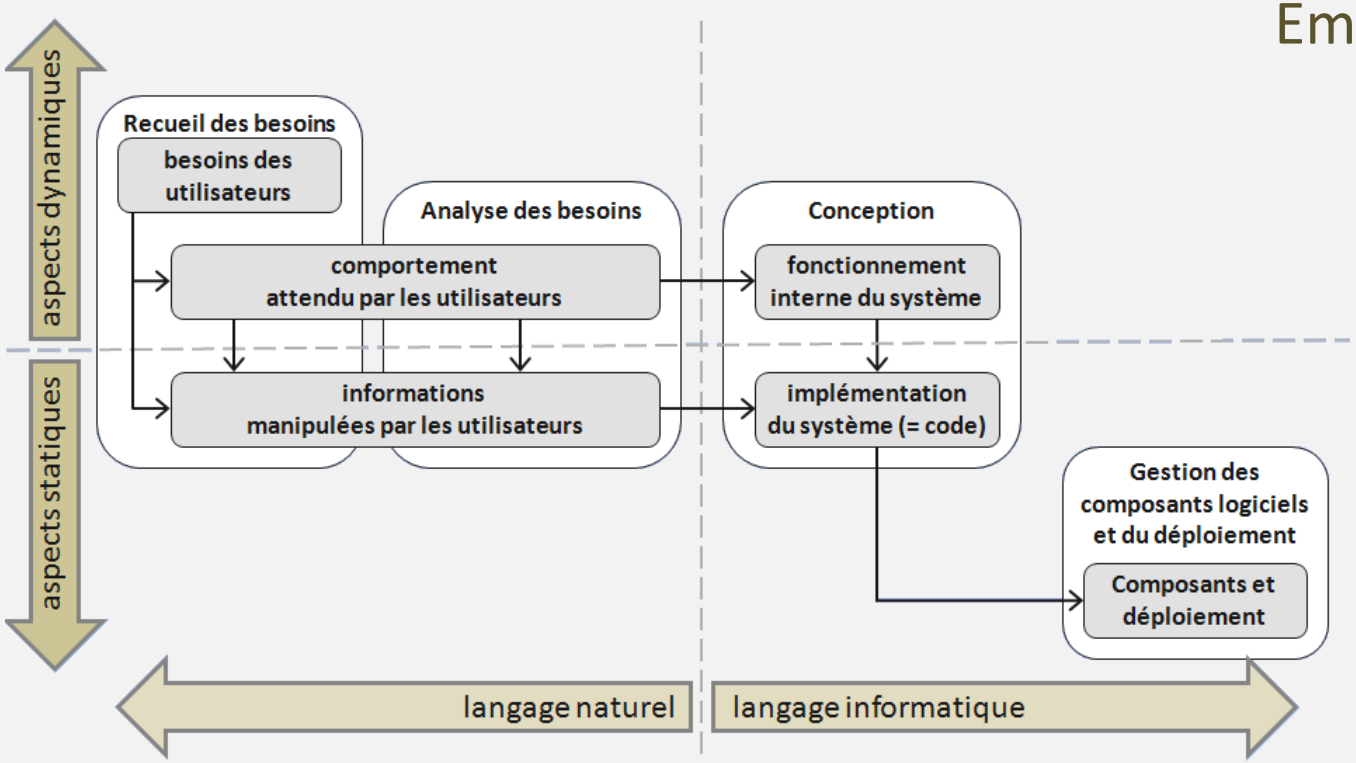
UML
Mise en œuvre dans un projet

Emmanuel Pichon
2013



Sommaire

- **Introduction**
- Rôles et activités dans un projet
- Définir la méthode de votre projet
- Adapter la modélisation à la méthode de votre projet
- Conseils de mise en œuvre de la modélisation
- Au-delà d'UML et des concepts « objet »

Peut-on développer un logiciel sans modéliser ?

- OUI ! Mais seulement si
  - Vous développez seul (pas de communication)
  - Un logiciel assez simple (une seule itération)
  - Un logiciel est autonome (pas d'intégration)
  - Vous êtes l'utilisateur du logiciel (pas de recette utilisateur)
  - Le logiciel évolue peu (pas d'analyse d'impacts)

- C'était le cas dans les années 80 quand on développait en BASIC sur Apple II

La modélisation garantit-elle une bonne conception ?

