Processus de Développement Logiciel

Cours M14

Pierre Gérard

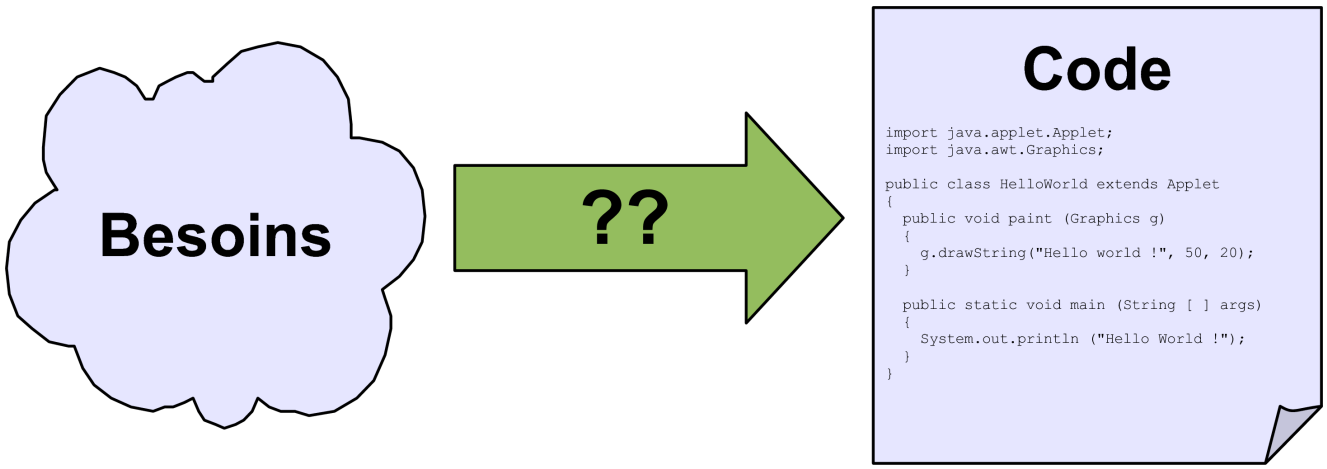
Université de Paris 13 – IUT Villetaneuse – Formation Continue

Licence Pro SIL

LaTeX

Plan

Nécessité d'une méthode



Processus de développement

Ensemble d'étapes partiellement ordonnées, qui concourent à l'obtention d'un système logiciel ou à l'évolution d'un système existant.

- **Objectif :** produire des logiciels
  - De qualité (qui répondent aux besoins de leurs utilisateurs)
  - Dans des temps et des coûts prévisibles
- A chaque étape, on produit
  - Des modèles
  - De la documentation
  - Du code

Méthode = Démarche + Langage

- La méthode MERISE fournit
  - Un langage de modélisation graphique (MCD, MPD, MOT, MCT...)

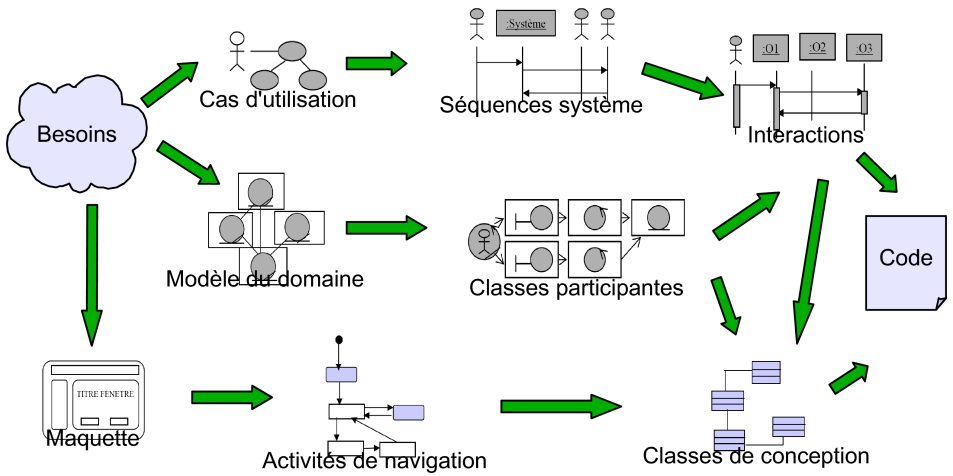
  ET Une démarche à adopter pour développent un logiciel
- **UML n'est qu'un langage**
  - Spécifie comment décrire des cas d'utilisation, des classes, des interactions...
  - Ne préjuge pas de la démarche employée
- **Méthodes s'appuyant sur UML**
  - RUP (Rational Unified Process) - par les auteurs d'UML
  - XP (eXtreme Programming) - pouvant s'appuyer sur UML

Méthode minimale

Objectif

Résoudre 80% des problèmes avec 20% d'UML

- Proposition d'une méthode **archi-minimale**
  - Vraiment très très nettement moins complexe que RUP
  - Adaptée pour des projets modestes
  - Minimum vital pour qui prétend utiliser *un peu* UML

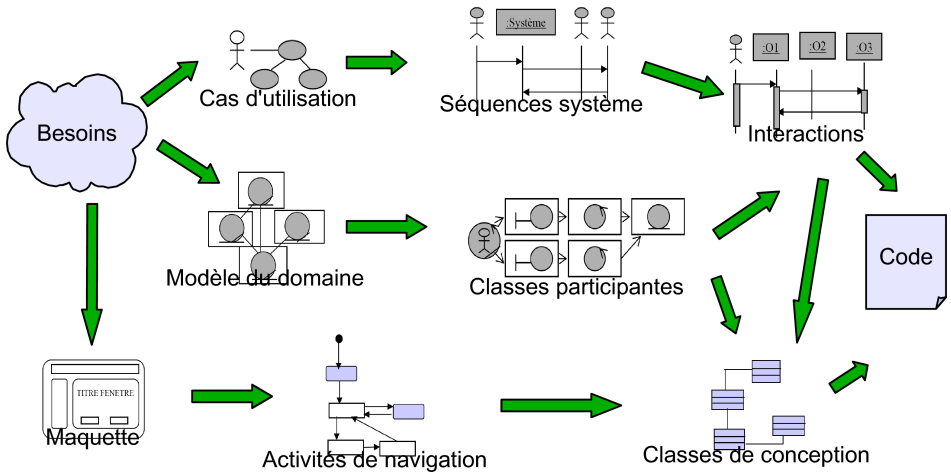


Méthode minimale

Objectif

Résoudre 80% des problèmes avec 20% d'UML

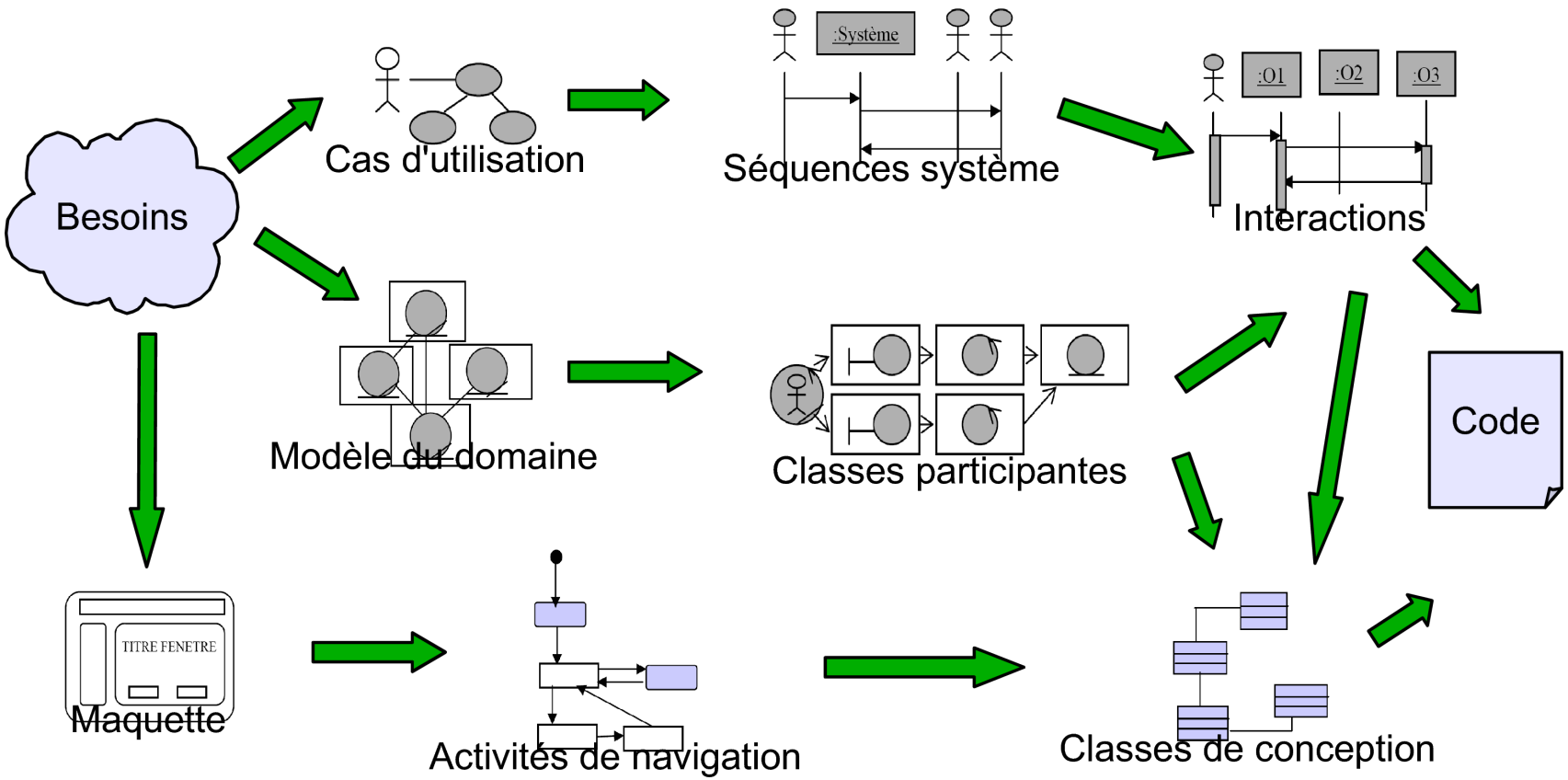
- Inspirée de
  - « UML 2 - Modéliser une application web »
    - **Pascal Roques**
    - Editions Eyrolles (2006)



Méthode minimale

Objectif

Résoudre 80% des problèmes avec 20% d'UML



Cas d'utilisation

- Comment aboutir au diagramme de cas d'utilisation ?
  1. Identifier les limites du système
  2. Identifier les acteurs
  3. Identifier les cas d'utilisation
  4. Structurer les cas d'utilisation en packages
  5. Ajouter les relations entre cas d'utilisation
  6. Classer les cas d'utilisation

Exemple de classement

Cas d'utilisation	Priorité	Risque
Rechercher des ouvrages	Haute	Moyen
Gérer son panier	Haute	Bas
Effectuer une commande	Moyenne	Haut
Consulter ses commandes en cours	Basse	Moyen
Consulter l'aide en ligne	Basse	Bas
Maintenir le catalogue	Haute	Haut
Maintenir les informations éditoriales	Moyenne	Bas
Maintenir le site	Moyenne	Bas

