

ENSTA IN204

Introduction à JAVA

Olivier Sigaud

LIP6/AnimatLab

olivier.sigaud@lip6.fr

01.44.27.88.53

Plan du cours 7

- La conduite de projets
- Cycle de développement du logiciel
- Analyse / conception / implémentation
- Un processus << standard >>
- Rôle du chef de projet

Objectifs de la conduite de projets

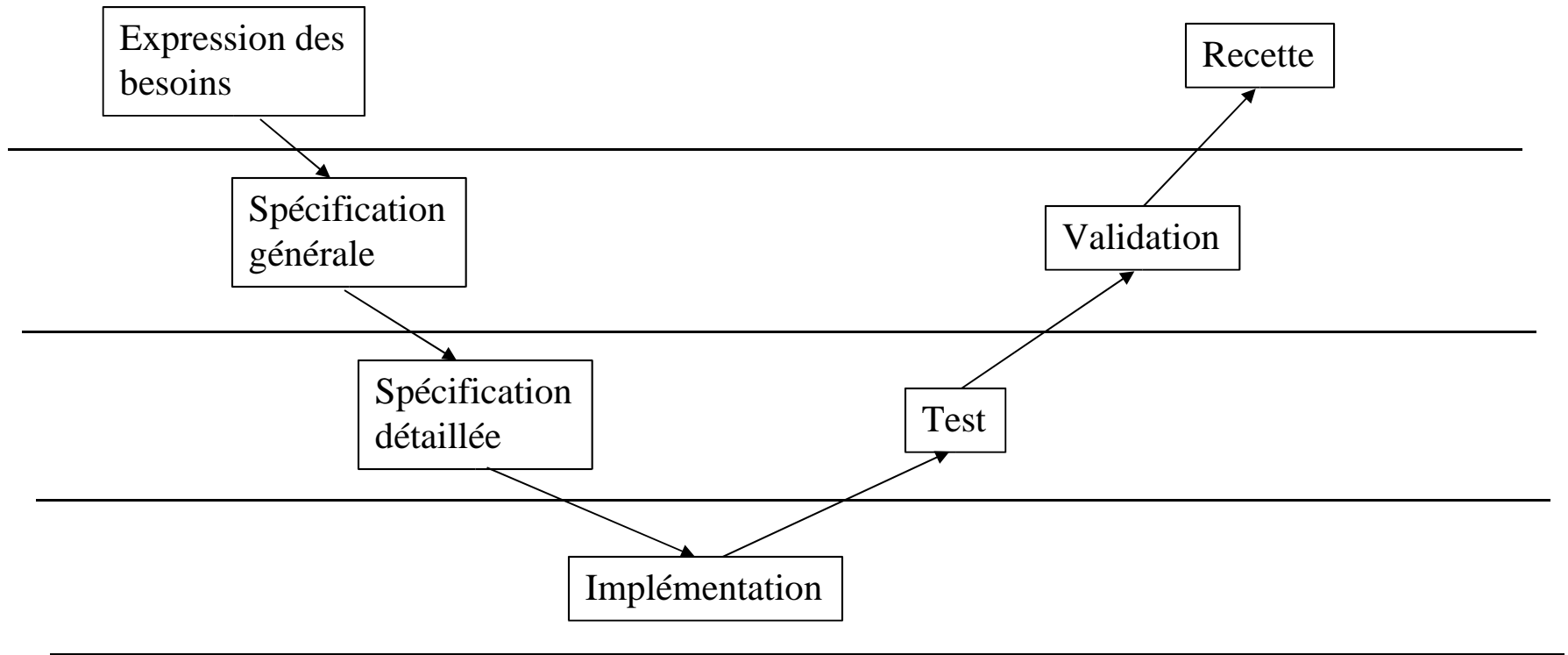
- Travailler à plusieurs (6, 15, 100, 1000..).
- Maximiser l'efficacité
- Maîtriser la complexité
 - Des logiciels de plus en plus complexes
 - **Développés par des équipes importantes**
- Estimer le temps, les ressources nécessaires
- Répondre aux besoins exprimés

Objectifs de la méthodologie

- Partager le travail (équipes de conception, équipes d'analystes programmeurs)
- Communiquer à un haut niveau d'abstraction
- Couvrir tout le cycle de développement
- Savoir où on va avant de coder :
 - Se mettre d'accord avec les utilisateurs
 - Eviter les erreurs de conception
- Favoriser la genericité, la réutilisabilité