- NON !
  - Parce qu'il n'y a pas une bonne conception mais plusieurs
  - On peut faire des erreurs de modélisation
    comme on peut faire des erreurs de conception

- Il faut d'abord respecter les contraintes imposées au projet
  - Les normes et les standards
  - Les contraintes d'architecture (client léger/lourd, *n-tiers*, …)
  - Les exigences (sécurité, performance, disponibilité, …)
  - Les contraintes des langages et des outils d'implémentation

Build the system <u>right</u>

Introduction
Comment garantir une bonne conception ?

- Toujours appliquer le principe d'encapsulation !
  - Limite les impacts d'une évolution ou d'une erreur de conception

- Mettre en œuvre les bonnes pratiques
  - Patterns GRASP (Craig Larmann) = « SAISIR »
    - General Responsibility Assignment Software Patterns
      = principes « objet » d'affectation des responsabilités
    - Exemples : couplage faible, forte cohésion, expert en information, polymorphisme, …
  - Patterns GoF (Gang of Four)
    - Patterns orientés implémentation (diagrammes de classes)
  - D'autres patterns existent
    - n-tiers, MVC, loi de Demeter (ne pas parler aux inconnus) …

Build the system <u>right</u>

Introduction
Un bon logiciel est-il un logiciel bien conçu ?

- Oui, pour le développeur ou l'équipe qui va le maintenir

- Mais, une bonne conception ne garantit pas de répondre aux besoins des utilisateurs

- Le recueil et l'analyse des besoins sont indispensables pour
  - Identifier et spécifier les besoins des utilisateurs
  - Définir ce qui sera testé

Build the <u>right</u> system

Sommaire

- Introduction
- **Rôles et activités dans un projet**
- Définir la méthode de votre projet
- Adapter la modélisation à la méthode de votre projet
- Conseils de mise en œuvre de la modélisation
- Au-delà d'UML et des concepts « objet »

Les rôles dans le projet (non liés à une méthode)

- Maitrise d'ouvrage (MOA)
  - Participe au cadrage général du projet
  - Exprime les besoins de l'utilisateur final
  - Participe et valide le cadrage et l'analyse des besoins
  - Prépare et exécute la recette du logiciel