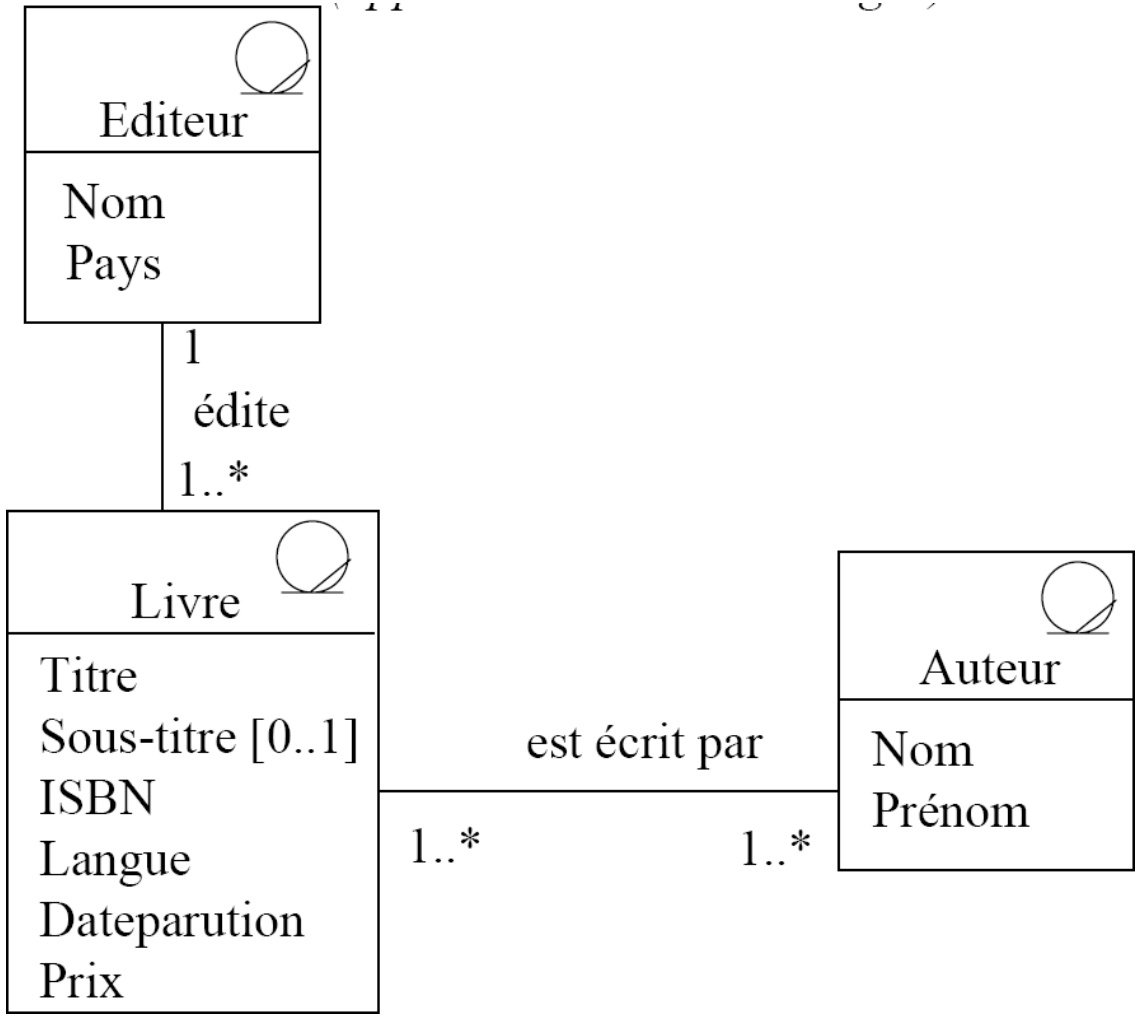
- Un tel classement permet de déterminer les cas d'utilisation centraux en fonction
  - de leur priorité fonctionnelle
  - du risque qu'il font courrir au projet dans son ensemble
- Les fonctionnalités des cas les plus centraux seront développées en premier lieu

Modèle du domaine

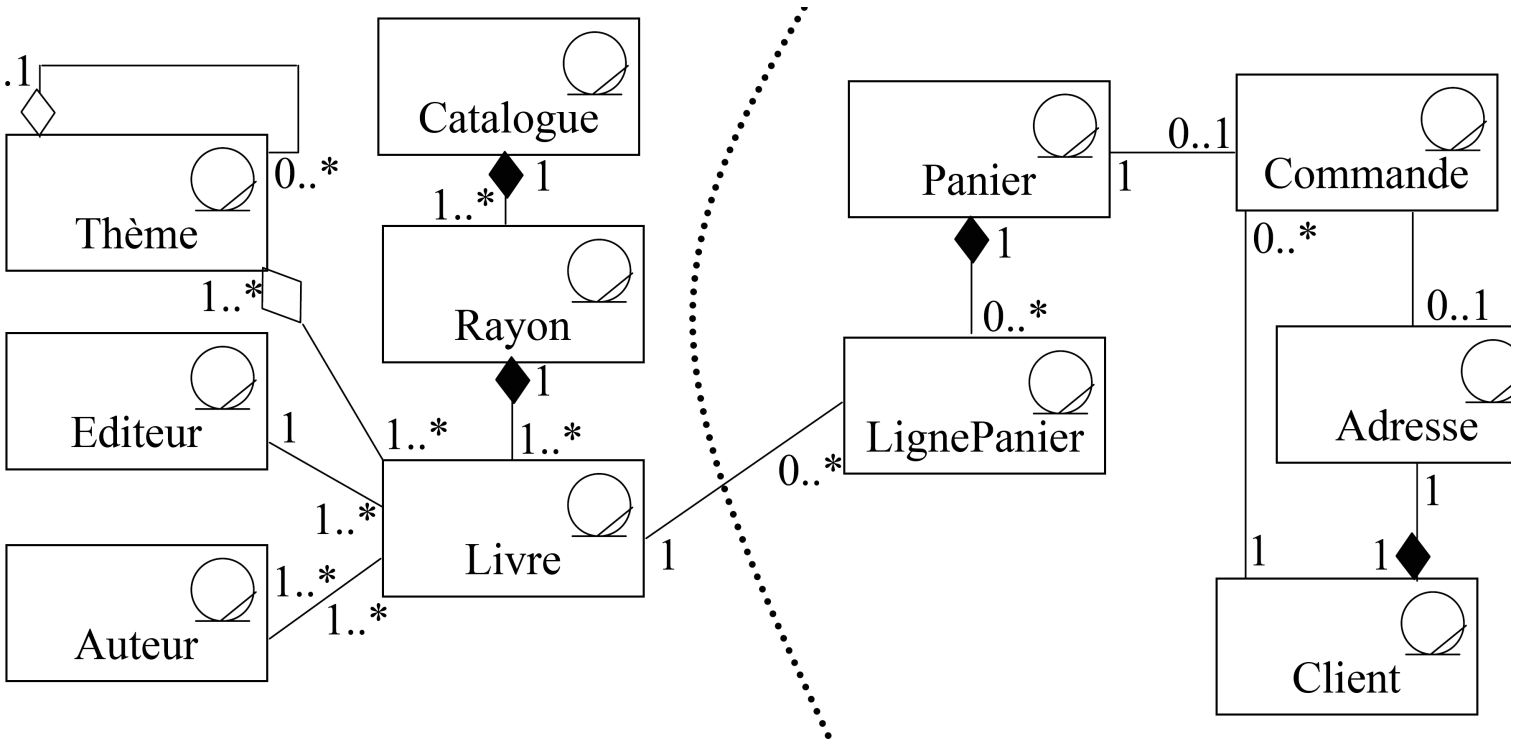
Le modèle du domaine est constitué d'un ensemble de classes dans lesquelles aucune opération n'est définie

- Le **modèle du domaine** décrit les **concepts invariants** du domaine d'application
  - **Exemple :** Pour un logiciel de gestions de factures, on aura des classes comme Produit, Client, Facture...
    - Peu importe que le logiciel soit en ligne ou non
    - Peu importe qu'on utilise php ou ajax
- **Etapes de la démarche :**
  1. Identifier les concepts du domaine
  2. Ajouter les associations et les attributs
  3. Généraliser les concepts
  4. Structurer en packages : structuration selon les principes de cohérence et d'indépendance.
- Les concepts du domaine peuvent être identifiés directement à partir de la connaissance du domaine ou par interview des experts métier.

Exemple de modèle du domaine



Structuration en packages



Définition synthétique des packages

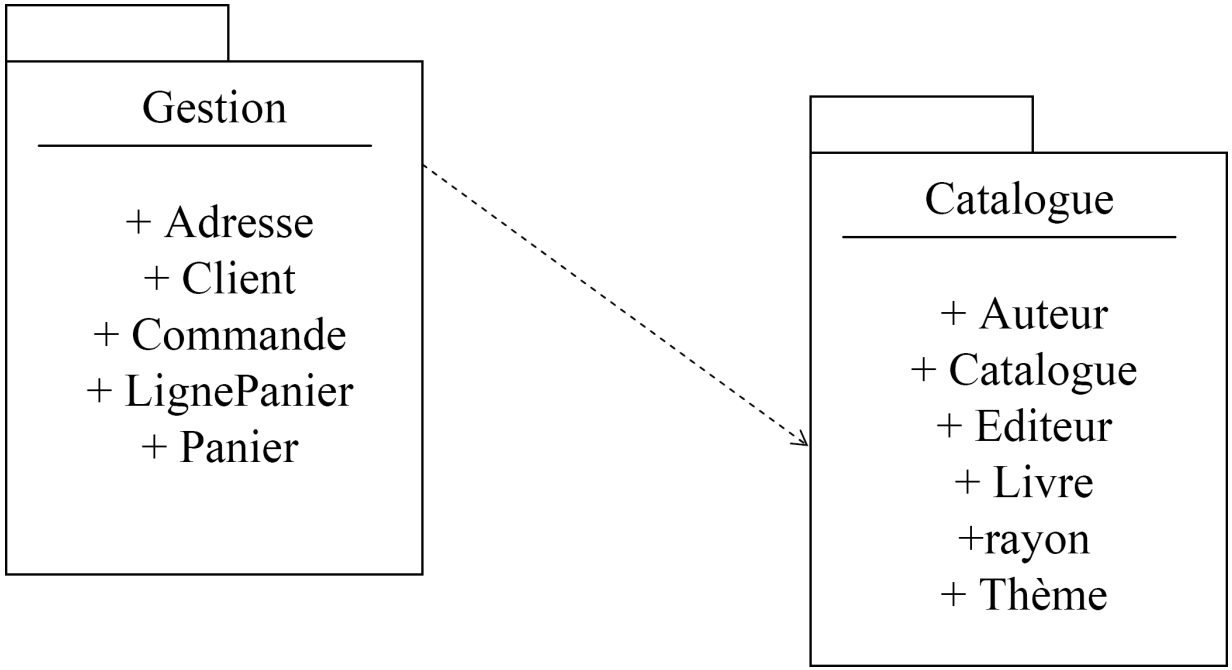
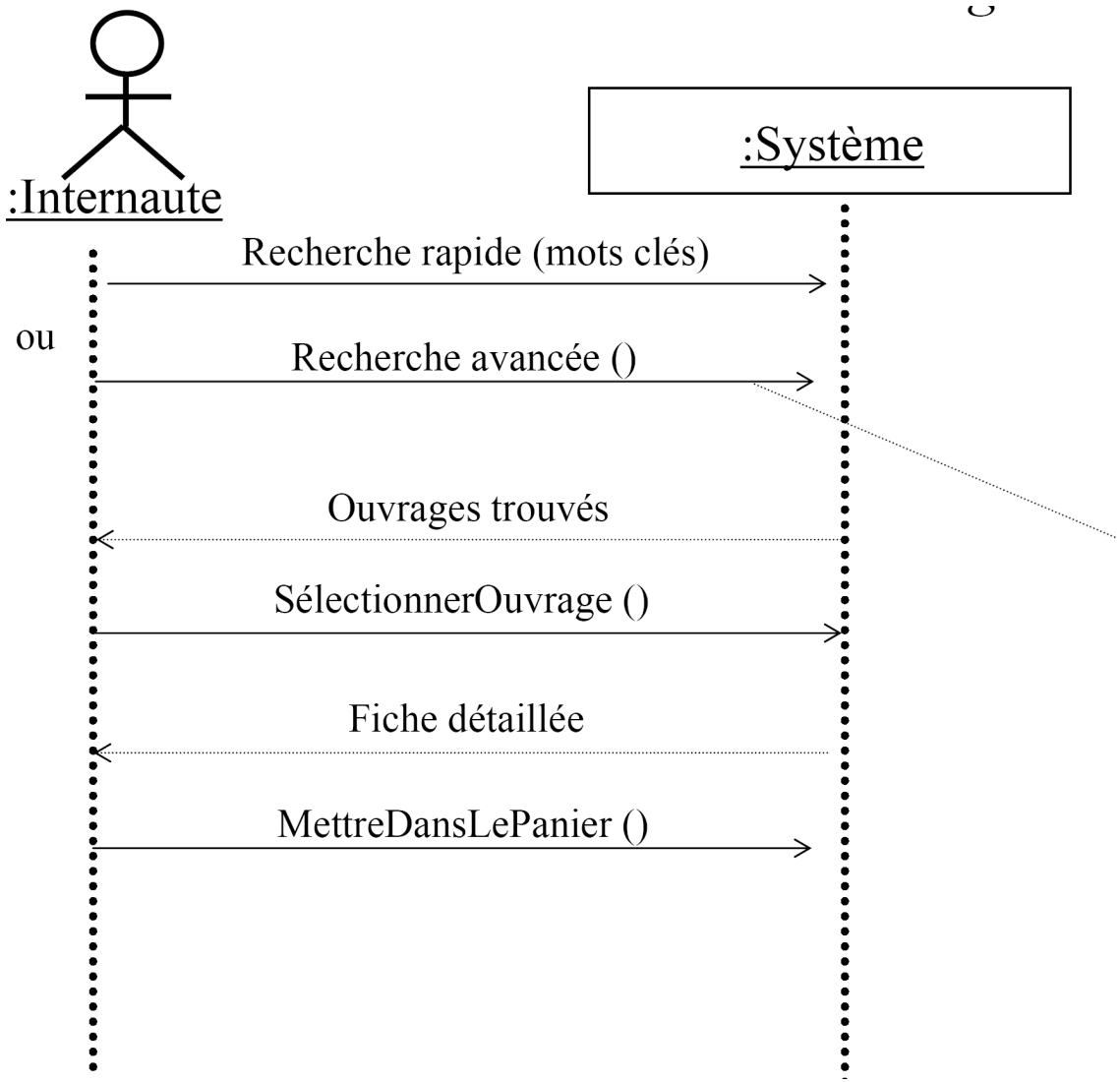