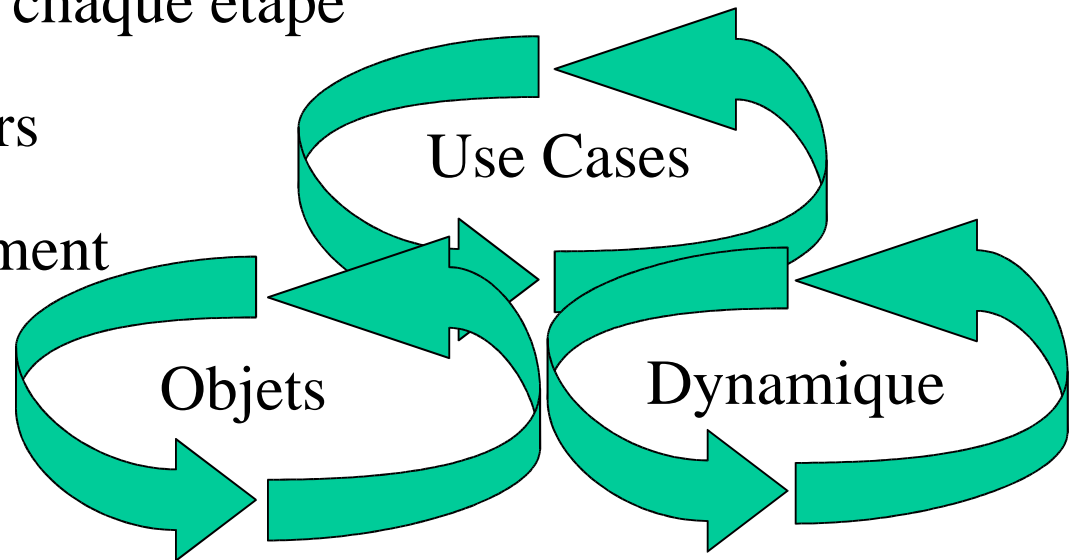
Cycle de développement

Le cycle de développement en V

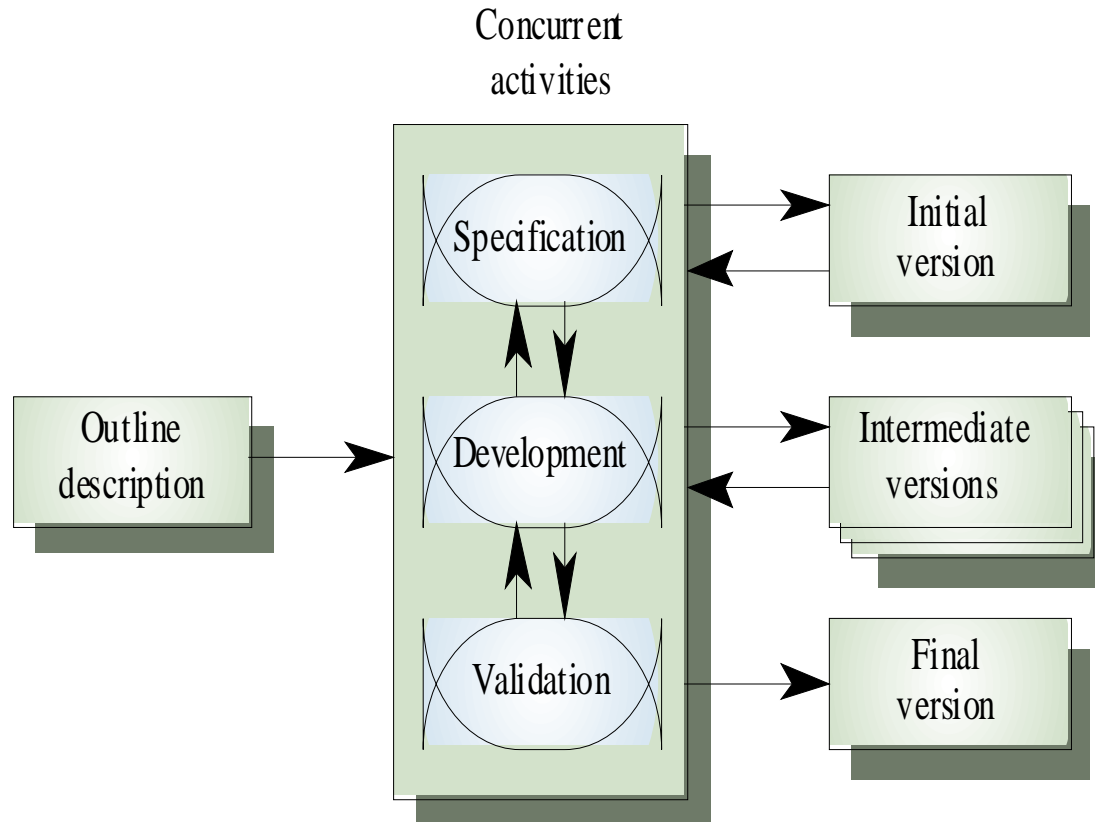


Démarche UML

- Définir des vues sur le logiciel
- Utiliser la redondance entre les vues
- Travailler les différentes vues en parallèle
- Vérifier la cohérence à chaque étape
- Nombreux allers-retours
- Faire valider régulièrement



Le processus unifié



Analyse et conception

Analyse / Conception

- **Analyse** = Comprendre les spécifications, en extraire les exigences, présenter formellement le problème à résoudre
 - Vérifier la conformité de l'analyse aux besoins
- **Conception** = Concevoir une solution au problème analysé
 - Eviter de figer des choix de conception pendant l'analyse
 - L'analyse = << quoi ? >>, la conception = << comment ? >>.
- **Implémentation** = Réaliser la solution sur une plate-forme et dans un langage donnés
- **Validation** = Vérifier la conformité de la solution aux spécifications

Processus d'analyse

- Le découpage en packages doit être fait tôt, mais **nécessite des réaménagements**
- Diagramme des cas => différents cas (et retours)
- use case => diagrammes des classes (et retours)
- use case => diagrammes de séquences/ajouts de classes
- identifier les automates nécessaires
- diagrammes de séquences => automates (synthèse)

Processus de conception

- **Partir d'un modèle en analyse**
 - partager le travail
 - détailler le contenu, les échanges
 - compléter les diagrammes de classes
 - compléter les diagrammes de séquence
 - **identifier les doublons, les problèmes**
 - modifier le partage (itérations)
- **Figurer les interfaces**
 - Classes et méthodes d'interface
 - Paramètres échangés, valeurs de retour
- **Passage à l'implémentation**