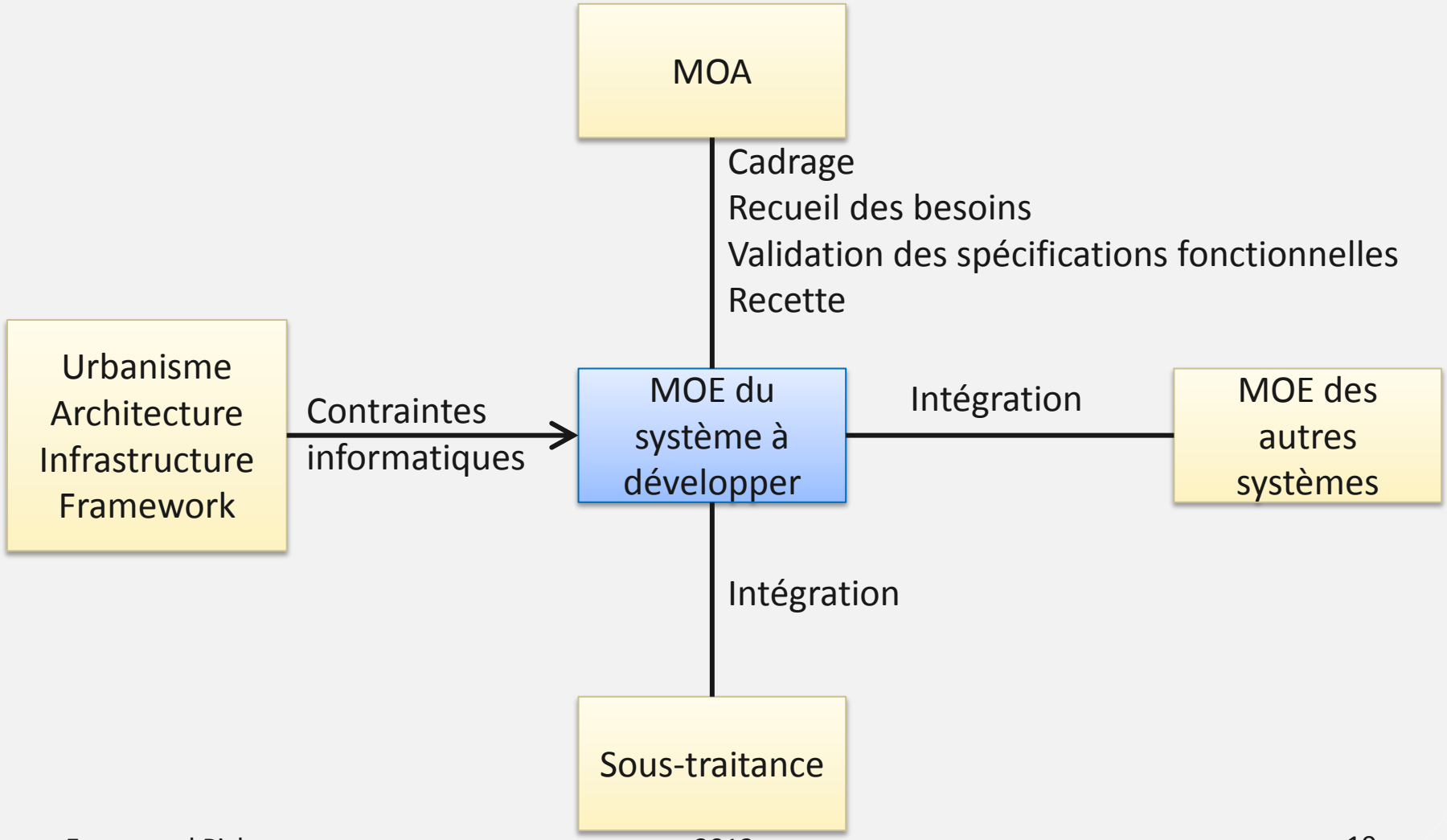
- Risques liés à la MOA
  - MOA non représentative de l'utilisateur final
  - MOA multiple et/ou en concurrence
  - Pas de MOA ou MOA non identifiée (projet interne ou projet d'infrastructure)

Les rôles dans le projet (non liés à une méthode)