Diagramme de séquence système

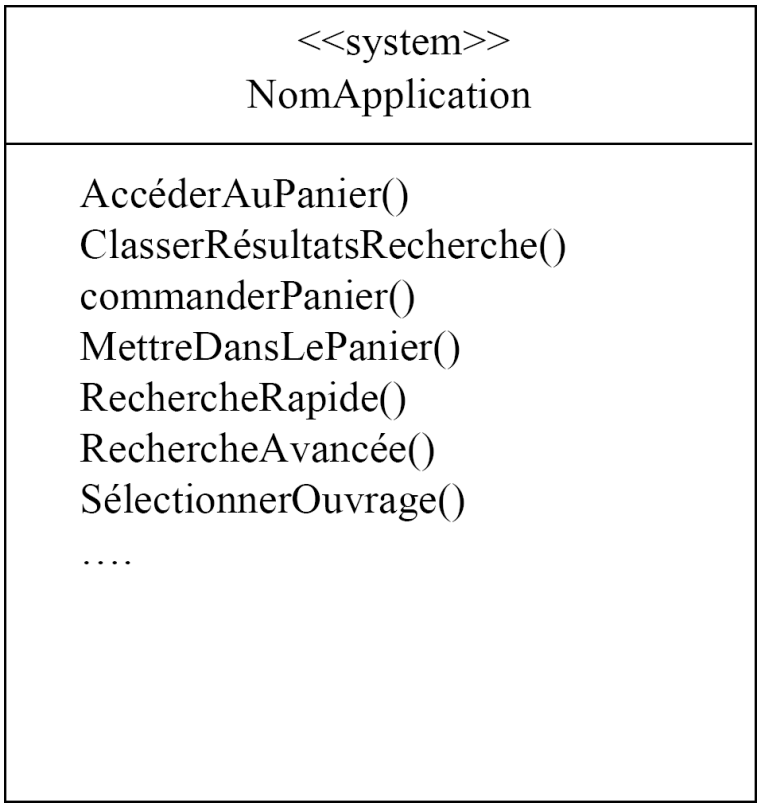
Un diagramme de séquence système est une formalisation des descriptions textuelles des cas d'utilisation

- **Un diagramme différent est produit pour chaque cas d'utilisation.**
- Construire un DSS implique la
  - Construction des diagrammes de séquence système
  - Mise à jour des cas d'utilisation (on peut réviser le diagramme de cas à la lumière des réflexions que nous inspirent la production des DSS)
  - Spécification des opérations système
- **Le système est considéré comme un tout**
  - On s'intéresse à ses interactions avec les acteurs
- Les diagrammes de séquence système sont parfois *très* simples mais ils seront enrichis par la suite

Exemple de diagramme de séquence système



Opérations système



- Les opérations système sont des opérations qui devront être réalisées par l'une ou l'autre des classes du système
- Elles correspondent à tous les messages qui viennent des acteurs vers le système dans les différents DSS

Classes d'analyse

- Réalisation des cas d'utilisation par les **classes d'analyse**
- Typologie des classes d'analyse
  - Les **classes dialogue** sont celles qui permettent les interactions entre les utilisateurs et l'application.
  - Les **classes contrôle** contiennent la dynamique de l'application
    - Elles font le lien entre les classes dialogue et les classes métier.
    - Elles permettent de contrôler la cinématique de l'application, l'ordre dans lequel les choses doivent se dérouler.
  - Les **classes métier** ou entités représentent les objets métier. Elles proviennent directement du modèle du domaine (mais peuvent être complétées en fonction des cas d'utilisation).

Les classes d'analyse sont les classes métier auxquelles on adjoint toutes les classes sui permettront au système de fonctionner : dialogues et contrôles

Diagramme de classes participantes

Le diagramme des classes participantes est un diagramme de classes décrivant les classes d'analyse et dans lequel on ajoute les acteurs

- A ce point du développement, seules les classes dialogue ont des **opérations** (actions de l'utilisateur sur l'IHM)
  - Ces opérations correspondent aux **opérations système**, c'est à dire aux messages entrants que seules les classes de dialogues sont habilitées à intercepter.
- **Associations** :
  - Les dialogues ne peuvent être reliés qu'aux contrôles ou à d'autres dialogues (en général, associations unidirectionnelles)
  - Les classes métier ne peuvent être reliées qu'aux contrôles ou à d'autres classes métier.
  - Les contrôles ont accès à tous les types de classes.

Exemples de classes participantes

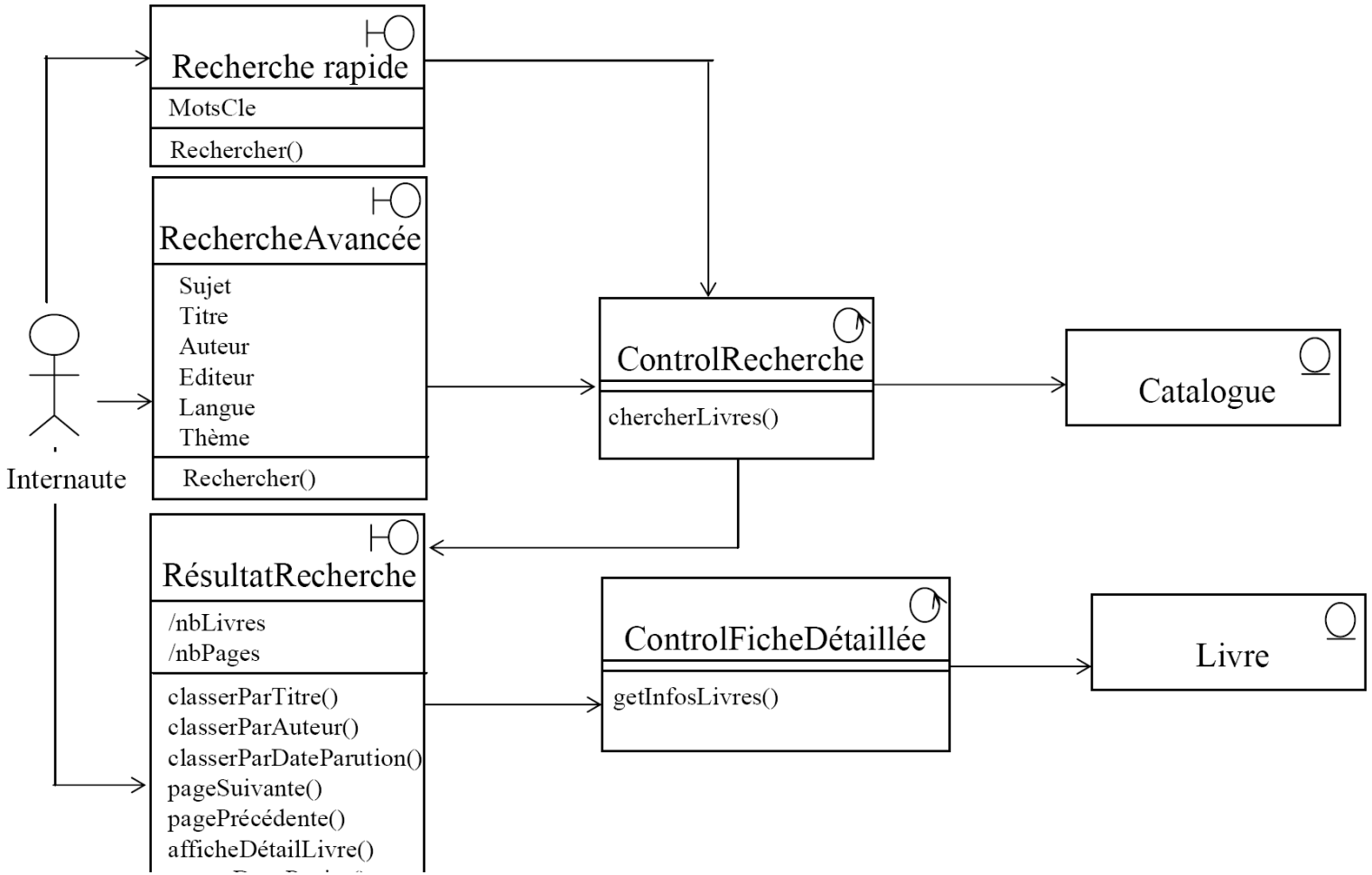
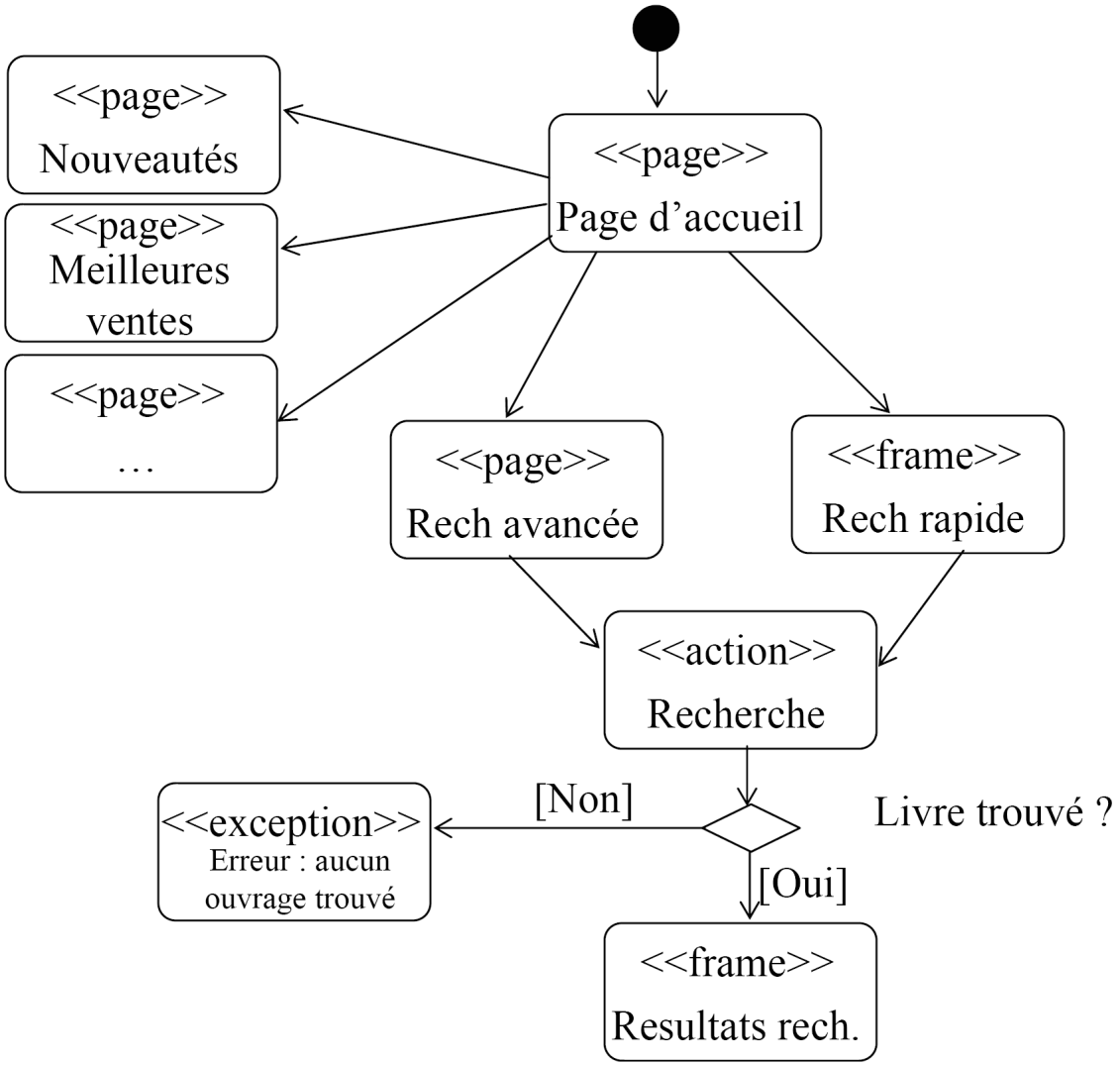


Diagramme d'activités de navigation

- **Modélisation de l'Interface Homme-Machine (IHM)** avec des diagrammes d'activité
  - Les activités peuvent représenter des écrans, des fenêtres de l'application, des pages php...
- Exploitation des maquettes de manière à représenter l'ensemble des chemins possibles entre les principaux écrans proposés à l'utilisateur.