Un processus standard

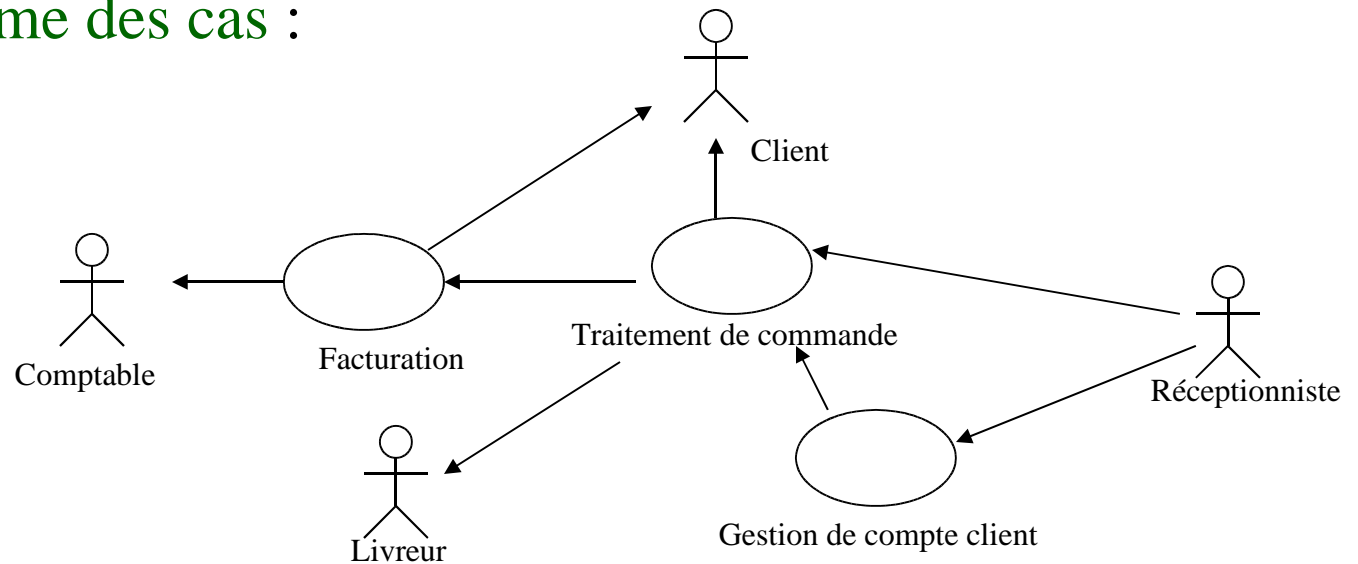
Conceptualisation

- Il faut conceptualiser le problème **du point de vue du client**
 - Identifier les **acteurs** (entités externes agissant sur le système)
- **Use case** : séquence d'actions réalisées par le système produisant un résultat **observable par un acteur**
- **Diagramme des cas** : ensemble des **Use cases**, doit décrire les exigences fonctionnelles du système
- Permet de :
 - **développer orienté utilisateur**
 - découper les grandes tâches
 - **communiquer** entre équipes et clients

Orienté SSII

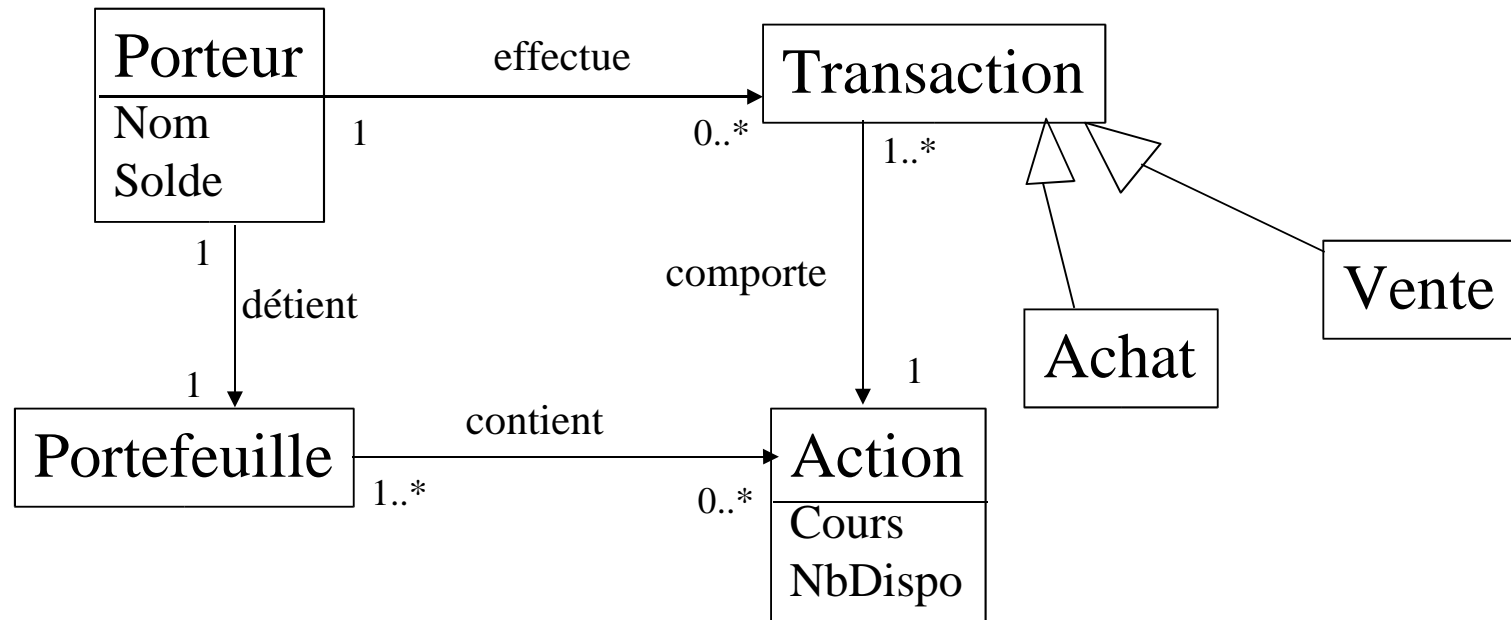
Cas d'utilisation

- Description textuelle :
 - nom, résumé, contexte, description, exceptions, acteurs, effets...
 - on en extrait les objets, actions, états... de la modélisation
- Diagramme des cas :



