



Polytech Paris-Sud

Formation initiale 3^e année

Spécialité Informatique

Année 2012-2013

UML

Introduction au génie logiciel
et à la modélisation

Delphine Longuet
delphine.longuet@iri.fr

Logiciel

Ensemble d'**entités** nécessaires au fonctionnement d'un processus de traitement automatique de l'information

- Programmes, données, documentation...

Ensemble de **programmes** qui permet à un système **informatique** d'assurer une **tâche** ou une **fonction** en particulier

Logiciel = programme + utilisation

Logiciel

Aspects importants du logiciel

- **Environnement** :
 - utilisateurs : traitement de texte, distributeur de billets
 - autres logiciels : librairie, composant
 - matériel : ABS, système d'alarme
- **Spécification** : ce que doit faire le logiciel, ensemble de critères que doivent satisfaire son fonctionnement interne et ses interactions avec son environnement

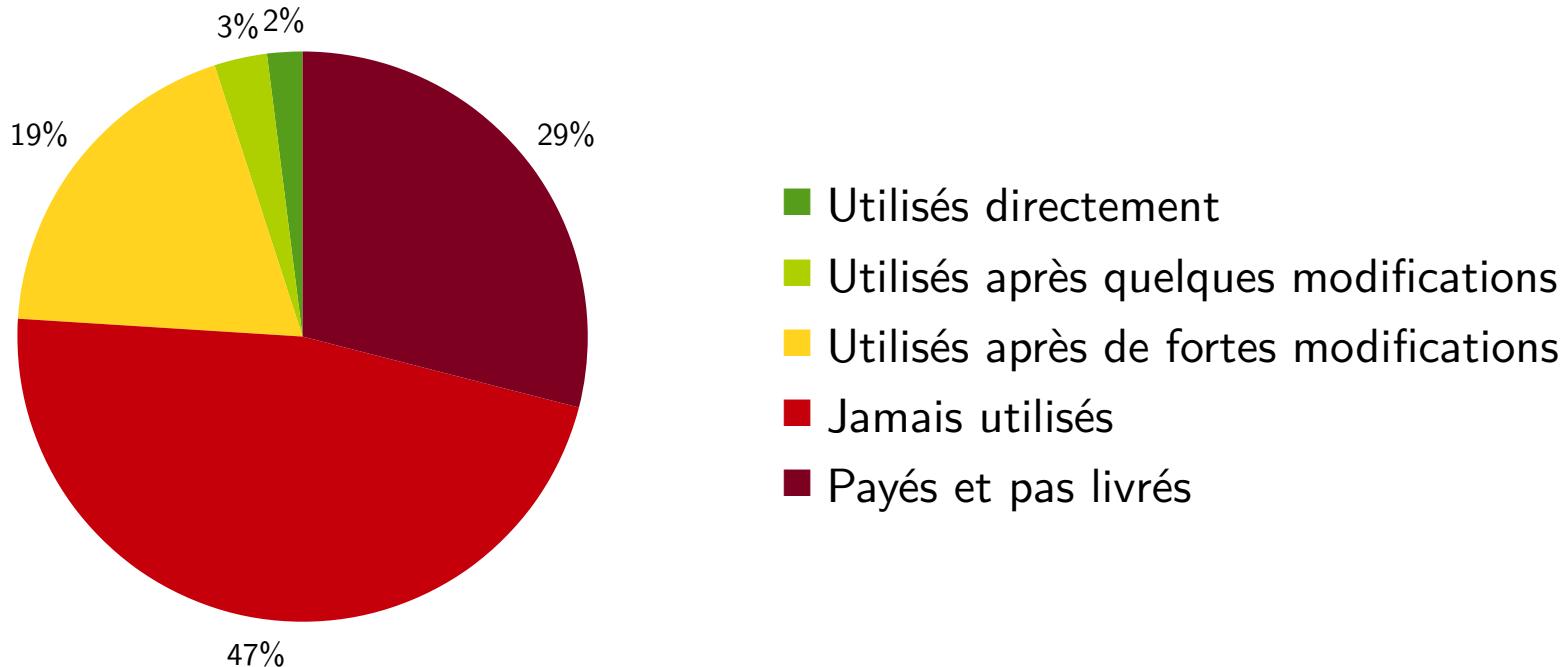
Crise du logiciel

Constat du développement logiciel fin années 60 :

- délais de livraison non respectés
- budget non respecté
- ne répond pas aux besoins de l'utilisateur ou du client
- difficile à utiliser, maintenir, et faire évoluer

Étude du DoD années 80

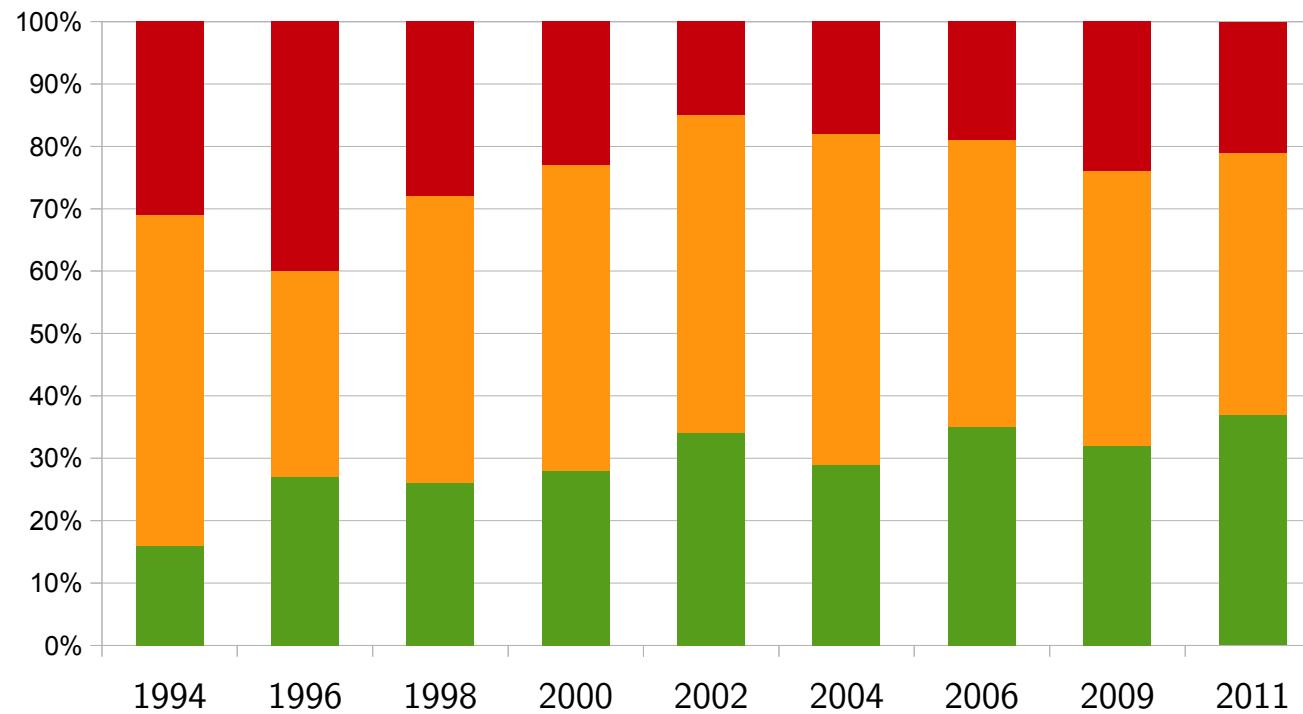
Étude du *Department of Defense* des États-Unis sur les logiciels produits dans le cadre de 9 gros projets militaires