Exemple de diagramme d'activités de navigation



Diagrammes d'interaction

- Dans les diagrammes de séquence système, le système était vu comme une boîte noire
  - Mais on sait maintenant de quels types d'objets est composé le système (diag. de classes participantes)
  - Le système n'est plus une boîte noire.
- **Chaque diagramme de séquence système donne lieu à un diagramme d'interaction**

Les **diagrammes d'interaction** montrent les interactions du système avec l'extérieur et les interactions internes qu'elles provoquent

- Les DSS sont repris mais l'objet « système » est éclaté pour donner le détail des classes d'analyse
  - **Les lignes de vie correspondent aux classes participantes**

Des séquences système aux interactions internes

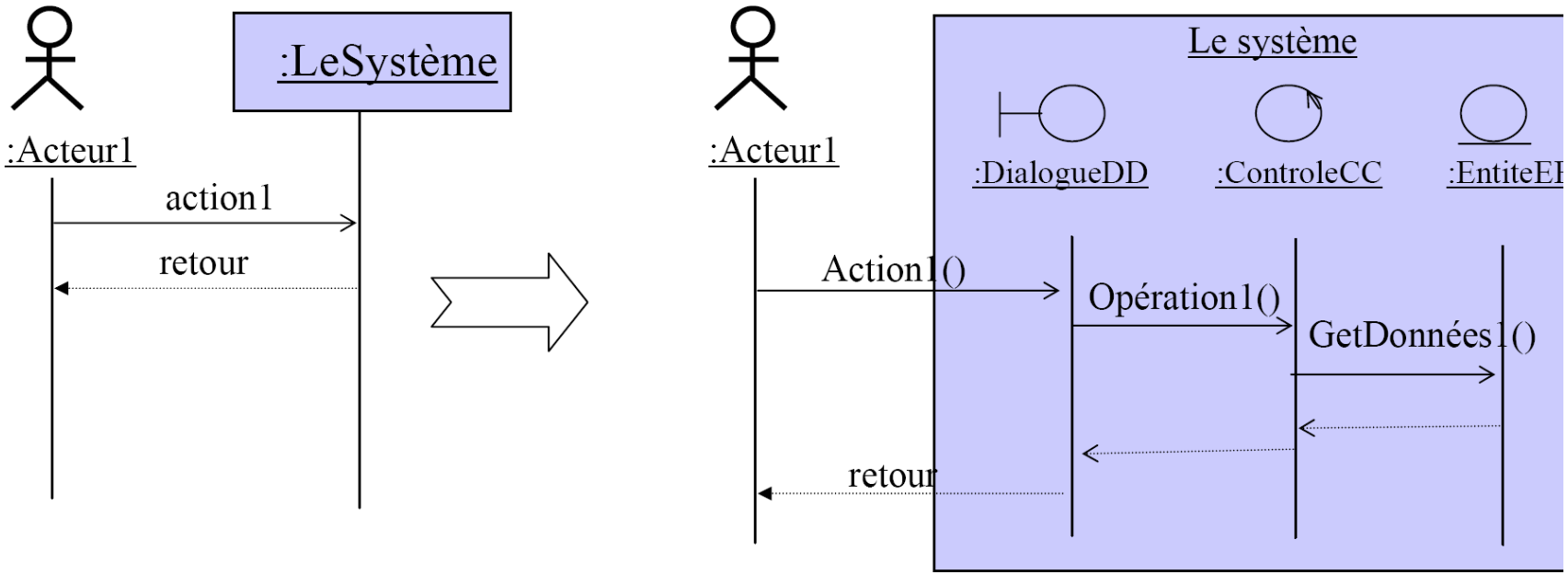


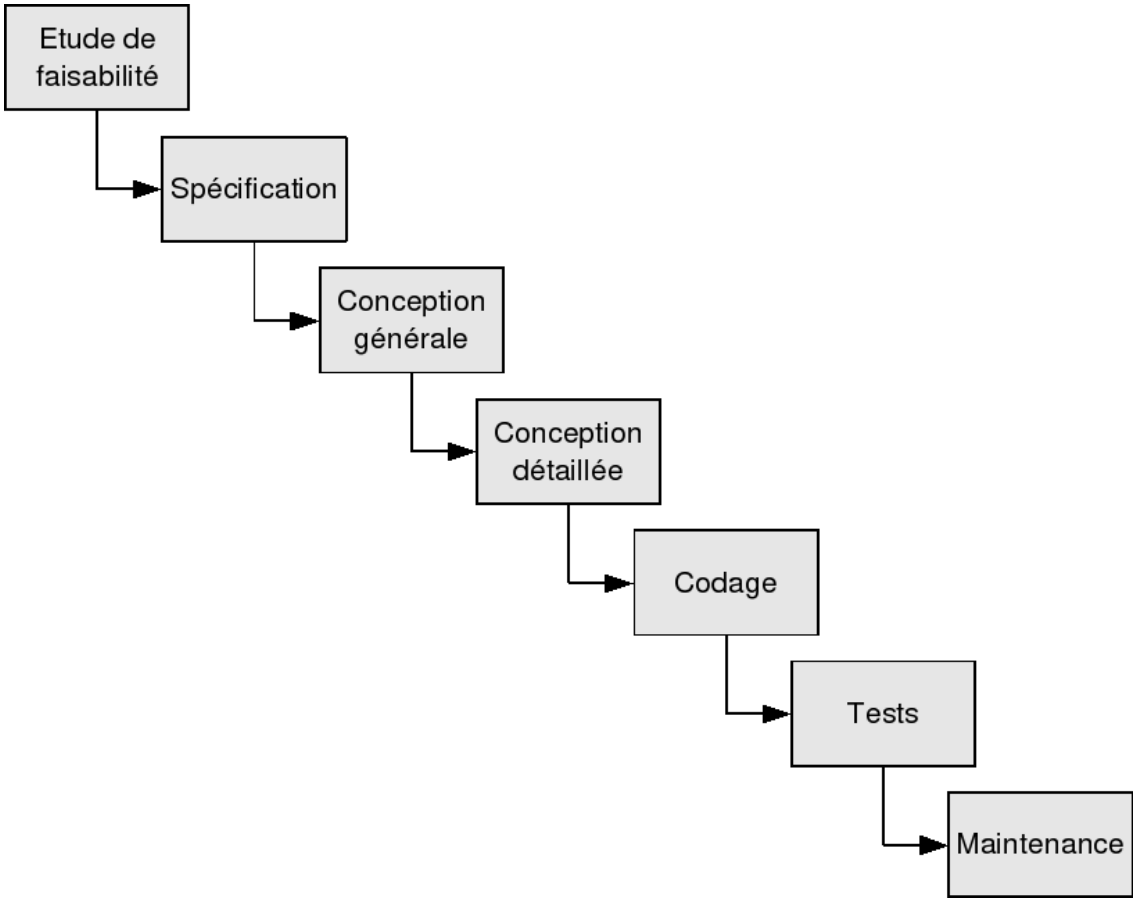
Diagramme des classes de conception

- **Enrichissement du diagramme de classes** pour
  - Prendre en compte l'architecture logicielle hôte
  - Modéliser les opération privées des différentes classes
  - Finaliser le modèle des classes avant l'implémentation
- On peut utiliser des diagrammes de séquence pour détailler :
  - Les interactions entre classes
  - Cartains algorithmes

Plan

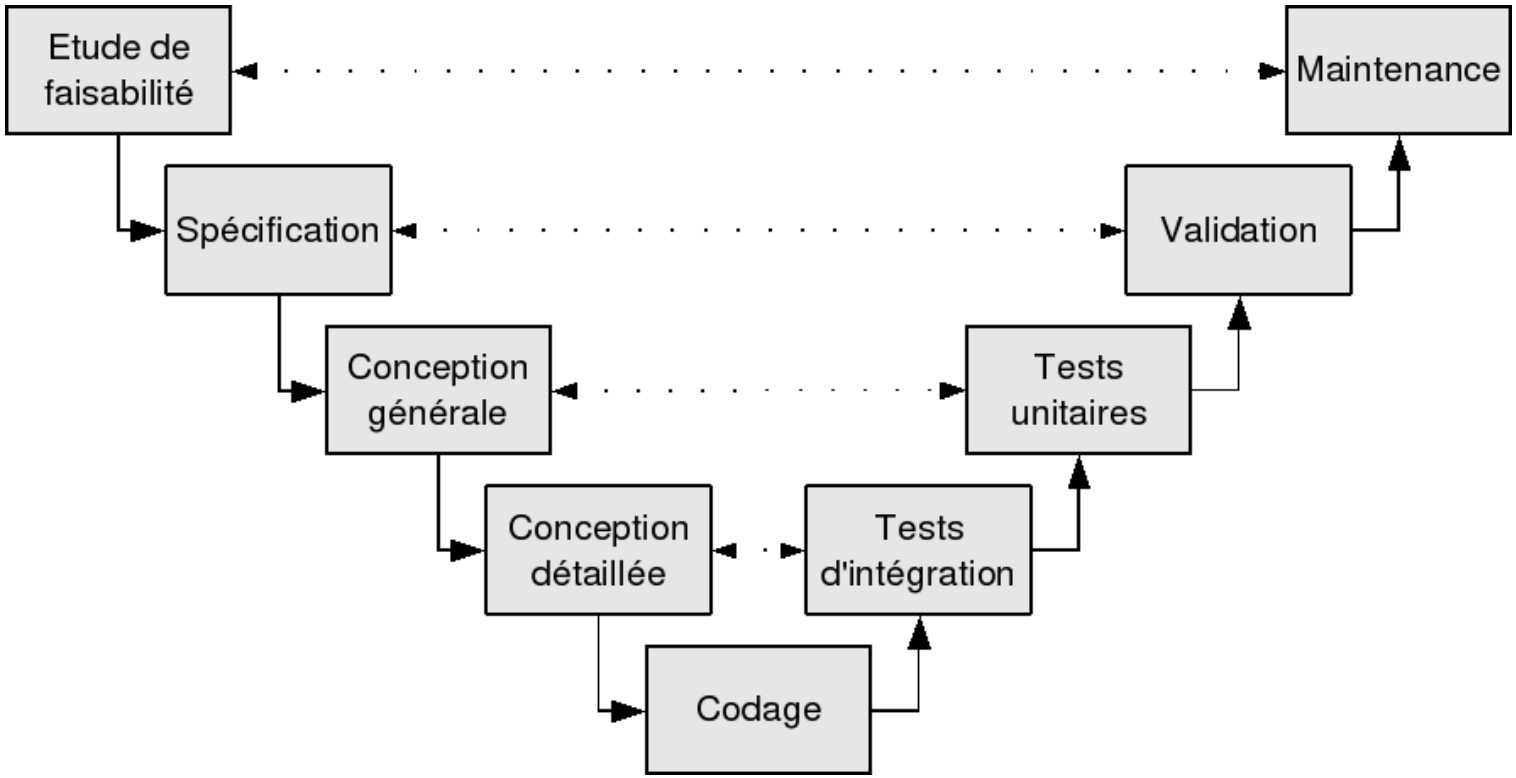
Modèles de cycles de vie linéaire

- Les phases du développement se suivent dans l'ordre et sans retour en arrière



Modèles de cycles de vie linéaire

- Les phases du développement se suivent dans l'ordre et sans retour en arrière



Problèmes des cycles linéaires

- Risques élevés et non contrôlés
  - Identification **tardive** des problèmes
  - Preuve **tardive** de bon fonctionnement
  - **« Effet tunnel**
- **Améliorations :** construction **itérative** du système
  - Chaque itération produit un nouvel **incrément**
  - Chaque nouvel incrément a pour objectif la maîtrise d'une partie des risques et apporte une preuve tangible de faisabilité ou d'adéquation
    - Enrichissement d'une série de prototypes
    - Les versions livrées correspondent à des étapes de la chaîne des prototypes