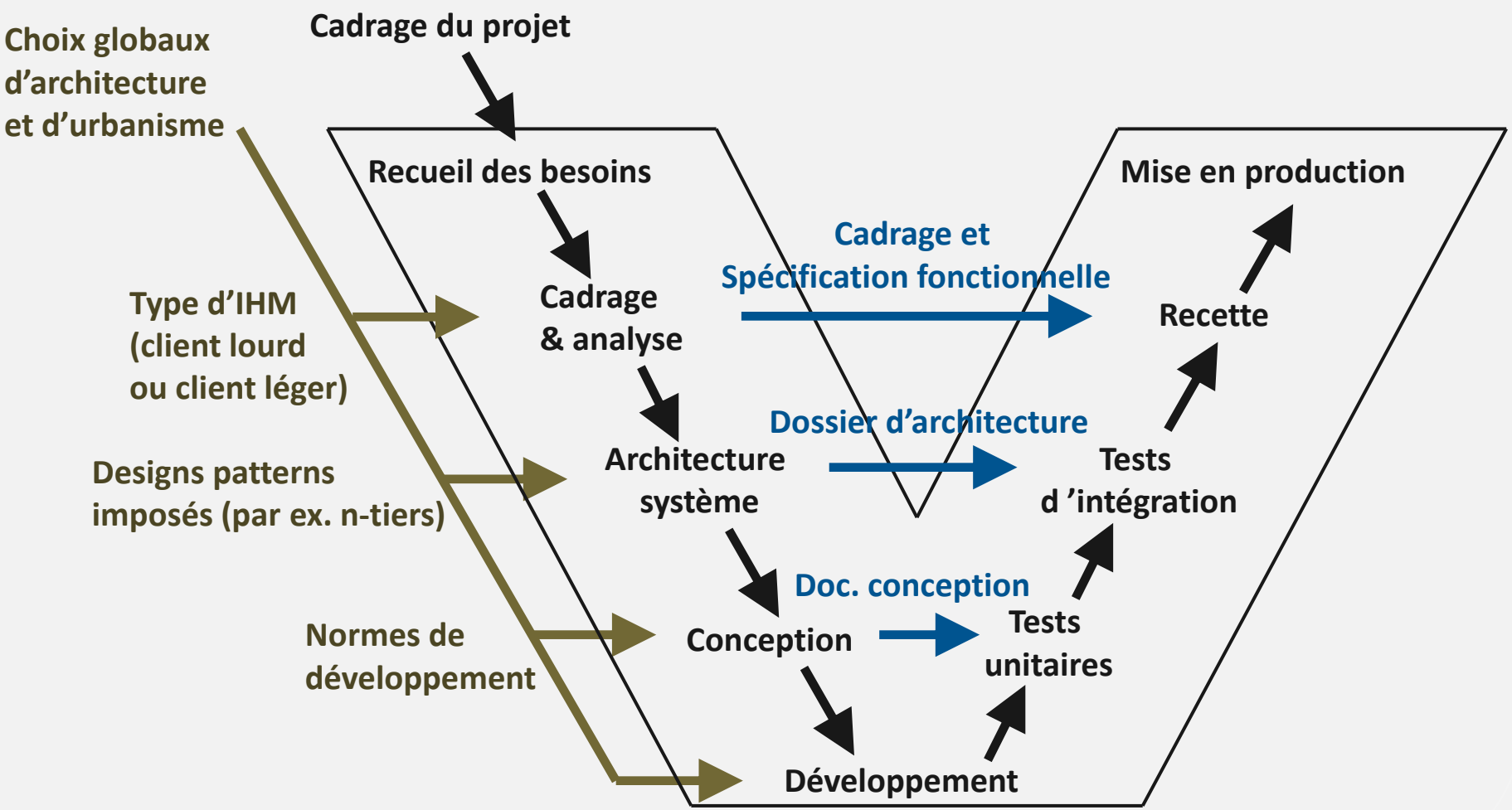
- Maitrise d'œuvre (MOE)
  - Le chef de projet cadre le projet, gère les ressources, le planning, les risques, les arbitrages, …
  - L'analyste participe au cadrage, recueille et analyse le besoin fonctionnel
  - L'architecte participe au cadrage, recueille les exigences d'architecture et définit l'architecture du système
  - Le concepteur/développeur conçoit, développe et teste unitairement
  - L'intégrateur prépare et exécute les tests d'intégration (en relation avec d'autres MOE)
  - L'exploitant met en production

Focus du cours

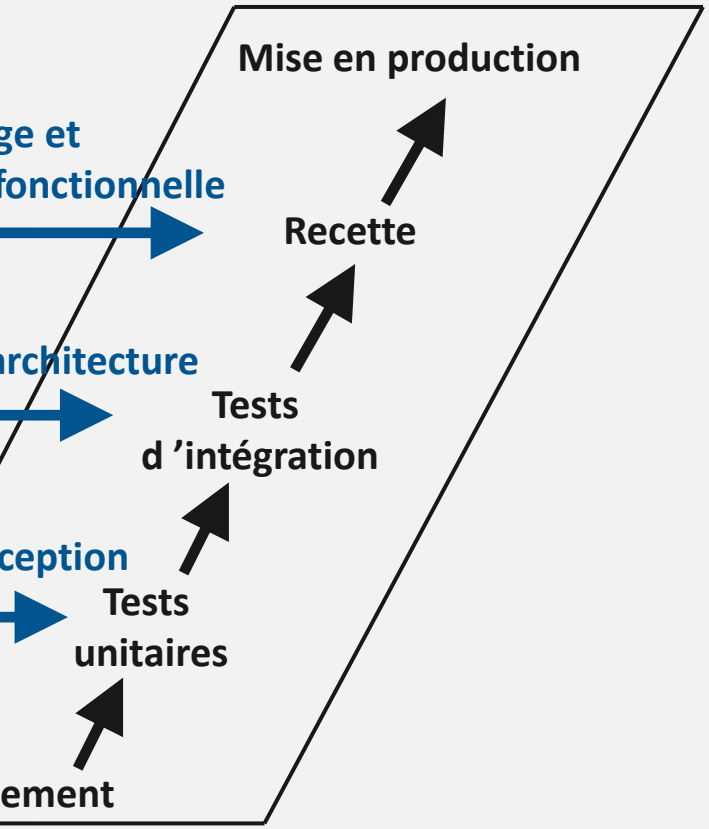
Les rôles dans le projet et leurs relations