Report of the Defense Science Board Task Force on Military Software, 1987

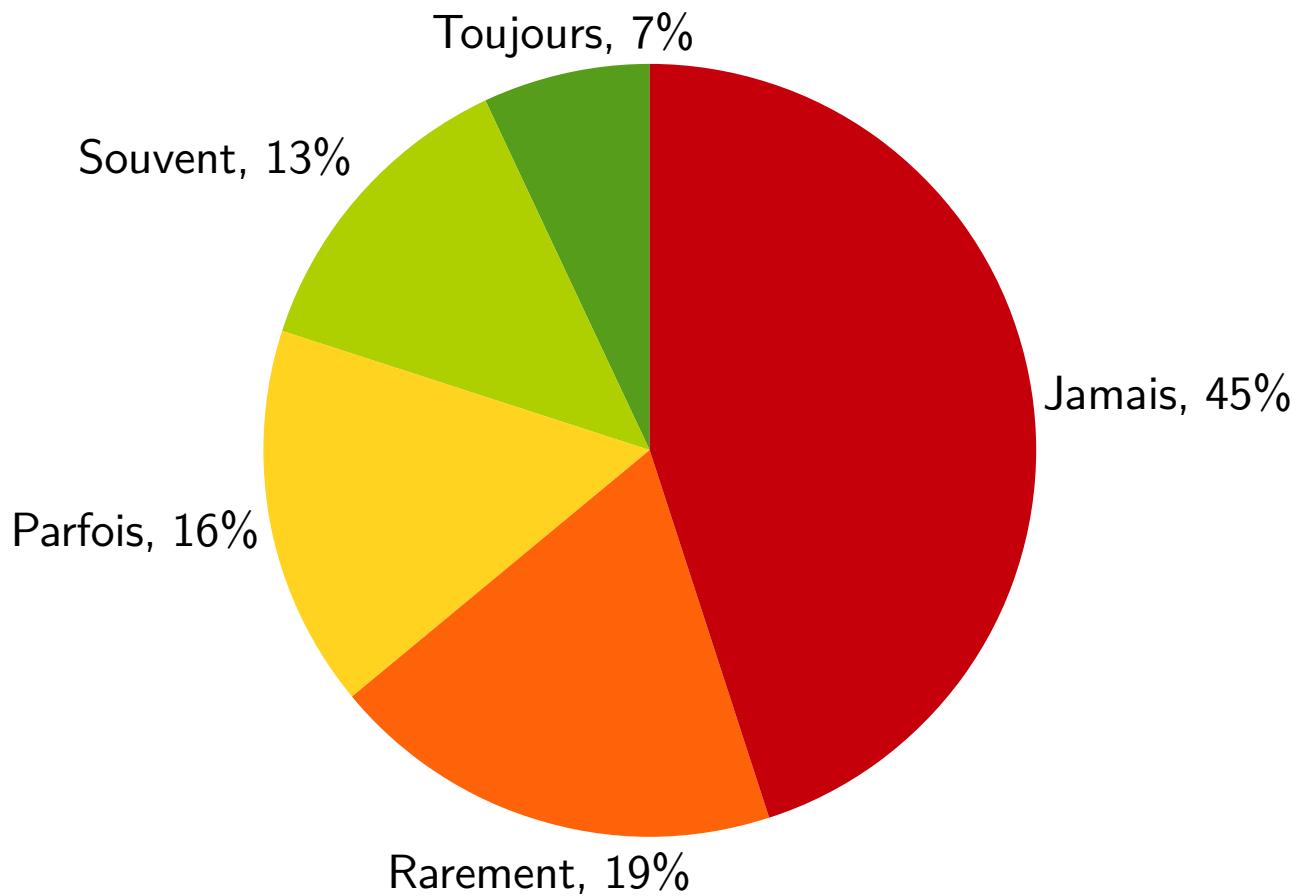
Étude du Standish group

En 1994 : enquête sur 8380 projets de toutes tailles et de tous secteurs



- Projets réussis : achevés dans les délais et pour le budget impartis, avec toutes les fonctionnalités demandées
- Projets mitigés : achevés et opérationnels, mais livrés hors délais, hors budget ou sans toutes les fonctionnalités demandées
- Projets ratés : abandonnés avant la fin

Utilisation des fonctionnalités implantées



Bugs célèbres

Sonde Mariner 1, 1962

- Détruite 5 minutes après son lancement
- Coût : 18,5 millions de dollars
- Défaillance des commandes de guidage due à une **erreur de spécification**
- Erreur de transcription manuelle d'un symbole mathématique dans la spécification

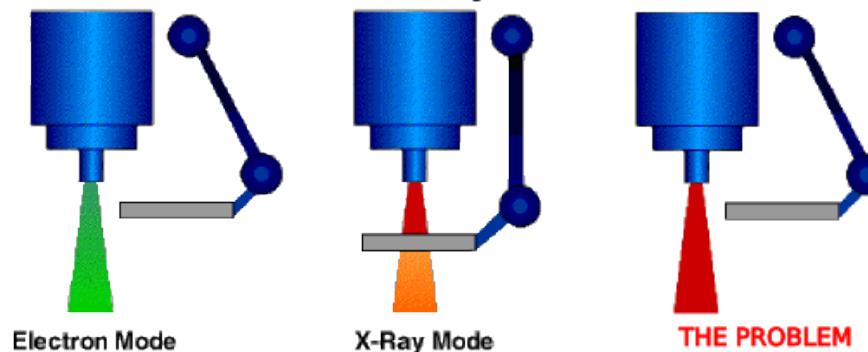


http://fr.wikipedia.org/wiki/Mariner_1

Bugs célèbres

Therac-25, 1985-87

- Au moins 5 morts par dose massive de radiations
- Problème d'accès concurrents dans le contrôleur



<http://fr.wikipedia.org/wiki/Therac-25>

Processeur Pentium, 1994

- Bug dans la table de valeurs utilisée par l'algorithme de division



http://fr.wikipedia.org/wiki/Bug_de_la_division_du_Pentium

Bugs célèbres

Ariane V vol 501, 1996

- Explosion après 40 secondes de vol
- Coût : 370 millions de dollars
- Panne du système de navigation due à un dépassement de capacité (arithmetic overflow)
- Réutilisation d'un composant d'Ariane IV non re-testé



http://fr.wikipedia.org/wiki/Vol_501_d'Ariane_5

Pourquoi ?

Raisons principales des bugs :

- Erreurs humaines
- Taille et complexité des logiciels
- Taille des équipes de conception/développement
- Manque de méthodes de conception
- Négligence de la phase d'analyse des besoins du client
- Négligence et manque de méthodes et d'outils des phases de validation/vérification

Mythes du logiciel

- Idée grossière du logiciel suffisante pour commencer à programmer
Faux : échecs dus principalement à une idée imprécise du logiciel
- Travail terminé quand programme écrit et fonctionnel
Faux : maintenance du logiciel = plus du 50% du coût total
- Facile de gérer spécifications changeantes
Faux : changements de spécifications souvent coûteux
- En cas de retard, solution : ajouter des programmeurs
Faux : période de familiarisation et communication plus difficile impliquent perte de productivité
« Ajouter des programmeurs à un projet en retard ne fait que le retarder davantage »

Génie logiciel

Idée : appliquer les méthodes classiques d'ingénierie au domaine du logiciel

Ingénierie (ou génie) : Ensemble des fonctions allant de la conception et des études à la responsabilité de la construction et au contrôle des équipements d'une installation technique ou industrielle

Génie civil, naval, aéronautique, mécanique, chimique...