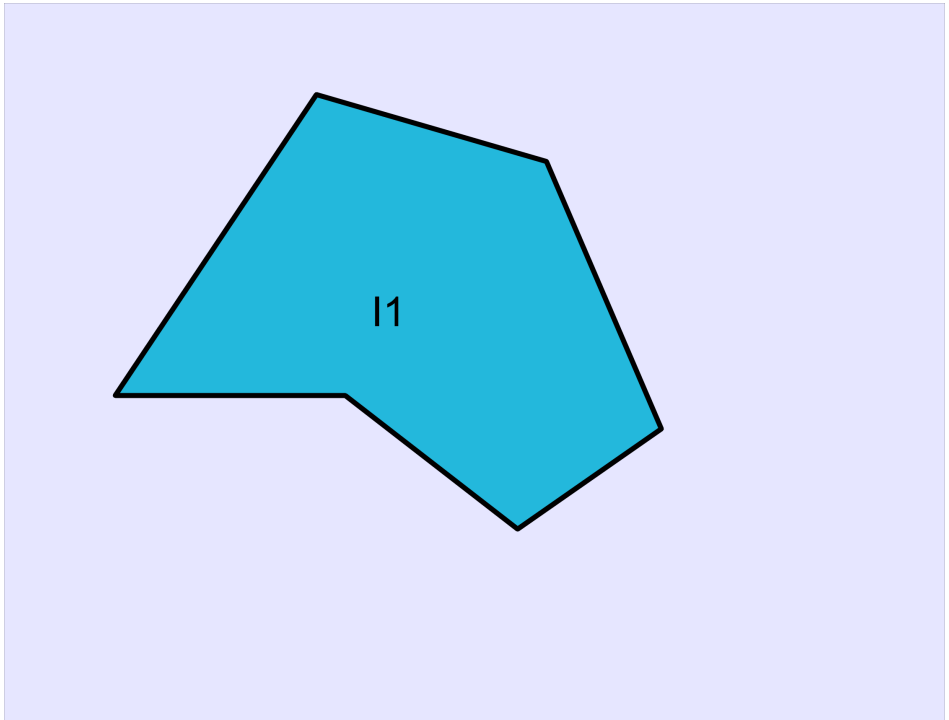
Production itérative d'incréments

- Itérations 0

- A chaque itération, on refait
  1. Spécification
  2. Conception
  3. Implémentation
  4. Tests

Production itérative d'incréments

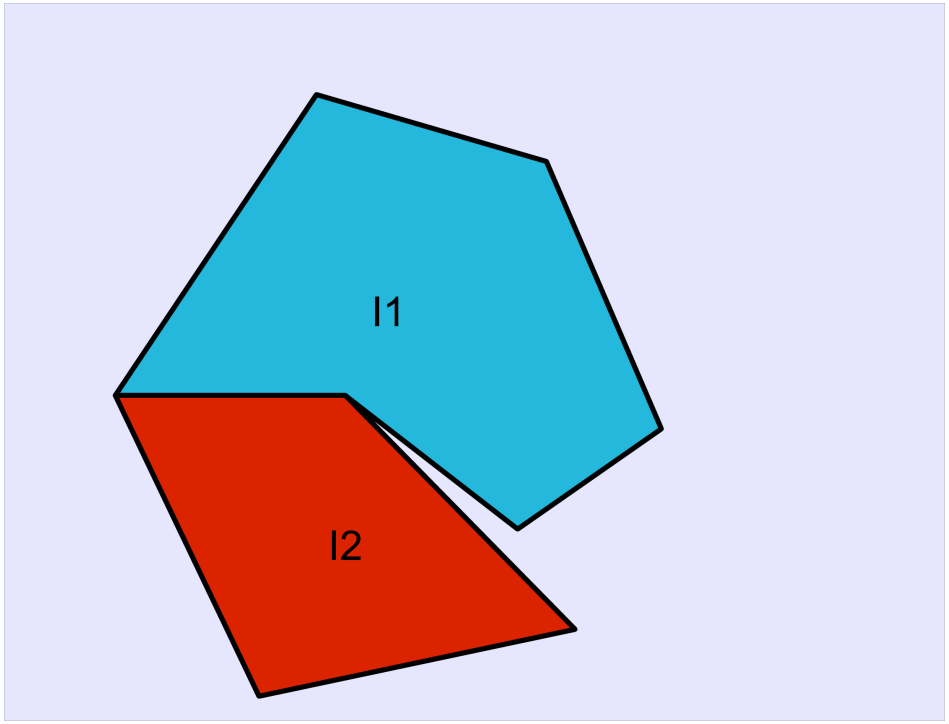
- Itérations 1



- A chaque itération, on refait
  1. Spécification
  2. Conception
  3. Implémentation
  4. Tests

Production itérative d'incréments

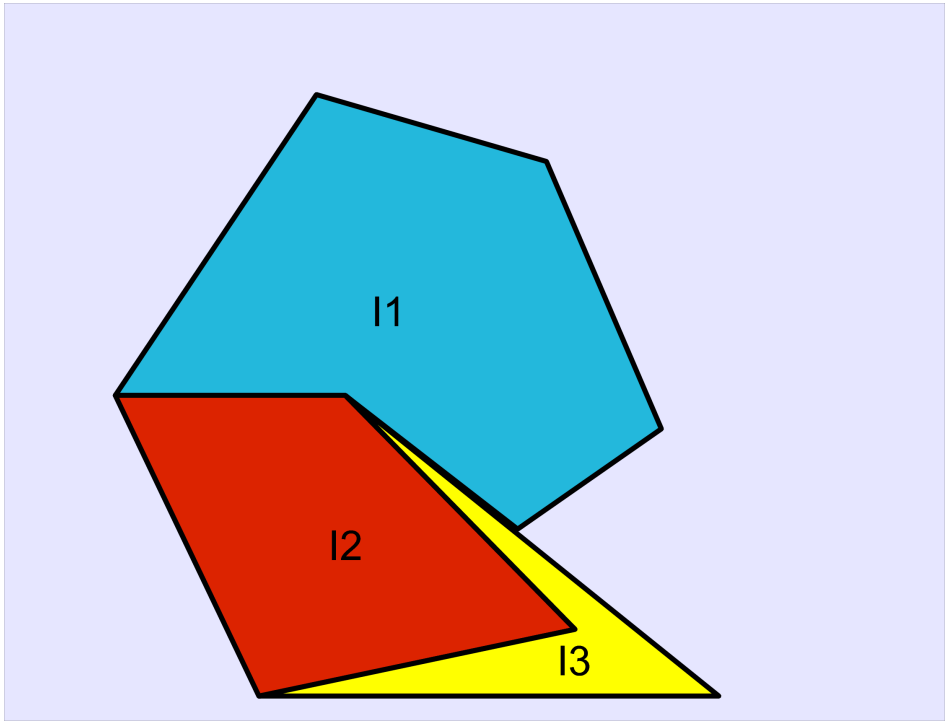
- Itérations 2



- A chaque itération, on refait
  1. Spécification
  2. Conception
  3. Implémentation
  4. Tests

Production itérative d'incréments

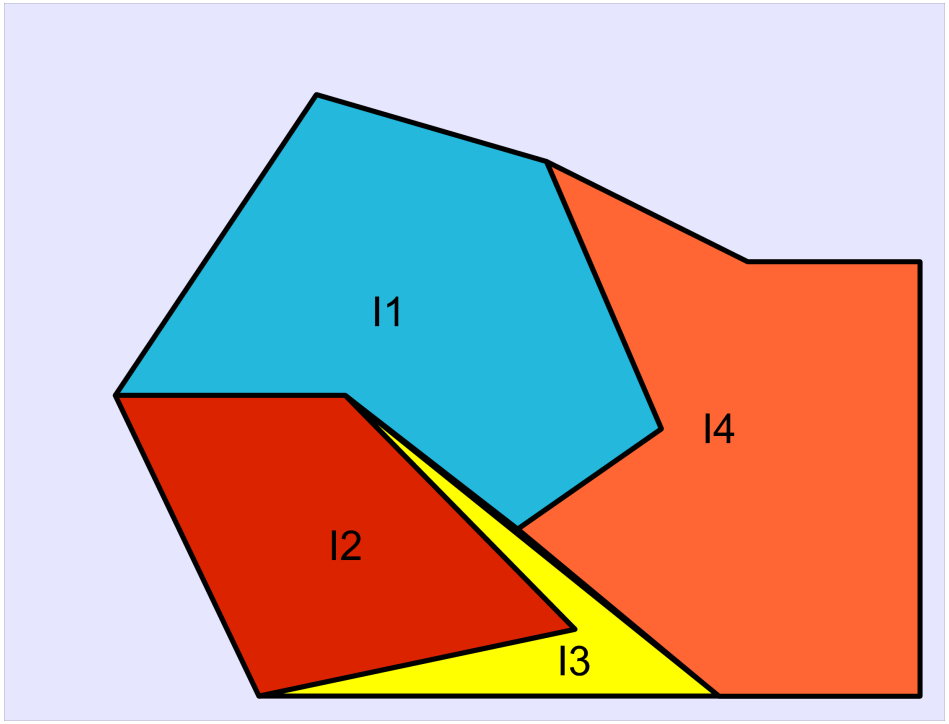
- Itérations 3



- A chaque itération, on refait
  1. Spécification
  2. Conception
  3. Implémentation
  4. Tests

Production itérative d'incréments

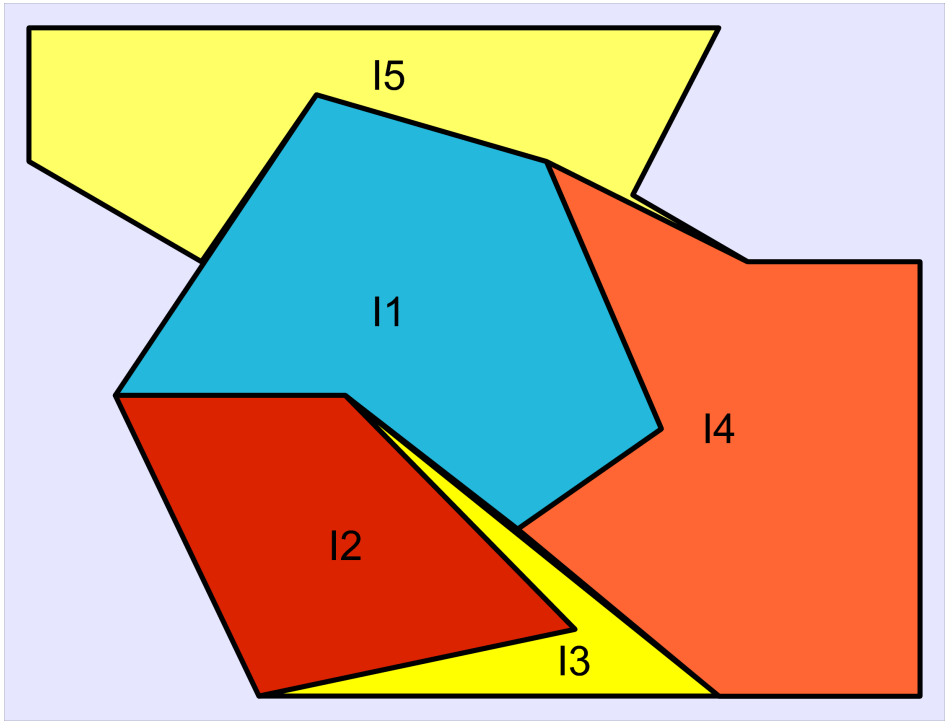
- Itérations 4



- A chaque itération, on refait
  1. Spécification
  2. Conception
  3. Implémentation
  4. Tests