Activités de fabrication d'un logiciel (indépendamment de la méthode du cycle en V)



Zoom sur les tests (où beaucoup de projets dérapent…)



- Les aspects dynamiques servent à préparer les scénarios de test
- Les aspects statiques servent à préparer les jeux de données
- Les faiblesses des phases amont
  - apparaissent lors de l'intégration, la recette ou même en production
  - et sont plus couteuse à corriger
- L'architecture système sert de référence pour l'intégration
- Les spécifications servent de référence pour la recette

Sommaire

- Introduction
- Rôles et activités dans un projet
- **Définir la méthode de votre projet**
- Adapter la modélisation à la méthode de votre projet
- Conseils de mise en œuvre de la modélisation
- Au-delà d'UML et des concepts « objet »

Quelle méthode pour le projet ?

- Méthodes classiques : en cascade, en V, en Y, …
  - Etapes du projet en séquence
  - Effet tunnel (pas de visibilité intermédiaire)

- Méthodes itératives : un cycle projet pour chaque itération
  - Une itération = une partie du logiciel mise en production
  - Visibilité intermédiaire

- Méthodes agiles : XP, SCRUM, …
  - Visibilité court terme

- UML s'applique à toutes ces méthodes
  - Plusieurs niveaux de détails possibles : esquisse, analyse, cadrage, spécification, architecture, conception, …

Adapter la méthode au contexte du projet

- Identifier les activités que votre équipe prend en charge dans le projet
  - Identifier les éléments en entrée de vos activités (normes, architecture, …)
  - Définir les livrables en sortie de vos activités (dossier de spécification, dossier d'architecture, dossier de conception, code logiciel, …)

- Identifier les contraintes imposées au projet
  - Situation de sous-traitance, projet au forfait (engagement de résultat)
  - Méthode imposée par l'entreprise pour l'ensemble du projet ou pour certaines activités (par exemple MCD/MLD/MPD pour base de données)
  - Identifier les exigences projet (délai, ressources, …)

- Intégrer une gestion des risques liés à la fabrication de votre logiciel
  - Prototyper les parties nécessitant une vérification de faisabilité
    Cycles courts de développement

Sommaire

- Introduction
- Rôles et activités dans un projet
- Définir la méthode de votre projet
- **Adapter la modélisation à la méthode de votre projet**
- Conseils de mise en œuvre de la modélisation
- Au-delà d'UML et des concepts « objet »

Adapter la modélisation à votre méthode

- Modéliser pour communiquer à l'oral
  - Echanger des idée en réunion
  - Niveau de modélisation : esquisse
- Modéliser pour définir un contrat = une référence commune
  - Cadrage et spécification = contrat MOA-MOE
  - Architecture système = contrat MOE-MOE
  - Conception = contrat inter-développeurs
  - Adapter le niveau de modélisation
- Modéliser pour analyser ou concevoir une partie complexe

Sommaire

- Introduction
- Rôles et activités dans un projet
- Définir la méthode de votre projet
- Adapter la modélisation à la méthode de votre projet
- **Conseils de mise en œuvre de la modélisation**
- Au-delà d'UML et des concepts « objet »