Génie logiciel

Définition : Ensemble des méthodes, des techniques et des outils dédiés à la conception, au développement et à la maintenance des systèmes informatiques

Objectif : Avoir des procédures systématiques pour des logiciels de grande taille afin que

- la spécification corresponde aux besoins réels du client
- le logiciel respecte sa spécification
- les délais et les coûts alloués à la réalisation soient respectés

Génie logiciel

Particularités du logiciel :

- produit invisible et immatériel
- difficile de mesurer la qualité
- conséquences critiques causées par modifications infimes
- mises à jour et maintenance dues à l'évolution rapide de la technologie
- difficile de raisonner sur des programmes
- défaillances logicielles principalement humaines

Qualité du logiciel

Critères de qualité

- **Validité** : réponse aux besoins des utilisateurs
- **Facilité d'utilisation** : prise en main et robustesse
- **Performance** : temps de réponse, débit, fluidité...
- **Fiabilité** : tolérance aux pannes
- **Sécurité** : intégrité des données et protection des accès
- **Maintenabilité** : facilité à corriger ou transformer le logiciel
- **Portabilité** : changement d'environnement matériel ou logiciel

Principes d'ingénierie pour le logiciel

- Rigueur
- Abstraction
- Décomposition en sous-problèmes
- Modularité
- Construction incrémentale
- Généricité
- Anticipation des évolutions

- Documentation
- Standardisation/normalisation

Principes d'ingénierie pour le logiciel

- **Rigueur** : principale source d'erreurs **humaine**, s'assurer par tous les moyens que ce qu'on écrit est bien ce qu'on veut dire et que ça correspond à ce qu'on a promis (outils, revue de code...)
- **Abstraction**
- **Décomposition en sous-problèmes**
- **Modularité**
- **Construction incrémentale**
- **Généricité**
- **Anticipation des évolutions**
- **Documentation**
- **Standardisation/normalisation**

Principes d'ingénierie pour le logiciel

- Rigueur
- Abstraction : extraire des concepts généraux sur lesquels raisonner, puis instancier les solutions sur les cas particuliers
- Décomposition en sous-problèmes
- Modularité
- Construction incrémentale
- Généricité
- Anticipation des évolutions
- Documentation
- Standardisation/normalisation

Principes d'ingénierie pour le logiciel

- Rigueur
- Abstraction
- Décomposition en sous-problèmes : traiter chaque aspect séparément, chaque sous-problème plus simple que problème global
- Modularité
- Construction incrémentale
- Généricité
- Anticipation des évolutions
- Documentation
- Standardisation/normalisation

Principes d'ingénierie pour le logiciel

- Rigueur
- Abstraction
- Décomposition en sous-problèmes
- Modularité : partition du logiciel en modules interagissant, remplissant une fonction et ayant une interface cachant l'implantation aux autres modules
- Construction incrémentale
- Généricité
- Anticipation des évolutions

- Documentation
- Standardisation/normalisation

Principes d'ingénierie pour le logiciel

- Rigueur
- Abstraction
- Décomposition en sous-problèmes
- Modularité
- Construction incrémentale : construction pas à pas, intégration progressive
- Généricité
- Anticipation des évolutions
- Documentation
- Standardisation/normalisation

Principes d'ingénierie pour le logiciel

- Rigueur
- Abstraction
- Décomposition en sous-problèmes
- Modularité
- Construction incrémentale
- Généricité : proposer des solutions plus générales que le problème pour pouvoir les réutiliser et les adapter à d'autres cas
- Anticipation des évolutions
- Documentation
- Standardisation/normalisation

Principes d'ingénierie pour le logiciel

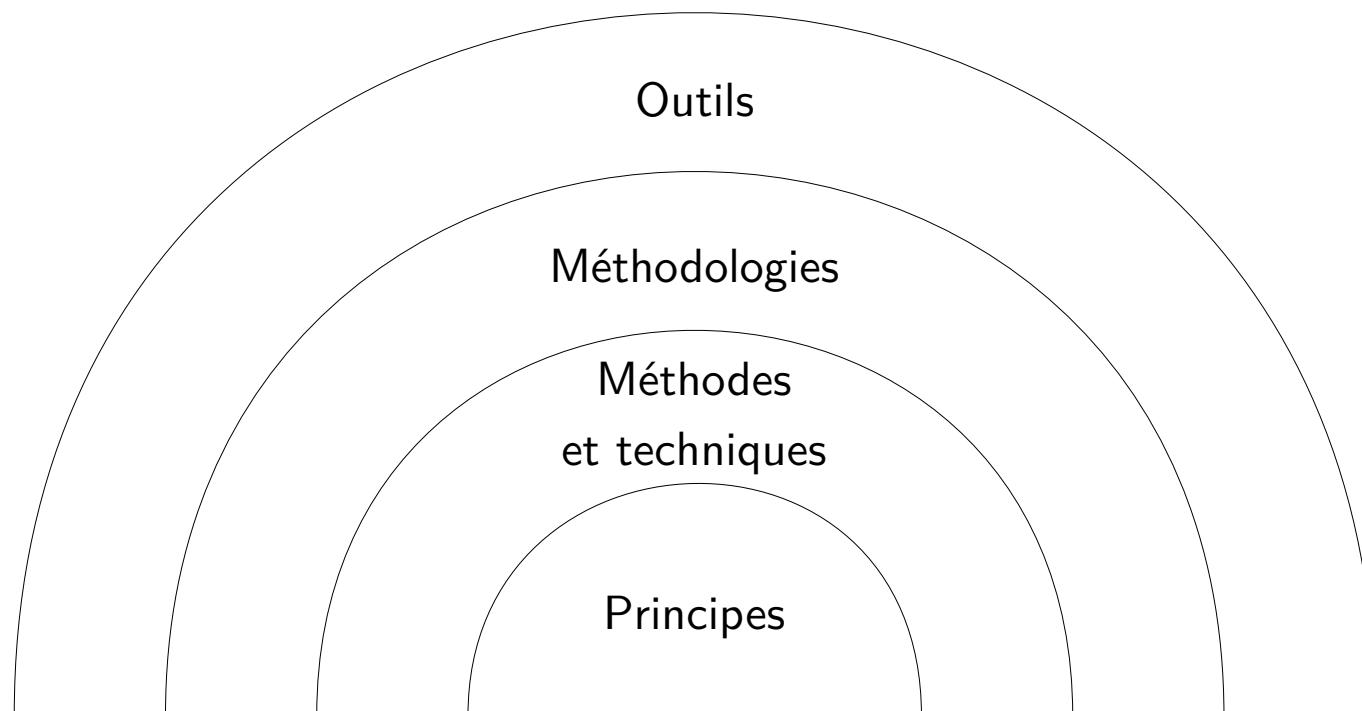
- Rigueur
- Abstraction
- Décomposition en sous-problèmes
- Modularité
- Construction incrémentale
- Généricité
- Anticipation des évolutions : liée à la généricité et à la modularité, prévoir les ajouts/modifications possibles de fonctionnalités
- Documentation
- Standardisation/normalisation

Principes d'ingénierie pour le logiciel

- Rigueur
- Abstraction
- Décomposition en sous-problèmes
- Modularité
- Construction incrémentale
- Généricité
- Anticipation des évolutions
- Documentation : essentielle pour le suivi de projet et la communication au sein de l'équipe de projet
- Standardisation/normalisation : aide à la communication pour le développement, la maintenance et la réutilisation

Des principes à la pratique

- Principes deviennent pratique grâce à méthodes et techniques
- Méthodes et techniques utilisés dans le cadre d'une méthodologie
- Méthodologie appuyée par des outils



Processus de développement logiciel

Ensemble d'activités successives, organisées en vue de la production d'un logiciel

En pratique :

- Pas de processus idéal
- Choix du processus en fonction des contraintes (taille des équipes, temps, qualité...)
- Adaptation de « processus types » aux besoins réels

Processus de développement logiciel

Activités du développement logiciel

- Analyse des besoins
- Spécification
- Conception
- Programmation
- Validation et vérification
- Livraison
- Maintenance

Pour chaque activité : Utilisation et production de documents

Analyse des besoins

Objectif : Comprendre les **besoins du client**

- Objectifs généraux, environnement du futur système, ressources disponibles, contraintes de performance...