Production itérative d'incréments

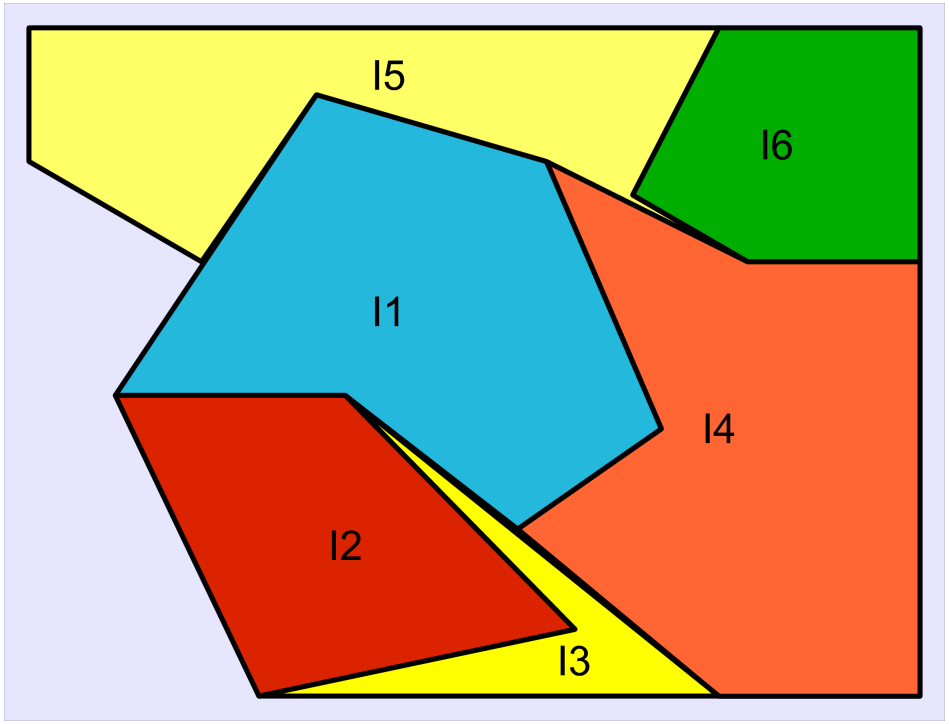
- Itérations 5



- A chaque itération, on refait
  1. Spécification
  2. Conception
  3. Implémentation
  4. Tests

Production itérative d'incréments

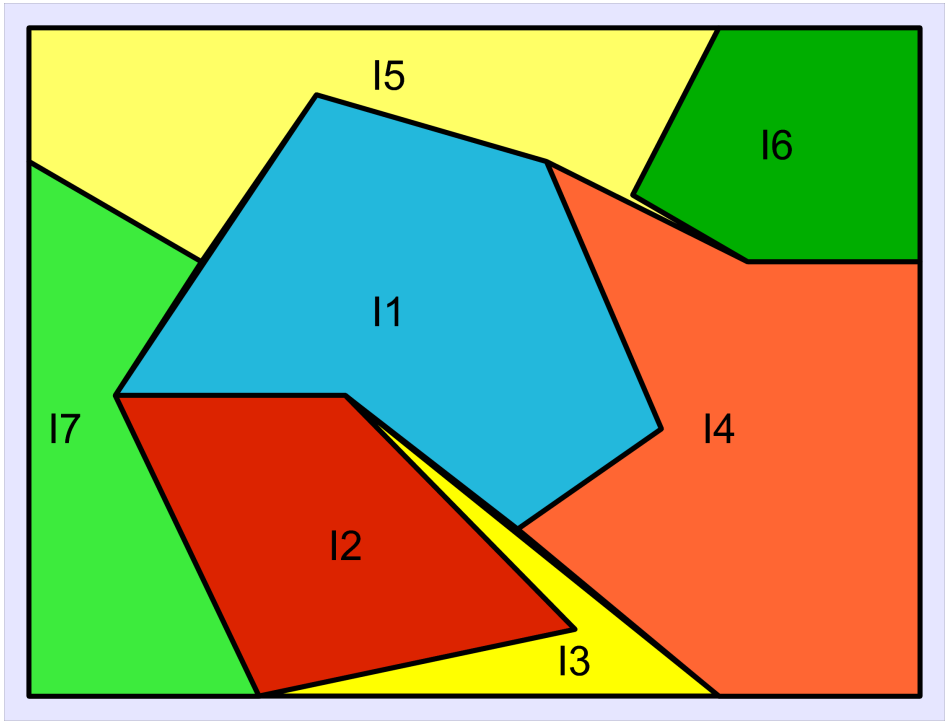
- Itérations 6



- A chaque itération, on refait
  1. Spécification
  2. Conception
  3. Implémentation
  4. Tests

Production itérative d'incréments

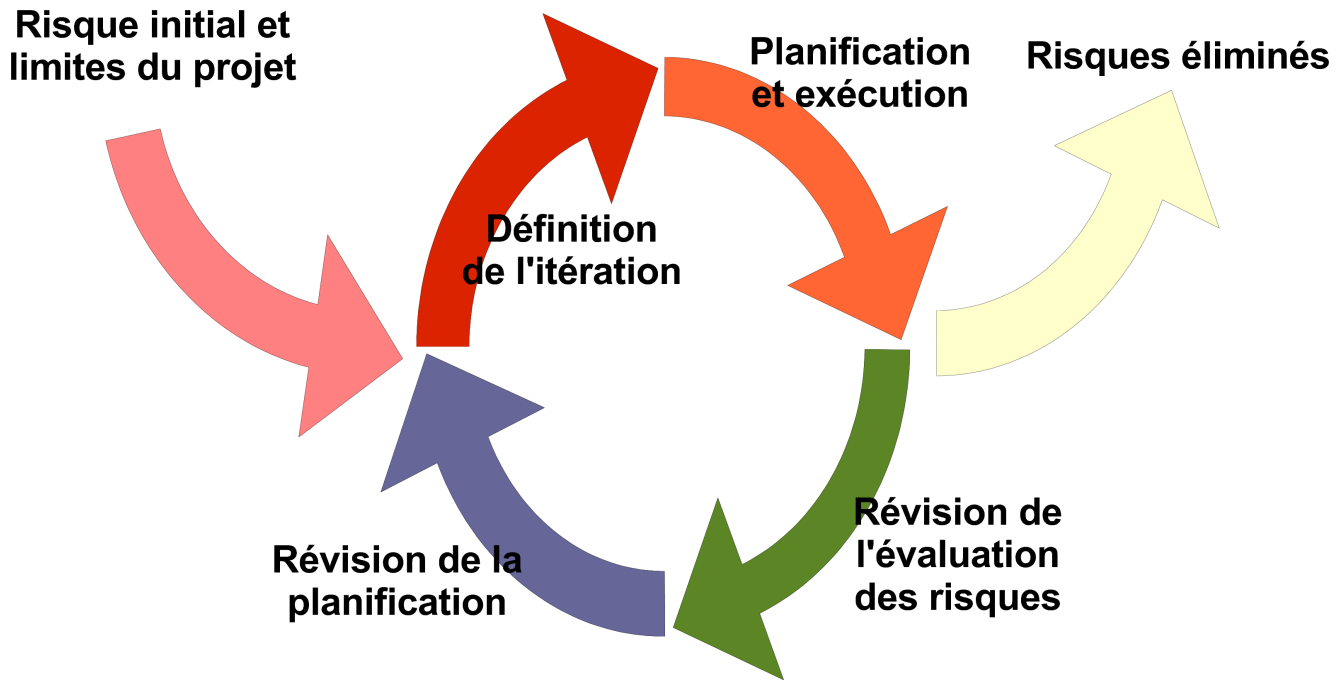
- Itérations 7



- A chaque itération, on refait
  1. Spécification
  2. Conception
  3. Implémentation
  4. Tests

Elimination des risques à chaque itération

On peut voir le développement d'un logiciel comme un processus graduel d'élimination de risques



- C'est pendant « **Planification et éxécution** » qu'on répète Spécification → Conception → Implémentation → Tests

Rational Unified Process

RUP est une démarche de développement qui est souvent utilisé conjointement au langage UML

- **Rational Unified Process** est
  - Piloté par les cas d'utilisation
  - Centré sur l'architecture
  - Itératif et incrémental

RUP est itératif et incrémental

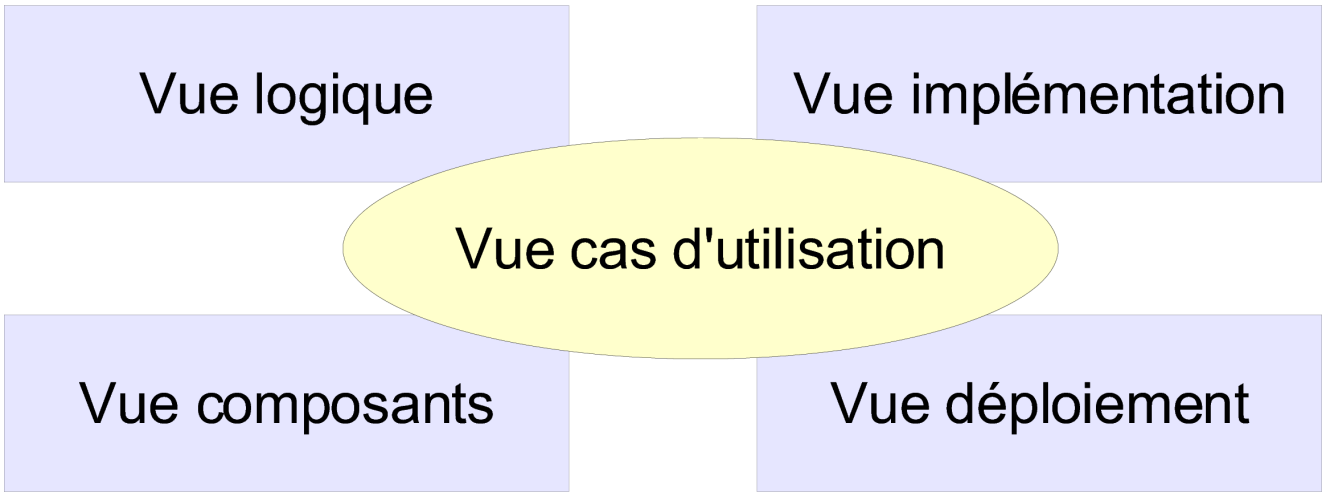
- Chaque itération prend en compte un certain nombre de cas d'utilisation
- Les risques majeurs sont traités en priorité
- Chaque itération donne lieu à un incrément et produit une nouvelle version exécutable

RUP est piloté par les cas d'utilisation

- La principale qualité d'un logiciel est son **utilité**
  - Adéquation du service rendu par le logiciel avec les besoins des utilisateurs
- Le développement d'un logiciel doit être centré sur l'utilisateur
- Les cas d'utilisation permettent d'exprimer ces besoins
  - Détection et description des besoins fonctionnels
  - Organisation des besoins fonctionnels

RUP est centré sur l'architecture

- Modélisation de différentes pespectives indépendantes et complémentaires
- **Architecture en couches et vues** de Krutchen



Vues du système

- **Vue cas d'utilisation**
  - Description du système comme un ensemble de transactions du point de vue de l'utilisateur
- **Vue logique**
  - Créée lors de la phase d'élaboration et raffinée lors de la phase de construction
  - Utilisation de diagrammes de classes, de séquences...
- **Vue composants**
  - Description de l'architecture logicielle
- **Vue déploiement**
  - Description de l'architecture matérielle du système
- **Vue implémentation**
  - Description des algorithmes, code source

Organisation en phases du développement

- **Initialisation**
  - Définition du problème
- **Elaboration**
  - Planification des activités, affectation des ressources, analyse
- **Construction**
  - Développement du logiciel par incréments successifs
- **Transition**
  - Recettage et déploiement

Les phases du développement sont les grandes étapes du développement du logiciel

- Le projet commence en phase d'initialisation et termine en phase de transition

Phase d'initialisation : Objectifs

- Définition du cadre du projet, son concept, et inventaire du contenu
- Elaboration des cas d'utilisation critiques ayant le plus d'influence sur l'architecture et la conception
- Réalisation d'un ou de plusieurs prototypes démontrant les fonctionnalités décrites par les cas d'utilisation principaux
- Estimation détaillée de la charge de travail, du coût et du planning général ainsi que de la phase suivante d'élaboration Estimation des risques

Phase d'initialisation : Activités

- Formulation du cadre du projet, des besoins, des contraintes et des critères d'acceptation
- Planification et préparation de la justification économique du projet et évaluation des alternatives en termes de gestion des risques, ressources, planification
- Synthèse des architectures candidates, évaluation des coûts