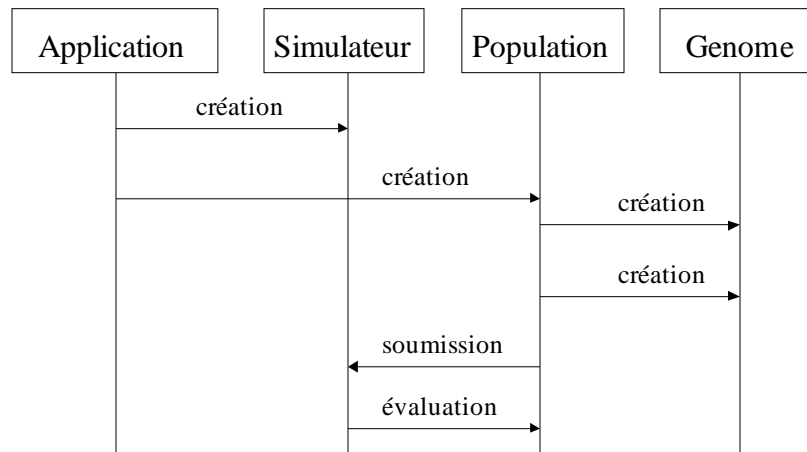
A chaque **Use case** correspond un **diagramme d'objets participants**

Classes d'analyse



Séquences d'analyse

Le diagramme de séquence



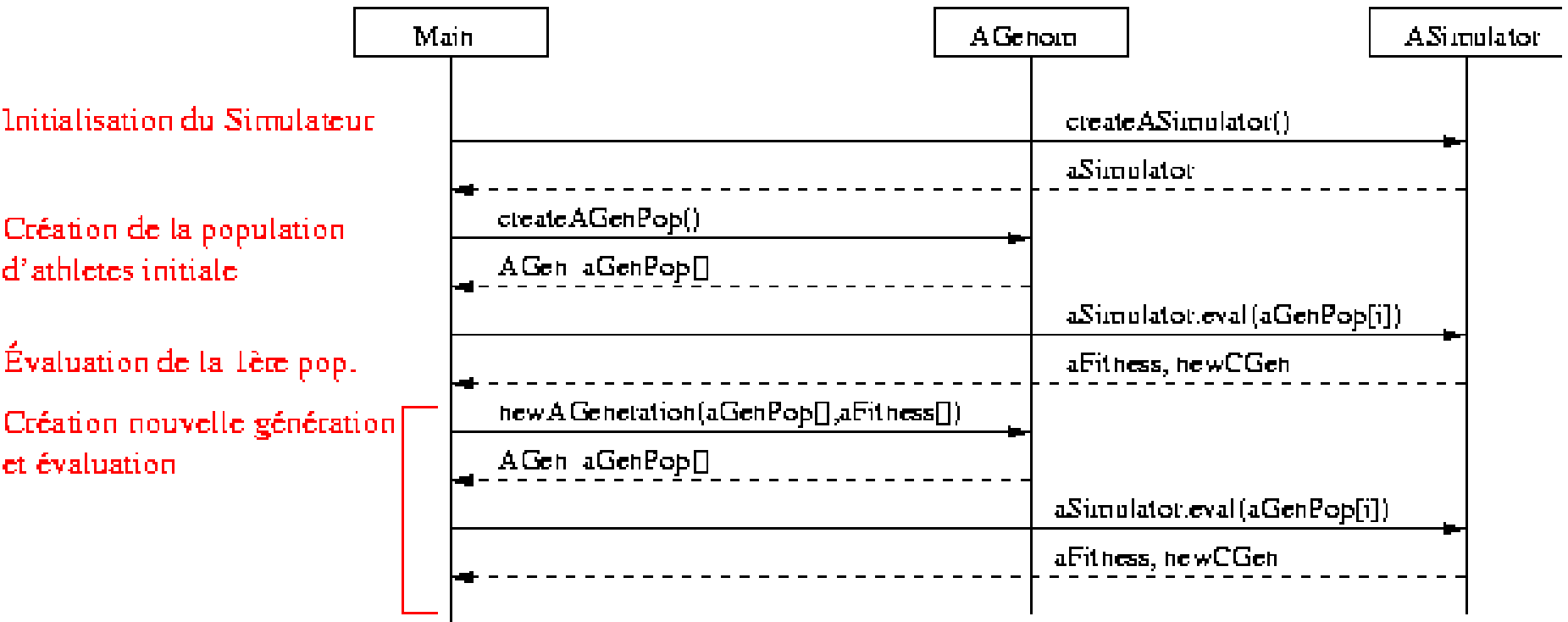
Permet d'identifier les interactions entre objets concernés