Entrée : Fournies par le **client**

- Expert du domaine d'application, futur utilisateur

Document produit : Cahier des charges (+ manuel d'utilisation préliminaire)

Spécification

Objectifs :

- Établir une description claire de **ce que doit faire** le logiciel (fonctionnalités détaillées, exigences de qualité, interface...)
- Clarifier le cahier des charges (ambiguïtés, contradictions)

Entrée : Cahier des charges + considérations de faisabilité ?

Document produit : Cahier des charges fonctionnel

Conception

Objectif : Élaborer une solution concrète réalisant la spécification

- Description architecturale en composants (avec interface et fonctionnalités)
- Réalisation des fonctionnalités par les composants (algorithmes, organisation des données)
- Réalisation des exigences non-fonctionnelles (performance, sécurité...)

Entrée : Cahier des charges fonctionnel

Document produit : Dossier de conception

Programmation

Objectif : **Implantation** de la solution conçue

- Choix de l'environnement de développement, du/des langage(s) de programmation, de normes de développement...

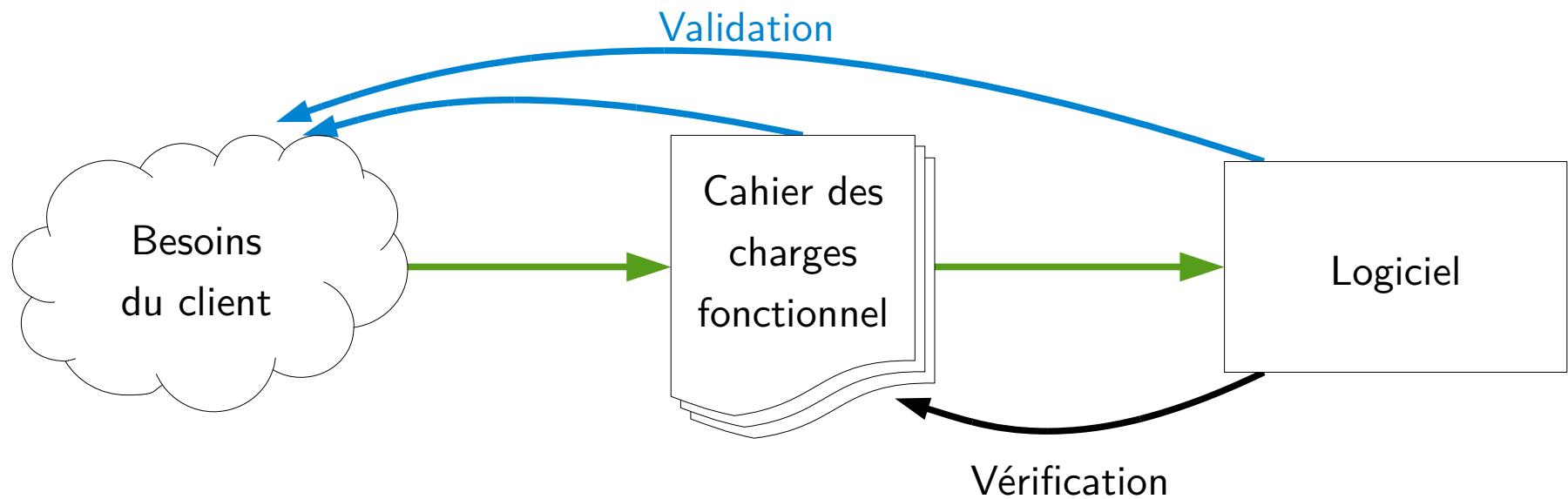
Entrée : Dossier de conception

Document produit : Code documenté + manuel d'utilisation

Validation et vérification

Objectifs :

- Validation : assurer que les **besoins du client** sont satisfaits (au niveau de la spécification, du produit fini...)
- Vérification : assurer que le logiciel satisfait sa **spécification**



Méthodes de validation et vérification

Méthodes de validation :

- Revue de spécification, de code
- Prototypage rapide
- Développement de tests exécutables (*extreme programming*)

Méthodes de vérification :

- Test (à partir de la spécification)
- Preuve de programmes
- Model-checking de programmes

Méthodes de vérification

Test (à partir de la spécification) : exécution d'un programme sur des données choisies dans le but de détecter des non-conformités par rapport à la spécification (cahier des charges fonctionnel)

Preuve de programmes : preuve mathématique qu'un programme satisfait sa spécification en termes de pré- et post-conditions

Model-checking de programmes : analyse d'un programme dans le but de prouver mathématiquement qu'il vérifie certaines propriétés dynamiques

Méthodes de vérification

Test :

- ✓ Nécessaire : exécution du système réel, découverte d'erreurs à tous les niveaux (spécification, conception, implantation)
- ✗ Pas suffisant : exhaustivité impossible

Preuve :

- ✓ Exhaustif
- ✗ Mise en oeuvre difficile, limitation de taille

Model-checking :