Phase d'initialisation : Livrables

- Un document de vision présentant les besoins de base, les contraintes et fonctionnalités principales
- Une première version du modèle de cas d'utilisation
- Un glossaire de projet
- Un document de justification économique incluant le contexte général de réalisation, les facteurs de succès et la prévision financière
- Une évaluation des risques
- Un plan de projet présentant phases et itérations
- Un ou plusieurs prototypes

Phase d'initialisation : Critères d'évaluation

- Un consensus sur
  - la planification,
  - les coûts
  - la définition de l'ensemble des projets des parties concernées
- La compréhension commune des besoins

Phase d'élaboration : objectifs

- Définir, valider et arrêter l'architecture
- Démontrer l'efficacité de cette architecture à répondre à notre besoin
- Planifier la phase de construction

Phase d'élaboration : activités

- Elaboration de la vision générale du système, les cas d'utilisation principaux sont compris et validés
- Le processus de projet, l'infrastructure, les outils et l'environnement de développement sont établis et mis en place
- Elaboration de l'architecture et sélection des composants

Phase d'élaboration : livrables

- Le modèle de cas d'utilisation est produit au moins à 80 %
- La liste des exigences et contraintes non fonctionnelles identifiées
- Une description de l'architecture
- Un exécutable permettant de valider l'architecture du logiciel à travers certaines fonctionnalités complexes
- La liste des risques revue et la mise à jour de la justification économique du projet
- Le plan de réalisation, y compris un plan de développement présentant les phases, les itérations et les critères d'évaluation

Phase d'élaboration : Critères d'évaluation

- La stabilité de la vision du produit final
- La stabilité de l'architecture
- La prise en charge des risques principaux est adressée par le(s) prototype(s)
- La définition et le détail du plan de projet pour la phase de construction
- Un consensus, par toutes les parties prenantes, sur la réactualisation de la planification, des coûts et de la définition de projet

Phase de construction : objectifs

- La minimisation des coûts de développement par
  - l'optimisation des ressources
  - la minimisation des travaux non nécessaires
- Le maintien de la qualité
- Réalisation des versions exécutables

Phase de construction : Activités

- La gestion et le contrôle des ressources et l'optimisation du processus de projet
- Evaluation des versions produites en regard des critères d'acceptation définis

Phase de construction : Livrables

- Les versions exécutables du logiciel correspondant à l'enrichissement itération par itération des fonctionnalités
- Les manuels d'utilisation réalisés en parallèle à la livraison incrémentale des exécutables
- Une description des versions produites

Phase de construction : Critères d'évaluation

- La stabilité et la qualité des exécutables
- La préparation des parties prenantes
- La situation financière du projet en regard du budget initial

Phase de transition : Objectifs

- Le déploiement du logiciel dans l'environnement d'exploitation des utilisateurs
- La prise en charge des problèmes liés à la transition
- Atteindre un niveau de stabilité tel que l'utilisateur est indépendant
- Atteindre un niveau de stabilité et qualité tel que les parties prenantes considèrent le projet comme terminé

Phase de transition : Activités

- Activités de « packaging » du logiciel pour le mettre à disposition des utilisateurs et de l'équipe d'exploitation
- Correction des erreurs résiduelles et amélioration de la performance et du champ d'utilisation
- Evaluation du produit final en regard des critères d'acceptation définis

Phase de transition : Livrables

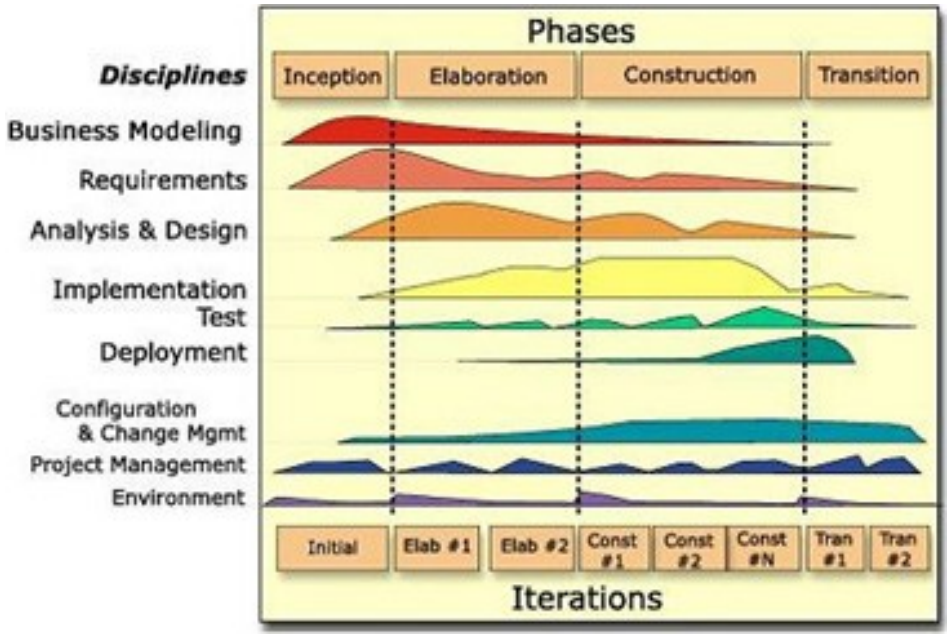
- La version finale du logiciel
- Les manuels d'utilisation

Phase de transition : Critères d'évaluation

- La satisfaction des utilisateurs
- La situation financière du projet en regard du budget initial

Organisation en activités de développement

- Chaque phase comprend plusieurs itérations
- Pour chacune des itérations, on se livre à plusieurs activités
  - Modélisation métier
  - **Expression des besoins**
  - **Analyse**
  - **Conception**
  - **Implémentation**
  - **Test**
  - Dépoloiement



- Les **activités** sont des étapes dans le développement d'un logiciel, mais à un niveau de granularité beaucoup plus fin que les phases
- Chaque activité est répétée autant de fois qu'il y a d'itérations

Modélisation métier

- **Objectif :** Mieux comprendre la structure et la dynamique de l'organisation.
  - Proposer la meilleure solution dans le contexte de l'organisation cliente.
  - Réalisation d'un glossaire des termes métiers.
  - Cartographie des processus métier de l'organisation cliente.
- Activité coûteuse mais qui permet d'accélérer la compréhension d'un problème

Expression des besoins

- **Objectif :** Cibler les besoins des utilisateurs et du clients grâce à une série d'interviews.
  - L'ensemble des parties prenantes du projet, maîtrise d'oeuvre et maîtrise d'ouvrage, est acteur de cette activité.
- L'activité de recueil et d'expression des besoins débouche sur ce que doit faire le système (question « **QUOI ?** »)

Expression des besoins

- Utilisation des **cas d'utilisation** pour
  - Schématiser les besoins
  - Structurer les documents de spécifications fonctionnelles.
- Les cas d'utilisation sont décomposés en scénarios d'usage du système, dans lesquels l'utilisateur « raconte » ce qu'il fait grâce au système et ses interactions avec le système.
- Un maquettage est réalisable pour mieux « immerger » l'utilisateur dans le futur système.
- Une fois posées les limites fonctionnelles, le projet est planifié et une prévision des coûts est réalisée.

Analyse

- **Objectif :** Transformer les besoins utilisateurs en modèles UML
  - Analyse objet servant de base à une réflexion sur les mécanismes internes du système
- Principaux livrables
  - Modèles d'analyse, neutre vis à vis d'une technologie.
  - Livre une spécification plus précise des besoins
- Peut envisagé comme une première ébauche du modèle de conception

Conception

- **Objectif :** Modéliser **comment** le système va fonctionner
  - Exigences non fonctionnelles
  - Choix technologiques.
- Le système est analysé et on produit
  - Une proposition d'architecture.
  - Un découpage en composants.

Impémentation

- **Objectif :** Implémenter le système par composants.
  - Le système est développé par morceaux dépendant les uns des autres.
  - Optimisation de l'utilisation des ressources selon leurs expertises.
- Les découpages fonctionnel et en couches sont indispensable pour cette activité.
- Il est tout à fait envisageable de retoucher les modèles d'analyse et de conception à ce stade.

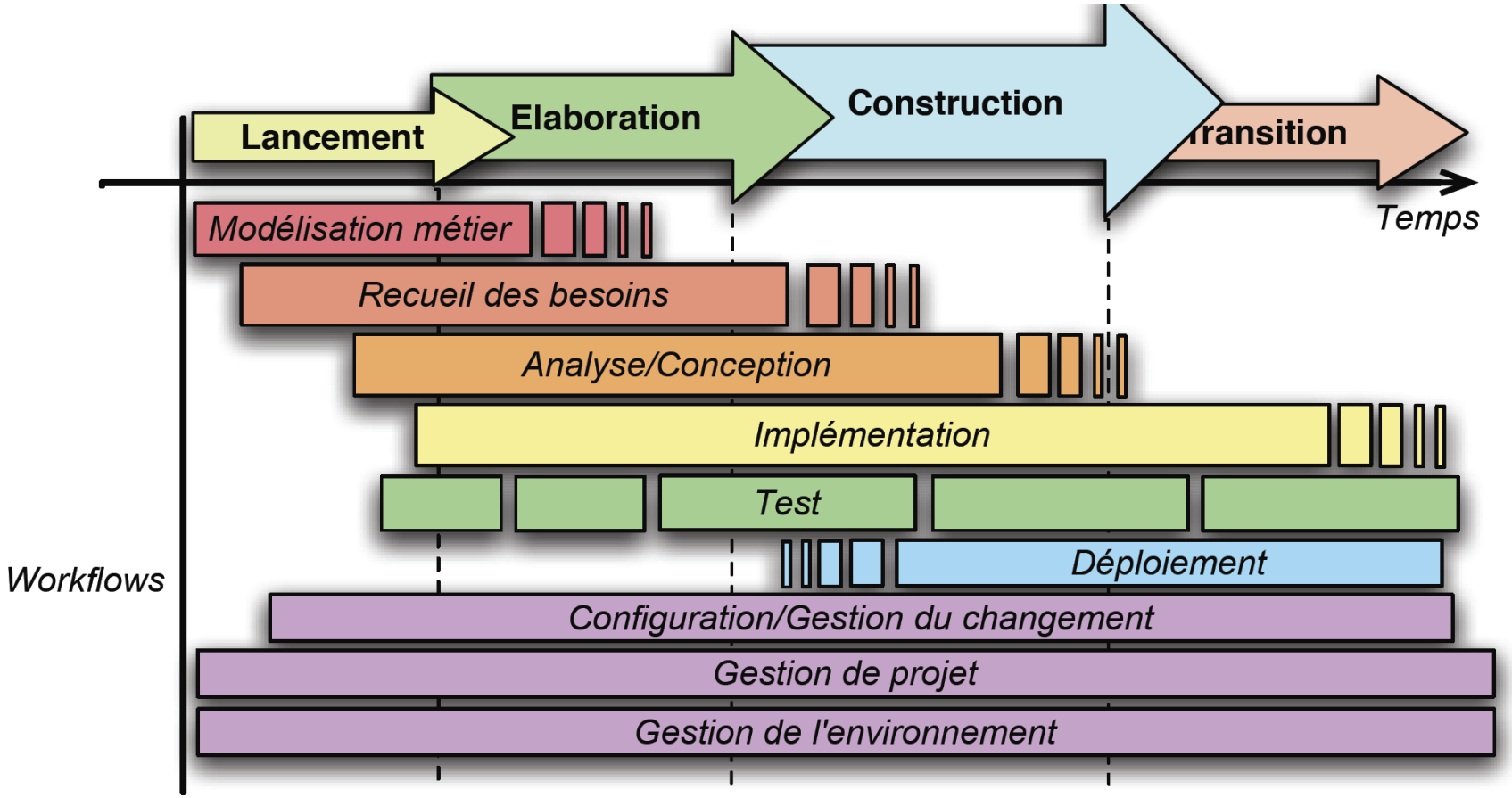
Test

- **Objectif :** Vérifier des résultats de l'implémentation en testant la construction.
  - **Tests unitaires :** tests composants par composants
  - **Tests d'intégration :** tests de l'interaction de composants préalablement testés individuellement
- Méthode :
  - Planification pour chaque itération
  - Implémentation des tests en créant des cas de tests
  - Exécuter les tests
  - Prendre en compte le résultat de chacun.

Déploiement

- **Objectif :** Déployer les développements une fois réalisés.
  - Peut être réalisé très tôt dans le processus dans une sousactivité de prototypage dont l'objectif est de valider
    - l'architecture physique
    - les choix technologiques.