Classes de conception

<Client> Portefeuille
+ NomPorteur : string - Id : string # Solde : double # \Solvable : boolean = true <u>NbMaxTitres : int = 1000</u>
- Débiter(montant:double) - Créditer(montant:double) - Acquérir(titre:Action, volume:int) - Céder(titre:Action, volume:int) + Acheter(titre:Action, volume:int) + Vendre(titre:Action, volume:int)
Texte libre

Séquences de conception



Reverse engineering

- Reconstruire les vues UML à partir d'un code
- Produit surtout le **diagramme de classes**
- Aide à comprendre un code ancien ou mal documenté
- **Le résultat est souvent un << plat de nouilles >>**
- La tendance est au maintien de cohérence entre code et vues UML
- A prendre comme une source de documentation comme une autre pour l'analyse

Délicat à utiliser

Génération de squelettes

- Se fait dans un **langage cible** (C++, Java..).
- Utilise surtout les **diagrammes de classes**
- Engendre
 - les packages, les classes, les attributs
 - la signature des méthodes
 - les associations, les cardinalités
 - composition/agrégation/héritage
- **N'engendre pas l'implémentation des algorithmes**
- Utilise parfois les automates

Implémentation

- Il s'agit de remplir le code des méthodes dans le langage cible
- On va (lentement) vers une automatisation de l'implémentation
- Tout retour sur la conception (nouveaux paramètres, changement de type de retour, nouvelles méthodes, nouvelles classes) est une source de problème, doit être signalé et géré

Tests

- A prévoir très tôt dans l'analyse
- Définir des tests unitaires
- Effectuer des tests de non régression
- Junit
- Créer une classe TotoTest pour chaque classe Toto
- Lister tous les comportements qui doivent être testés

Installation et maintenance

- Distinguer l'installation pendant le développement et l'installation sur site (éviter tous les chemins en dur.).
- Fournir une documentation d'installation bien faite
- Penser éventuellement à la désinstallation
- Prévoir les opérations de maintenance (manuel de maintenance.).

Gestion de la documentation

- Tout diagramme doit indiquer clairement ce qu'il représente
- Rassembler les diagrammes représentant différentes instances d'un même scénario
- Assurer la traçabilité entre cas d'utilisation, diagrammes de classe, diagrammes de séquence, et code produit
- Recours intensif aux liens hypertextes

Un diagramme = une page A4 max !

Rôles du chef de projet

Maîtriser le projet

- Maîtrise technique
 - faire les bons choix sur des bases solides
 - évaluer la difficulté
- Maîtrise du temps
 - assurer la coordination
 - anticiper les prises de retard
- Rôle organisationnel
- Rôle relationnel

Une personnalité polyvalente

Clefs d'une bonne gestion

- Pour bien gérer, savoir anticiper
- Déléguer plutôt que faire
- Intervenir à bon escient
 - Sur le plan technique
 - Sur le plan organisationnel
 - Sur le plan relationnel
- Motiver, assumer, épauler, rendre compte

Quelques erreurs classiques

- Le lièvre et la tortue
- La chevauchée fantastique
- Le combat des chefs
- Le monstre du lac
- Les aventuriers de l'énoncé perdu
- Le capitaine Fracasse

Conclusions

Conclusion : conduite de projet

- Un travail d'architecture
- Réalisé sur des vues graphiques indépendantes du codage
- Le codage comme activité de plus en plus secondaire
- Un travail collectif difficile à maîtriser

Conclusion sur UML

- UML : un langage très riche
- La méthodologie n'est pas figée
- Les outils sont embryonnaires
- L'évolution va très vite (UML 2.0)
 - vers la vérification de cohérence (ToDoList)
 - vers la production automatisée de code
- suivre l'actualité sur le web
- Rien ne remplace l'expérience pour utiliser UML efficacement

Pour vous, l'analyse, concrètement

Analyse : étape 1

- Conduire l'analyse :
 - Déterminer tout ce qu'il faut faire
 - Se partager le travail
 - Identifier les problèmes
 - Choisir des solutions générales
- Objectifs fin de séance 1 :
 - chacun sait quel problème il doit contribuer à résoudre
 - lecture des articles et reverse engineering des codes

Analyse : étape 2

(mise en commun)

- A partir des *reverse engineering* des solutions existantes
- Faire des diagrammes d'ensemble qui permettent de voir où sont les interactions
- Faire des diagrammes de classes par parties
- Identifier les éventuelles redondances, affiner les frontières entre les parties
- Objectif : se faire une vision cohérente d'une solution d'ensemble

Au boulot !
(après le CC Java)
(et les vacances)
(et les fêtes)