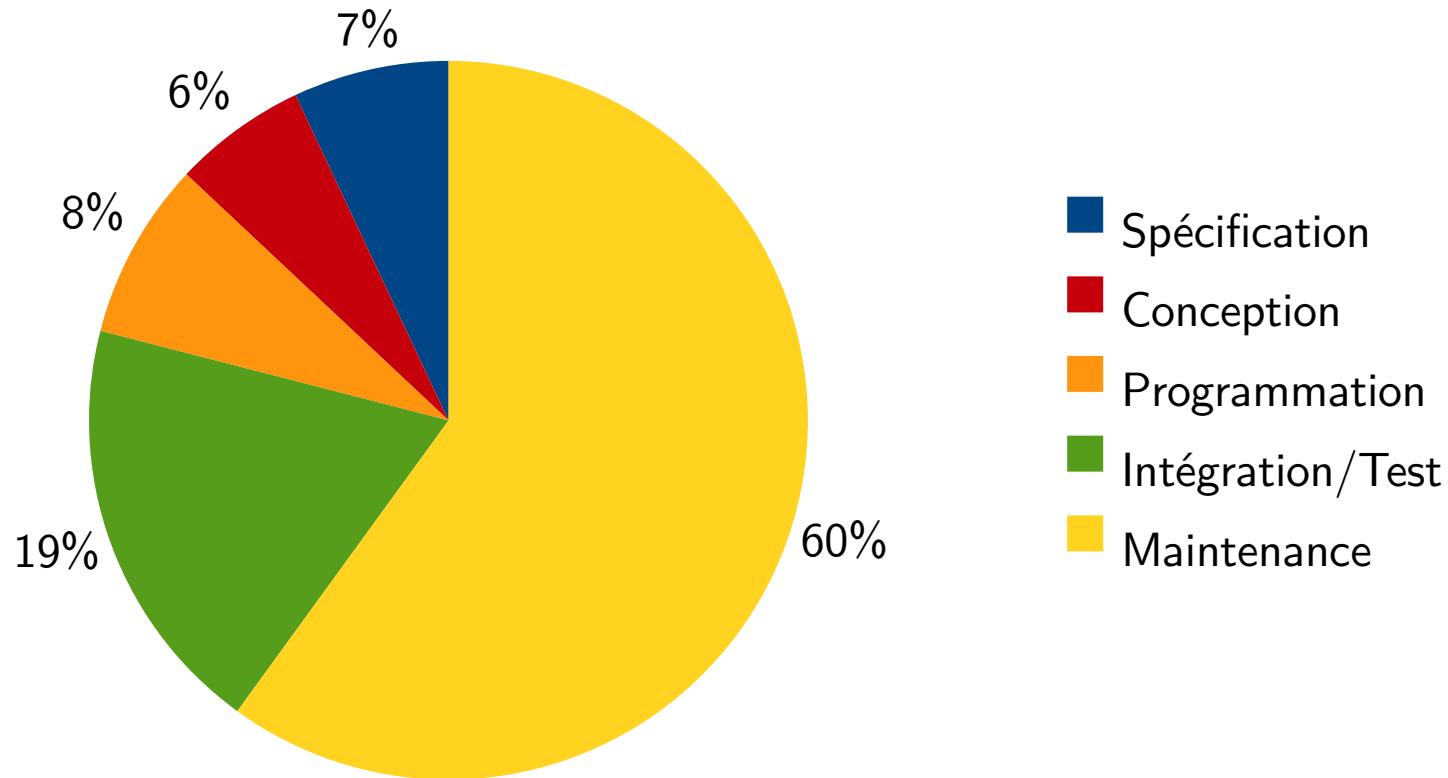
- ✓ Exhaustif, partiellement automatique
- ✗ Mise en oeuvre moyennement difficile (modèles formels, logique)

Maintenance

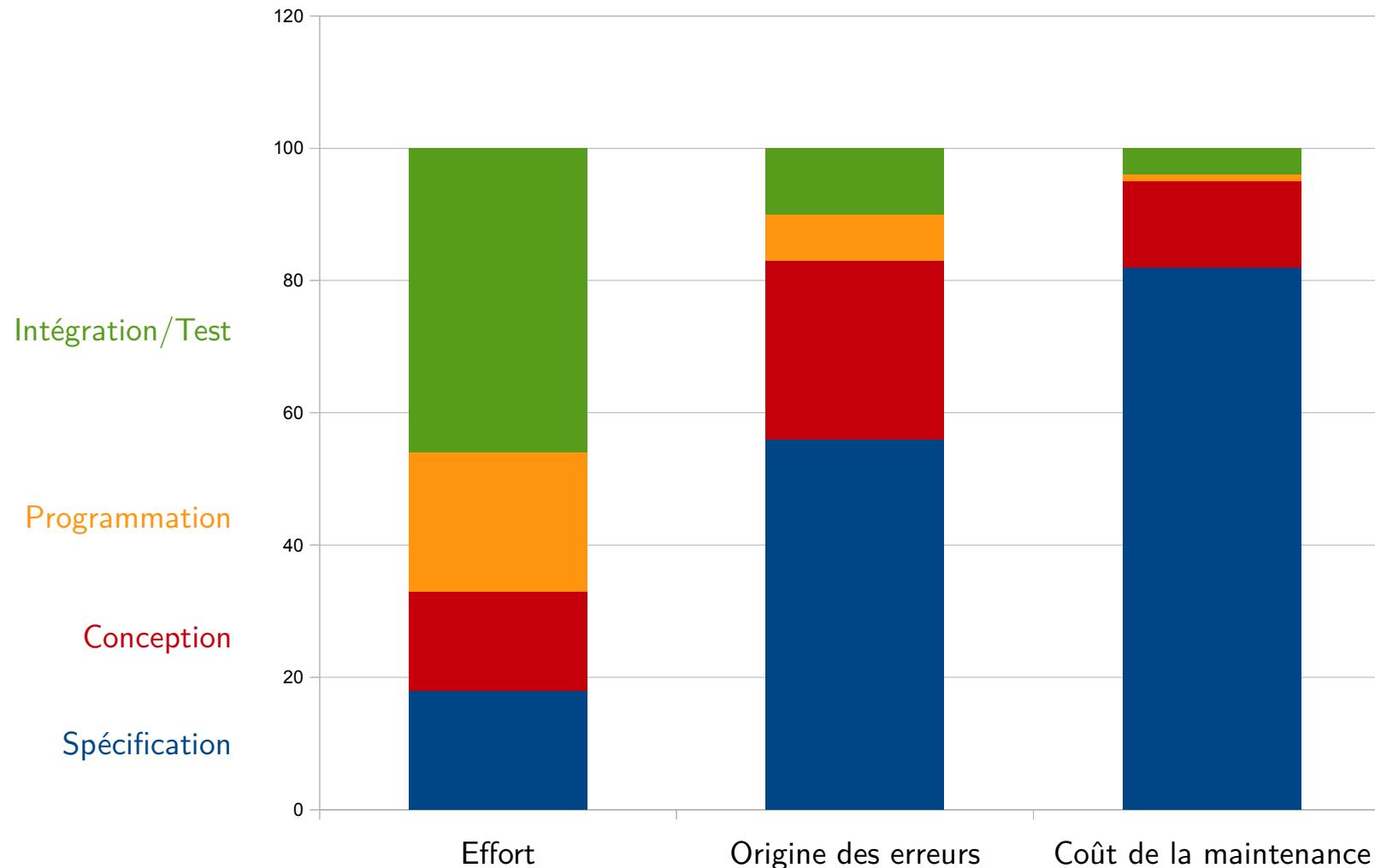
Types de maintenance :

- Correction : identifier et corriger des erreurs trouvées après la livraison
- Adaptation : adapter le logiciel aux changements dans l'environnement (format des données, environnement d'exécution...)
- Perfection : améliorer la performance, ajouter des fonctionnalités, améliorer la maintenabilité du logiciel