Conseils pour le cadrage des besoins

- Diagramme de cas d'utilisation
- Pour chaque cas d'utilisation
  - Description textuelle courte du comportement attendu
- Diagramme de classes pour les principales entités
- Pour chaque classe
  - Diagramme d'états si le cycle de vie est structurant
- Exigences non fonctionnelles
  - Description textuelle

- NB : utiliser le vocabulaire de l'utilisateur final
  - Définir un glossaire métier

Conseils pour la spécification des besoins

- Pour chaque cas d'utilisation
  - Diagramme de séquence pour les interactions avec les utilisateurs et les systèmes externes
  - Diagramme d'activité si processus non interactif
  - Décrire le fonctionnement interne sans décrire l'implémentation
- Diagramme de classes
- Pour chaque classe
  - Diagramme des états cités dans les règles de gestion

Conseils pour la spécification des besoins

- Eviter le copier/coller de modélisation et/ou de spécification
  - Factoriser les éléments communs/transverses
- Si une règle de gestion est trop complexe à documenter
  - Décomposer cette règle à l'aide d'un diagramme d'activités
  - Si besoin d'un diagramme d'interaction, ce n'est pas une règle de gestion
- Si besoin, ajouter des maquettes d'écran
  - Statiques (ne pas confondre maquette et prototype)
  - *Look* neutre (pour éviter les débats sur la charte graphique)

Conseils pour la spécification des besoins
Complétude et cohérence

- Complétude : spécifier jusqu'à quel niveau de détail ?
  - Répondre à l'expression des besoins
  - Spécifier tout ce que l'on souhaite tester en recette

- Assurer la cohérence de la spécification
  - Chaque donnée et chaque état cités dans une règle de gestion doivent être modélisés et documentés
  - Exemple de règle de gestion citant une donnée
    - si le solde du compte (EF_43_02) est négatif, le virement est impossible

Conseils pour la spécification des besoins
Distinction anomalie/évolution

- Anomalie = non-conformité du logiciel
  - Concernant son comportement lors des tests ou en production
  - Différentes type d'anomalies
    - D'implémentation : non-conformité à la conception
    - De conception : non-conformité à la spécification
    - De spécification : non-conformité à l'expression des besoins

- Evolution
  - Nouveau besoin
  - Ou l'expression des besoins est modifiée

Nécessité de définir un référentiel partagé entre les acteurs

Conseils pour la spécification des besoins
Les références

- Objectifs du référencement
  - Promouvoir & renforcer la notion de référentiel (= ce qui fait référence)
  - Faciliter …
    - La communication avec tous les outils (UML, Word, mails, code…)
    - La traçabilité entre les équipes et les activités
    - La recherche d'un élément pour analyse d'impacts et maintenance
    - La cohérence de la spécification (exemple d'une règle de gestion citant une donnée)

- Référencer les éléments unitaires de la spécification, par exemple
  - Règle de gestion = exigence fonctionnelle unitaire
    - Exemple : RG_12 vérifier solde du compte
  - Structure de données et donnée unitaire
    - Exemple : EF_43 Compte, EF_43_02 solde du compte
  - (+ possibilité de référencer les écrans en conception)

Conseils pour la spécification des besoins
Les références

- Conseils pour définir des références
  - Une bonne référence est stable dans le temps et courte

- Exemple de format : <type>_<identifiant>
  - <type> = type de l'élément référencé, par exemple RG pour une règle de gestion
  - « _ » est un exemple de caractère réservé aux références pour faciliter les recherches avec les outils
  - <identifiant> numérique et non significatif
    - = indépendant du fonctionnel et des évolutions

- Attendre que l'analyse soit assez avancée pour référencer (pour éviter les renumérotations)

Conseils pour l'architecture et la conception

- Architecture systèmes (si plusieurs sous-systèmes)
  - Identification des sous-systèmes, de leur responsabilité et de leurs relations (diagramme de classes ou de composants)
  - Interactions des sous systèmes (diagramme de séquence)
  - Objectifs
    - Répartition de la conception et du développement sur plusieurs équipes et/ou plusieurs personnes
    - Préparer les tests d'intégration

- Conception d'un sous système
  - Conception objet (aspects dynamiques et statiques)
  - Jusqu'à quel niveau (synthèse des principales classes ou toutes les classes)
  - Application des *design patterns*
  - Préparation des tests unitaires

