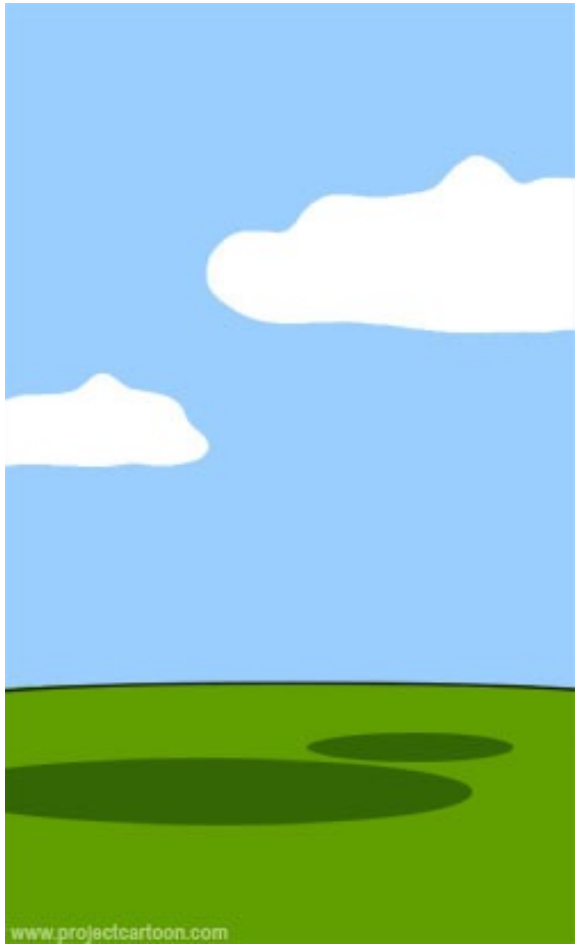
Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work

Comment les commerciaux l'ont décrit

Source : http://www.projectcartoon.com/cartoon/42

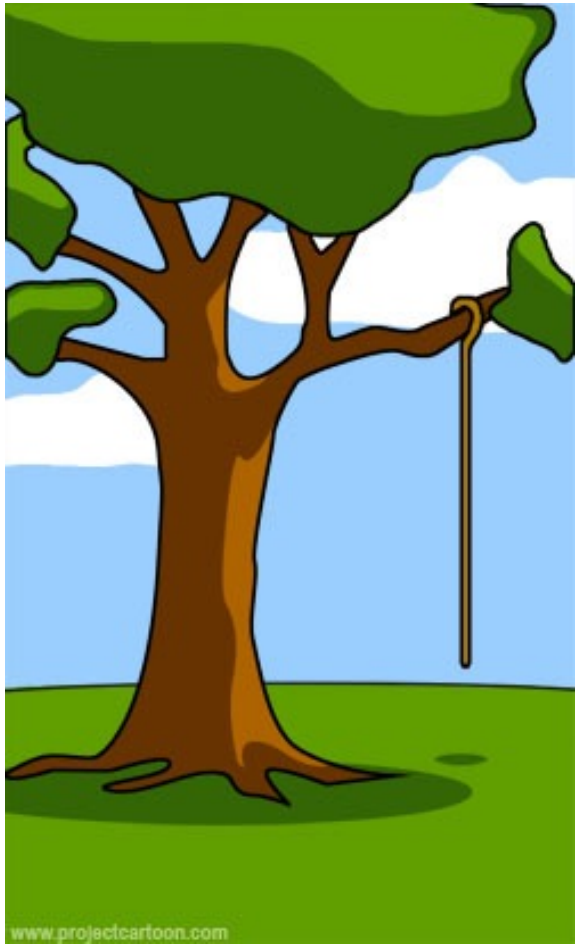
How IT Projects Really Work



Comment le projet a été documenté

Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work



Ce que la production
a installé

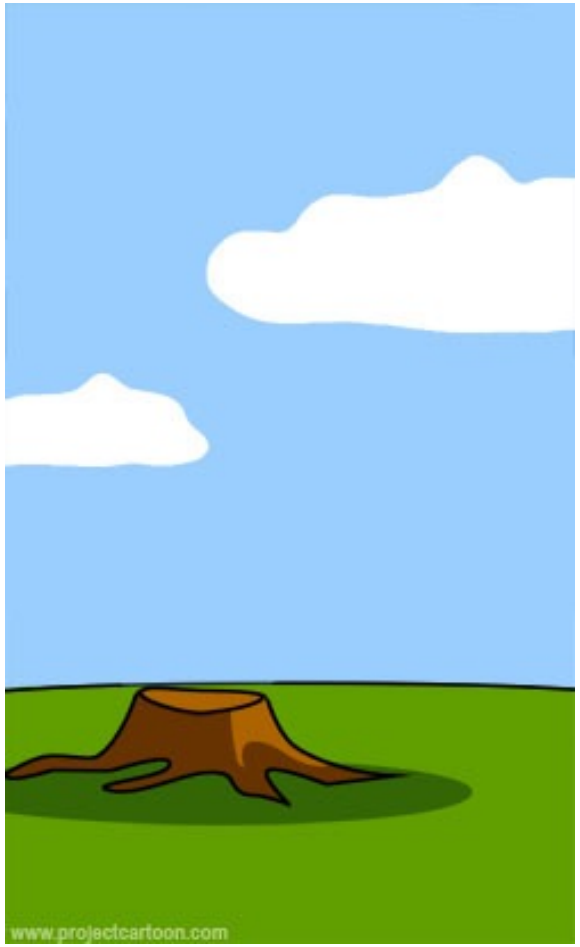
Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work

Comment le client a été facturé

Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work



Ce que l'assistance a apporté

Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work

Ce que le marketing a vendu

Source : http://www.projectcartoon.com/cartoon/42

How IT Projects Really Work

Ce dont le client avait vraiment besoin

Source : http://www.projectcartoon.com/cartoon/42

BIBLIOGRAPHIE

1. Modélisation Objet avec UML, *P.-A. Muller et N. Gaertner*, Eyrolles, 2005.
2. Présentation générale du langage de modélisation objet : U.M.L., *Bernard Espinasse*, cours de génie logiciel.
3. Software Engineering, Why? What, Alfred Strohmeier, École Polytechnique Fédérale de Lausanne, 2000.
4. Guide to the Software Engineering Body of Knowledge, IEEE Computer Society Professionnal Practices Committee, 2004.
5. Précis de génie logiciel, M-C Gaudel et al., Masson, 1996.
6. Génie Logiciel : principes, méthodes et techniques, A. Strohmeier et D. Buchs, Presses Polytechniques et universitaires romanes, 1996.
7. Cours de Génie Logiciel, François Jacquenet, Université de Saint Étienne
8. Cours de Génie Logiciel – Cycle de vie de Logiciels, M. Mostefai, ESI, 2011.