Répartition de l'effort

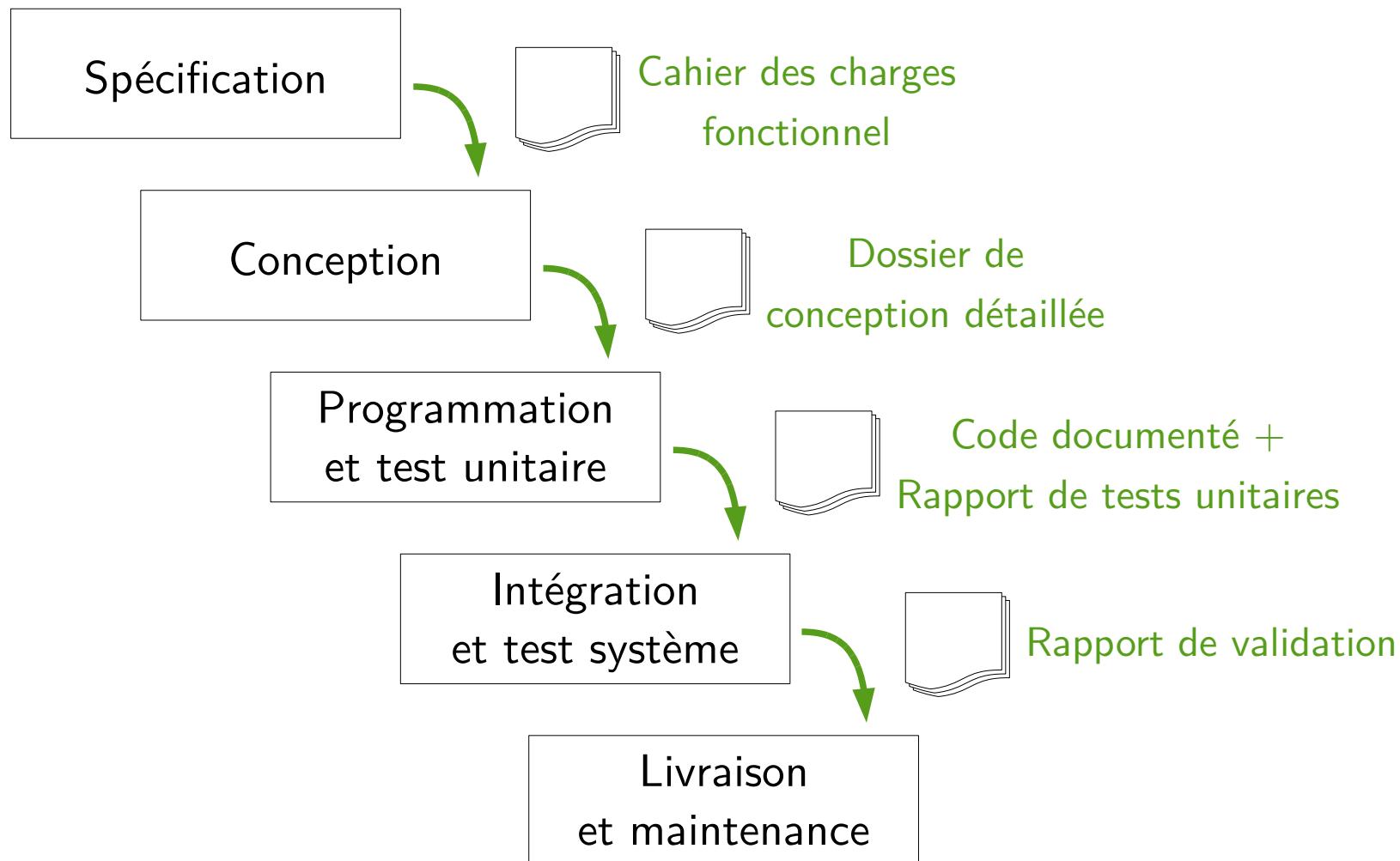


Rapport effort/erreur/coût



Processus en cascade

Chaque étape doit être terminée avant que ne commence la suivante
À chaque étape, production d'un document base de l'étape suivante



Processus en cascade

Caractéristiques :

- Hérité des méthodes classiques d'ingénierie
- Découverte d'une erreur entraîne retour à la phase à l'origine de l'erreur et nouvelle cascade, avec de nouveaux documents...
- Coût de modification d'une erreur important, donc choix en amont cruciaux (typique d'une production industrielle)

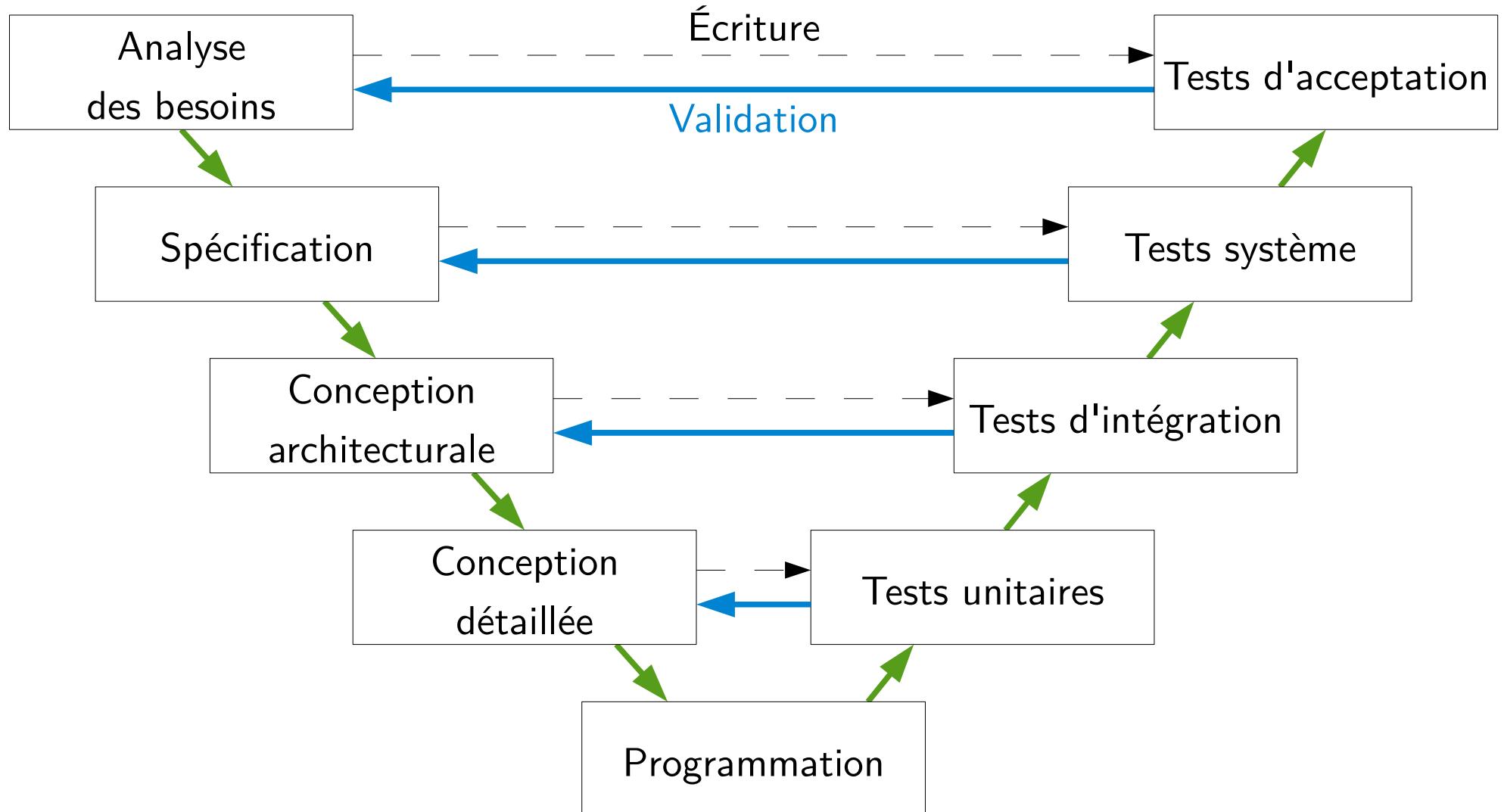
Pas toujours adapté à une production logicielle, en particulier si besoins du client changeants ou difficiles à spécifier

Processus en V

Caractéristiques :

- Variante du modèle en cascade
- Mise en évidence de la complémentarité des phases menant à la réalisation et des phases de test permettant de les valider