Importance des activités dans chaque phase



Principaux diagrammes UML par activité

- **Expression des besoins et modélisation métier**
  - Modèles métier, domaine, cas d'utilisation
  - Diagramme de séquences
  - Diagramme d'activité
- **Analyse**
  - Modèles métier, cas d'utilisation
  - Diagramme des classes, de séquences et de déploiement
- **Conception**
  - Diagramme des classes, de séquences
  - Diagramme état/transition
  - Diagramme d'activité
  - Diagramme de déploiement et de composant

2TUP, une variante du « Unified Process »

- **2TUP**, avec un processus de développement en « Y », développé par Valtech
  - **« UML 2.0, en action : De l'analyse des besoins à la conception J2EE »**
    - **Pascal Roques, Franck Vallée**
    - Editions Eyrolles (2004)

Plan

Méthodes agiles

« Quelles activités pouvons nous abandonner tout en produisant des logiciels de qualité ? »

« Comment mieux travailler avec le client pour nous focaliser sur ses besoins les plus prioritaires et être aussi réactifs que possible ? »

- **Filiation avec le RAD**
- Exemples de méthodes agiles
  - **XP** (eXtreme Programming), **DSDM** (Dynamic Software Development Method), **ASD** (Adaptative Software Development), **CCM** (Crystal Clear Methodologies), **SCRUM**, **FDD** (Feature Driven Development)

Priorités des méthodes agiles

- Priorité **aux personnes et aux interactions** sur les procédures de les outils
- Priorité **aux applications fonctionnelles** sur une documentation pléthorique
- Priorité **à la collaboration avec le client** sur la négociation de contrat
- Priorité **à l'acceptation du changement** sur la planification

eXtreme Programming

eXtreme Programming, une méthode « basée sur des pratiques qui sont autant de boutons poussés au maximum »

- Méthode qui peut sembler naturelle mais **concrètement difficile à appliquer et à maîtriser**
  - Réclame beaucoup de discipline et de communication (contrairement à la première impression qui peut faire penser à une ébullition de cerveaux individuels).
  - Aller vite mais sans perdre de vue la rigueur du codage et les fonctions finales de l'application.
- **Force de XP :** sa simplicité et le fait qu'on va droit à l'essentiel, selon un rythme qui doit rester constant.

Valeurs d'XP

- **Communication**
  - XP favorise la communication directe, plutôt que le cloisonnement des activités et les échanges de documents formels.
  - Les développeurs travaillent directement avec la maîtrise d'ouvrage
- **Feedback**
  - Les pratiques XP sont conçues pour donner un maximum de feedback sur le déroulement du projet afin de corriger la trajectoire au plus tôt.
- **Simplicité :**
  - Du processus
  - Du code
- **Courage :**
  - d'honorer les autres valeurs
  - de maintenir une communication franche et ouverte
  - d'accepter et de traiter de front les mauvaises nouvelles.

Pratiques d'XP

- XP est fondé sur des valeurs, mais surtout sur **13 pratiques** réparties en 3 catégories
  - Gestion de projets
  - Programmation
  - Collaboration

Pratiques de gestion de projets

- **Livraisons fréquentes**
  - L'équipe vise la mise en production rapide d'une version minimale du logiciel, puis elle fournit ensuite régulièrement de nouvelles livraisons en tenant compte des retours du client.
- **Planification itérative**
  - Un plan de développement est préparé au début du projet, puis il est revu et remanié tout au long du développement pour tenir compte de l'expérience acquise par le client et l'équipe de développement.
- **Client sur site**
  - Le client est intégré à l'équipe de développement pour répondre aux questions des développeurs et définir les tests fonctionnels.
- **Rythme durable**
  - L'équipe adopte un rythme de travail qui lui permet de fournir un travail de qualité tout au long du projet.
  - Jamais plus de 40h de travail par semaine (un développeur fatigué développe mal)

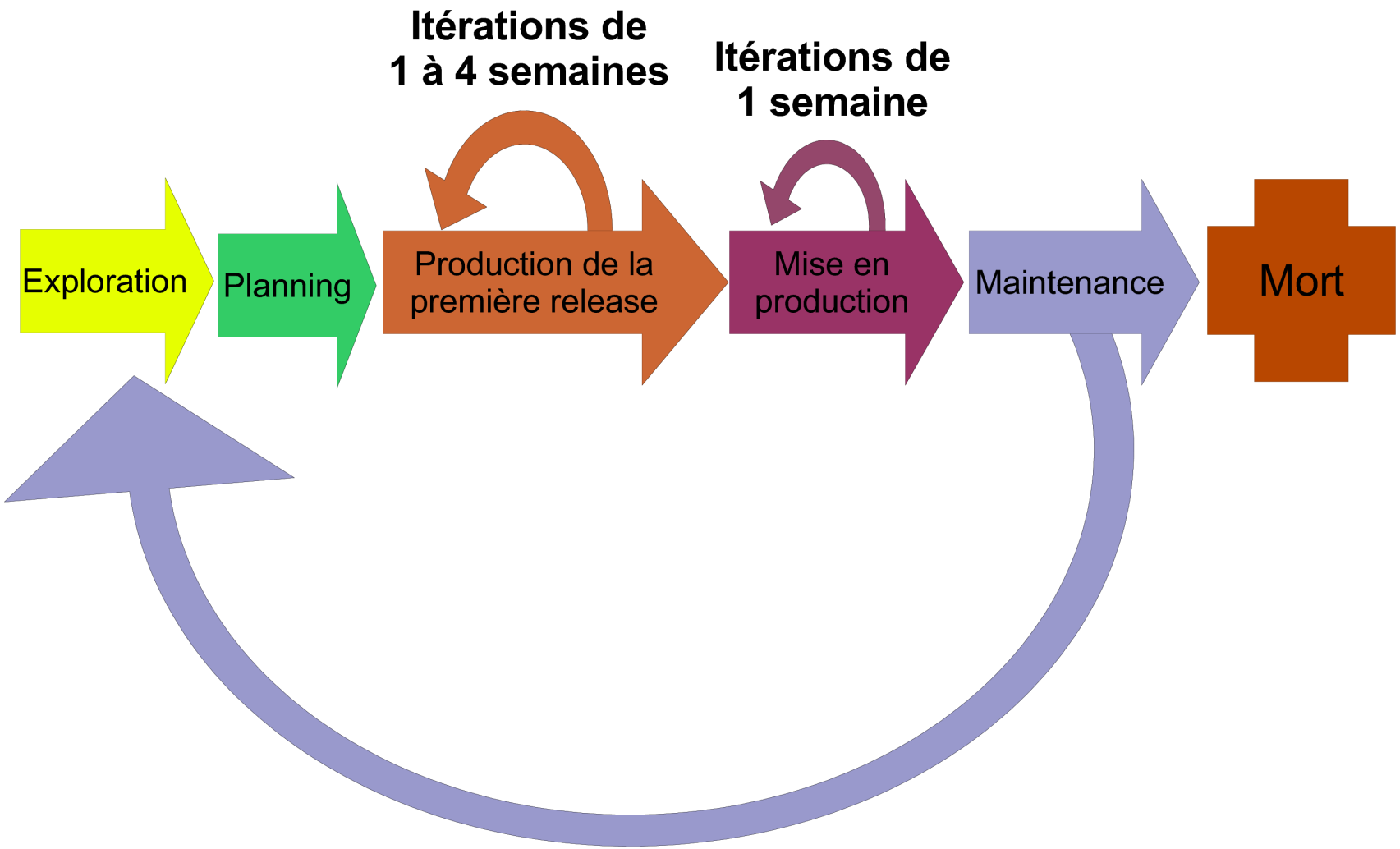
Pratiques de programmation

- **Conception simple**
  - On ne développe rien qui ne soit utile tout de suite.
- **Remaniement**
  - Le code est en permanence réorganisé pour rester aussi clair et simple que possible.
- **Tests unitaires**
  - Les développeurs mettent en place une batterie de tests de nonrégression qui leur permettent de faire des modifications sans crainte.
- **Tests de recette**
  - Les testeurs mettent en place des tests automatiques qui vérifient que le logiciel répond aux exigences du client.
  - Ces tests permettent des recettes automatiques du logiciel.

Pratiques de collaboration

- **Responsabilité collective du code**
  - Chaque développeur est susceptible de travailler sur n'importe quelle partie de l'application.
- **Programmation en binômes**
  - Les développeurs travaillent toujours en binômes, ces binômes étant renouvelés fréquemment.
- **Règles de codage**
  - Les développeurs se plient à des règles de codage strictes définies par l'équipe elle-même.
- **Métaphore**
  - Les développeurs s'appuient sur une description commune du design.
- **Intégration continue**
  - L'intégration des nouveaux développements est faite chaque jour.

Cycle de vie XP



Exploration

- Les développeurs se penchent sur des **questions techniques**
  - Explorer les différentes possibilités d'architecture pour le système
  - Etudier par exemple les limites au niveau des performances présentées par chacune des solutions possibles
- Le client s'habitue à **exprimer ses besoins** sous forme de user strories (proches de diagrammes de cas illustrés par des diagrammes de séquences)
  - Les développeurs estiment les temps de développement

Planning

- Planning de la première release :
  - Uniquement les fonctionnalités essentielles
  - Première release à enrichir par la suite
- Durée du planning : 1 ou 2 jours
- Première version (release) au bout de 2 à 6 mois

Itérations jusqu'à la première release

- Développement de la **première version de l'application**
- Itérations de une à quatre semaines
  - Chaque itération produit un sous ensemble des fonctionnalités principales
  - Le produit de chaque itération subit des tests fonctionnels
  - Itérations courtes pour identifier très tôt des déviation par rapport au planning
- Brèves réunions quotidiennes réunissant toute l'équipe, pour mettre chacun au courant de l'avancement du projet

Mise en production

- La mise en production produit un logiciel
  - Offrant toutes les fonctionnalités indispensables
  - Parfaitement fonctionnel
  - Mis à disposition des utilisateurs
- Itérations très courtes
- Tests constants en parallèle du développement
- Les développeurs procèdent à des **réglages affinés** pour améliorer les performances du logiciel

Maintenance

- Continuer à **faire fonctionner le système**
- Adjonction de **nouvelles fonctionnalités secondaires**
  - Pour les fonctionnalités secondaires, on recommence par une rapide exploration
  - L'ajout de fonctionnalités secondaires donne lieu à de nouvelles releases

Mort

- Quand le client ne parvient plus à spécifier de nouveaux besoins, le projet est dit « mort »
  - Soit que tous les besoins possibles sont remplis
  - Soit que le système ne supporte plus de nouvelles modifications en restsant rentable

Equipe XP

- Pour un travil en équipe, on distingue 6 rôles principaux au sein d'une équipe XP
  - Développeur
  - Client
  - Testeur
  - Tracker
  - Manager
  - Coach

Développeur

- Conception et programmation, même combat !
- Participe aux séances de planification, évalue les tâches et leur difficulté
- Définition des test unitaires
- Implémentation des fonctionnalités et des tests unitaires

Client

- Écrit, explique et maîtrise les scénarios
- Spécifie les tests fonctionnels de recette
- Définit les priorités

Testeur

- Écriture des tests de recette automatiques pour valider les scénarios clients
- Peut influer sur les choix du clients en fonction de la « testatibilité » des scénarios

Tracker

- Suivre le planning pour chaque itération.
- Comprendre les estimations produites par les développeurs concernant leur charges
- Interagir avec les développeurs pour le respect du planning de l'itération courante
- Détection des éventuels retards et rectifications si besoin

Manager

- Supérieur hiérarchique des développeurs
  - Responsable du projet
- Vérification de la satisfaction du client
- Contrôle le planning
- Gestion des ressources

Coach

- Garant du processus XP
  - Organise et anime les séances de planifications
  - Favorise la créativité du groupe, n'impose pas ses solutions techniques
  - Coup de gueules. . .

Spécification avec XP

- Pas de documents d'analyse ou de spécifications détaillées
- **Les tests de recette remplacent les spécifications**
- Emergence de l'architecture au fur et à mesure du développement

XP vs RUP

- **Inconvénients de XP**
  - Focalisation sur l'aspect individuel du développement, au détriment d'une vue globale et des pratiques de management ou de formalisation
  - Manquer de contrôle et de structuration, risques de dérive
- **Inconvénients de RUP**
  - Fait tout, mais lourd, « usine à gaz »
  - Parfois difficile à mettre en oeuvre de façon spécifique.

- XP pour les petits projets en équipe de 12 max
- RUP pour les gros projets qui génèrent beaucoup de documentation