- Dans les deux cas, outil de communication entre concepteurs

Modélisation des données XML (échange de données avec un autre système)

UML (niveau conception)	XML	Correspondance
Paquetage	Schéma	
Classe	Racine + type complexe	Nom et type
Attribut	Elément	Nom et cardinalités
Association orientée	Elément	Rôle et cardinalités
Héritage simple	Elément extension	Base = Super classe
Classe «Enumeration»	Type simple + énumération	Valeurs = attributs

Les spécificités XML sont modélisées par des stéréotypes et des valeurs marquées dans le cadre d'un profil UML dédié à XML.

- Exemples de stéréotypes : XSDschema, XSDgroup, XSDtopLevelElement, XSDunion, XSDsequence, XSDchoice, XSDelement, XSDattribute, XSDany, XSDrestriction…
- Exemples de valeurs marquées : targetNamespace, version, length, …

Il ne reste plus qu'à coder…

- Génération automatique du code à partir du modèle
  - Structure générale (packages)
  - Déclaration des classes, des opérations et des attributs

- Il reste à coder les opérations en respectant
  - La spécification : interactions et règles de gestion
  - Les diagrammes de séquence de conception
  - Les choix d'architecture et de conception

- Mise à jour du modèle et du code
  - Dépend du niveau d'intégration de l'outil de modélisation avec l'environnement de développement
  - En général, limité aux aspects statiques (packages et classes)

- Possibilité de Reverse Engineering sur code existant

Synthèse

- Modéliser le minimum nécessaire
  - Identifier à qui et à quoi va servir votre modélisation
  - Définir quelle partie de la modélisation sera maintenue

- Exemple de non modélisation pour la conception
  - Rétro spécification d'un logiciel dont la conception et le périmètre sont stables
  - Lien direct entre spécification et code logiciel par les références des règles de gestion insérées dans le code sous forme de commentaires

Sommaire

- Introduction
- Rôles et activités dans un projet
- Définir la méthode de votre projet
- Adapter la modélisation à la méthode de votre projet
- Conseils de mise en œuvre de la modélisation
- **Au-delà d'UML et des concepts « objet »**

SOA : Service Oriented Architecture

- Approche « objet » appliquée au niveau d'un système d'information
  - Objectif : obtenir l'agilité des systèmes « objet » pour les systèmes hétérogènes (dont les anciens systèmes en silos)

- Principe
  - Un service encapsule une fonctionnalité
    - Distinction entre utilisation et implémentation
    - Notion d'interface avec un fournisseur et des clients

Les différences par rapport à l'approche objet « classique » (= implémentation mono langage)

- Implémentation multi langages et multi plateformes
- Messages synchrones ou asynchrones
- Formats d'échange interopérables (XML, JSON, …)
- ESB = *Enterprise Service Bus*
  - *Permet un couplage faible*
    - *Pas de connexion « point à point » entre les systèmes*
    - *Un couplage reste dans les données échangées (pivots)*
  - *Routage dynamique sur la base des entêtes XML*
  - *Possibilités de transformer les données*
  - Intégration d'un annuaire de services sur le principe du web
  - Gestion de la sécurité des échanges

Résumé des approches SOA

- Approche top-down
  - Modélisation d'un processus métier (souvent avec BPMN)
  - Définition des applications (partie visible des utilisateurs)
  - Identification des services supportant ces applications
- Approche bottom-up
  - Identification des services disponibles dans le SI existant
  - Agrégation des services dans des applications
- Les deux sont complémentaires

Synthèse sur SOA

- Avantage
  - Tout est potentiellement intégrable (anciens et nouveaux systèmes)

- Points de vigilance
  - Il est souvent difficile
    - de trouver le bon niveau de découpage des services
    - de définir les formats pivots
  - Baisse des performances si trop de services ou trop d'interactions
  - Culturels
    - Faire communiquer des équipes indépendantes auparavant
    - Quel niveau de confiance pour utiliser un service développé par une autre équipe ? (Syndrome « Not Invented Here »)