Processus en V



Niveaux de test

Test unitaire : test de chaque unité de programme (méthode, classe, composant), indépendamment du reste du système

Test d'intégration : test des interactions entre composants (interfaces et composants compatibles)

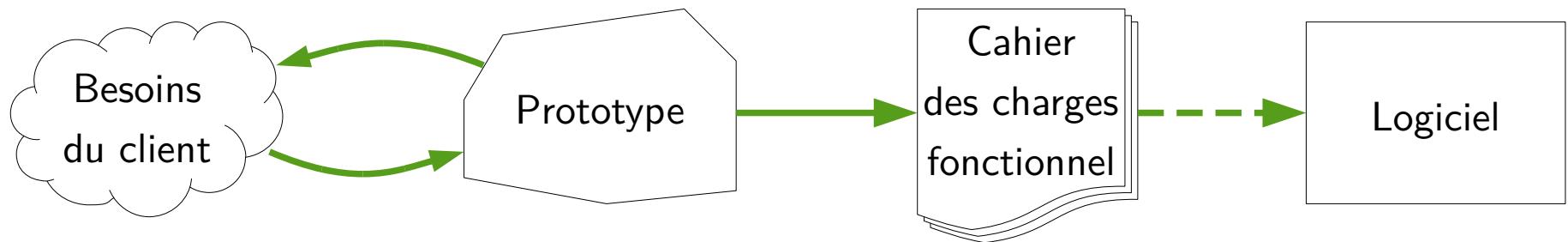
Test système : test du système complet par rapport à son cahier des charges

Test d'acceptation (recette) : fait par le client, validation par rapport aux besoins initiaux

Développement par prototypage

Principe :

- Développement rapide d'un prototype avec le client pour valider ses besoins
- Écriture de la spécification à partir du prototype, puis processus de développement linéaire

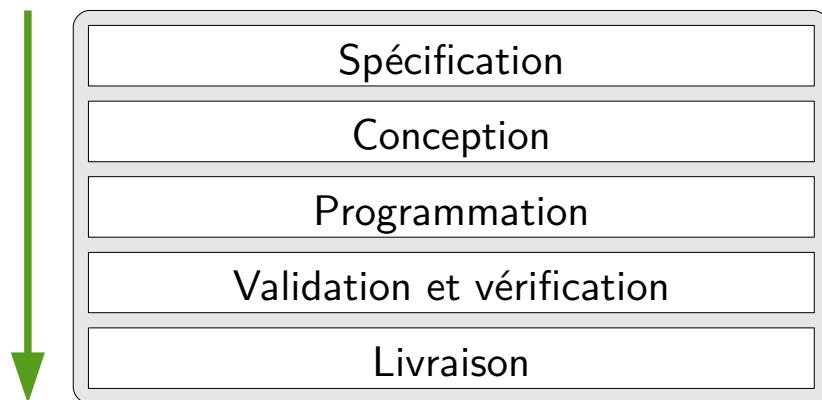


Avantage : Validation concrète des besoins, moins de risque d'erreur de spécification

Développement incrémental

Principe :

- Hiérarchiser les besoins du client
- Concevoir et livrer au client un produit implantant un sous-ensemble de fonctionnalités par ordre de priorité



Développement en cascade



Développement incrémental

Avantage : Minimiser le risque d'inadéquation aux besoins

Difficulté : Intégration fonctionnalités secondaires non pensées en amont

Méthodes agiles et *extreme programming*

Principes :

- Implication constante du client
- Programmation en binôme (revue de code permanente)
- Développement dirigé par les tests
- Cycles de développement rapides pour réagir aux changements

Avantages : développement rapide en adéquation avec les besoins

Inconvénients : pas de spécification, documentation = tests,
maintenance ?

Fonctionne pour petites équipes de développement (< 20) car
communication cruciale

Documentation

Objectif : Traçabilité du projet

Pour l'équipe :

- Regrouper et structurer les décisions prises
- Faire référence pour les décisions futures
- Garantir la cohérence entre les modèles et le produit

Pour le client :

- Donner une vision claire de l'état d'avancement du projet

Base commune de référence :

- Personne quittant le projet : pas de perte d'informations
- Personne rejoignant le projet : intégration rapide

Documents de spécification et conception

Rédaction : le plus souvent en langage naturel (français)

Problèmes :

- Ambiguïtés : plusieurs sens d'un même mot selon les personnes ou les contextes
- Contradictions, oublis, redondances difficiles à détecter
- Difficultés à trouver une information
- Mélange entre les niveaux d'abstraction (spécification vs. conception)

Documents de spécification et conception

Alternatives au langage naturel

Langages informels :

- Langage naturel structuré : modèles de document et règles de rédaction précis et documentés
- Pseudo-code : description algorithmique de l'exécution d'une tâche, donnant une vision opérationnelle du système

Langages semi-formels :

- Notation graphique : diagrammes accompagnés de texte structuré, donnant une vue statique ou dynamique du système

Langages formels :

- Formalisme mathématique : propriétés logiques ou modèle du comportement du système dans un langage mathématique

Documents de spécification et conception

Langages informels ou semi-formels :

- ✓ Avantages : intuitifs, fondés sur l'expérience, facile à apprendre et à utiliser, répandus
- ✗ Inconvénients : ambigus, pas d'analyse systématique

Langages formels :

- ✓ Avantages : précis, analysables automatiquement, utilisables pour automatiser la vérification et le test du logiciel
- ✗ Inconvénients : apprentissage et maîtrise difficiles

En pratique : utilisation de langages formels principalement pour logiciels critiques, ou restreinte aux parties critiques du système

Modélisation

Modèle : Simplification de la réalité, abstraction, vue subjective

- modèle météorologique, économique, démographique...

Modéliser un concept ou un objet pour :

- Mieux le comprendre (modélisation en physique)
- Mieux le construire (modélisation en ingénierie)

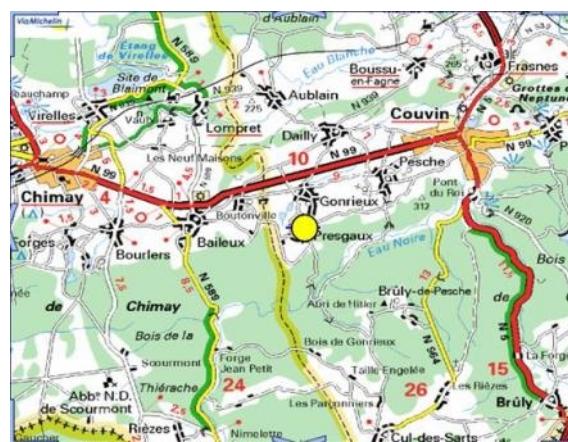
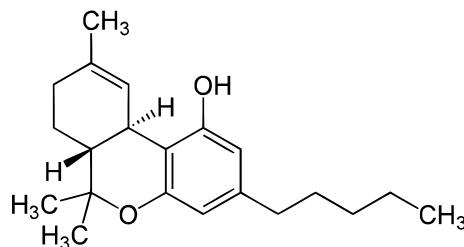
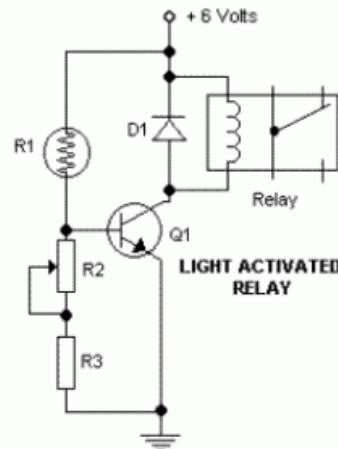
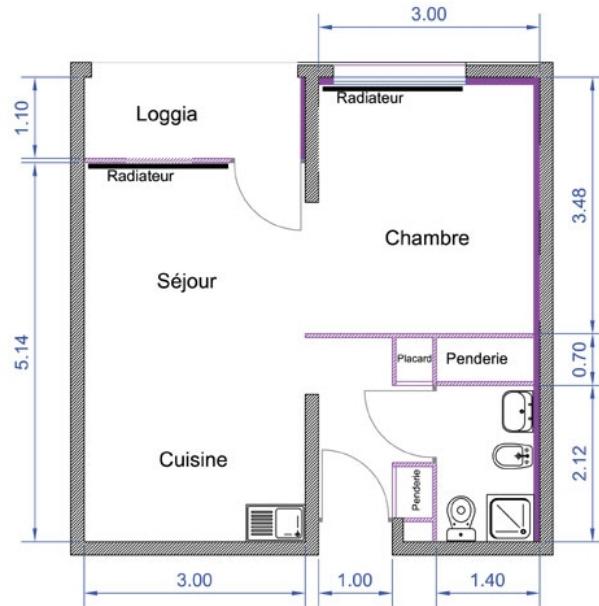
En génie logiciel :

- Modélisation = spécification + conception
- Aider la réalisation d'un logiciel à partir des besoins du client

Modélisation graphique

Principe : « *Un beau dessin vaut mieux qu'un long discours* »

Seulement s'il est compris par tous de la même manière



UML : Unified Modeling Language

Langage :

- Syntaxe et règles d'écriture
- Notations graphiques normalisées

... de modélisation :

- Abstraction du fonctionnement et de la structure du système
- Spécification et conception

... unifié :

- Fusion de plusieurs notations antérieures : Booch, OMT, OOSE
- Standard défini par l'OMG (Object Management Group)
- Dernière version : UML 2.4.1 (août 2011)

En résumé : Langage graphique pour visualiser, spécifier, construire et documenter un logiciel

Pourquoi UML ?

Besoin de modéliser pour construire un logiciel

- Modélisation des aspects statiques et dynamiques
- Modélisation à différents niveaux d'abstraction et selon plusieurs vues
- Indépendant du processus de développement

Besoin de langages normalisés pour la modélisation

- Langage semi-formel
- Standard très utilisé

Conception orientée objet

- Façon efficace de penser le logiciel
- Indépendance du langage de programmation (langages non objet)

Méthodes de conception

Conception fonctionnelle

- Système = ensemble de fonctions
- État du système (données) centralisé et partagé par les fonctions

Conception guidée par les données

- Système = base de données
- Fonctions communes à toutes les données
- Adaptée à l'élaboration de grandes bases de données

Conception orientée objet

- Système = ensemble d'objets
- Objet = données + fonctions
- État du système distribué entre tous les objets

Conception orientée objet

Principes

- Concept du domaine d'application = **objet**
Décrit par **état** (attributs) + **comportement** (opérations)
- Liens entre concepts : héritage, agrégation, composition...

Caractéristiques des objets

- **Identité** : objet = entité unique (mêmes attributs \Rightarrow même objet)
- **Classification** : regroupement des objets de même nature
(attributs + opérations)
- **Polymorphisme** : comportement différent d'une même opération dans différentes classes
- **Héritage** : partage hiérarchique des attributs et opérations

Conception orientée objet avec UML

UML

- Langage graphique : Ensemble de diagrammes permettant de modéliser le logiciel à selon différentes vues et à différents niveaux d'abstraction
- Modélisation orientée objet : modélisation du système comme un ensemble d'objets interagissant

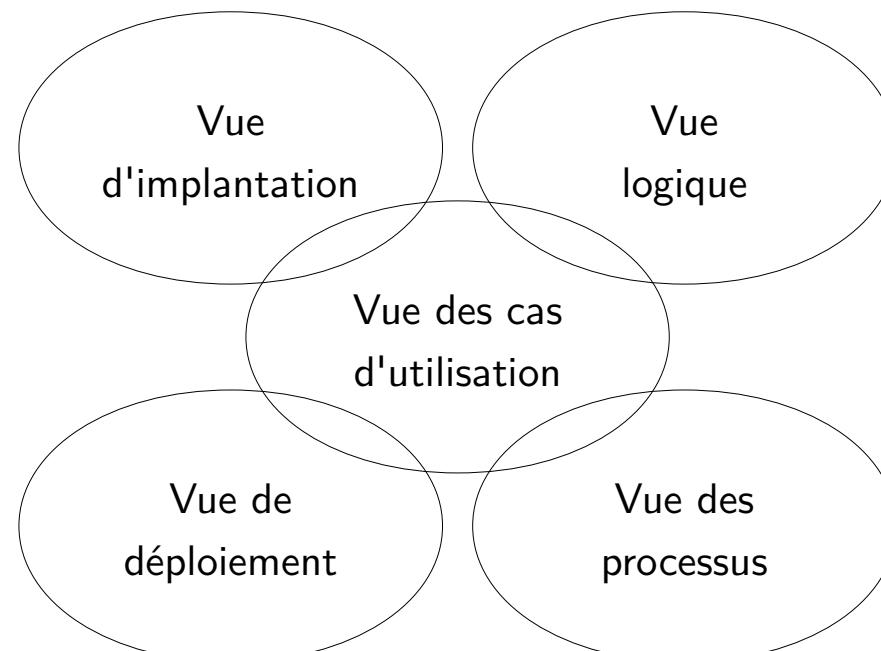
UML n'est pas une méthode de conception

UML est un outil indépendant de la méthode

Diagrammes UML

Représentation du logiciel à différents points de vue :

- **Vue des cas d'utilisation** : vue des acteurs (besoins attendus)
- **Vue logique** : vue de l'intérieur (satisfaction des besoins)
- **Vue d'implantation** : dépendances entre les modules
- **Vue des processus** : dynamique du système
- **Vue de déploiement** : organisation environnementale du logiciel



Diagrammes UML

14 diagrammes hiérarchiquement dépendants

Modélisation à tous les niveaux le long du processus de développement

Diagrammes structurels :

- Diagramme de classes
- Diagramme d'objets
- Diagramme de composants
- Diagramme de déploiement
- Diagramme de paquetages
- Diagramme de structure composite
- Diagramme de profils

Diagrammes comportementaux :

- Diagramme de cas d'utilisation
- Diagramme états-transitions
- Diagramme d'activité

Diagrammes d'interaction :

- Diagramme de séquence
- Diagramme de communication
- Diagramme global d'interaction
- Diagramme de temps

Exemple d'utilisation des diagrammes

- Diagrammes de **cas d'utilisation** : **besoins** des utilisateurs
- Diagrammes de **séquence** : **scénarios** d'interactions entre les utilisateurs et le logiciel, vu de l'extérieur
- Diagrammes d'**activité** : enchaînement d'actions représentant un **comportement du logiciel**

- Diagrammes de **classes** : **structure interne** du logiciel
- Diagrammes d'**objet** : état interne du logiciel à un instant donné
- Diagrammes **états-transitions** : évolution de l'état d'un objet
- Diagrammes de **séquence** : scénarios d'interactions avec les utilisateurs ou **au sein du logiciel**
- Diagrammes de **composants** : **composants physiques** du logiciel
- Diagrammes de **déploiement** : organisation **matérielle** du logiciel

Dans ce cours

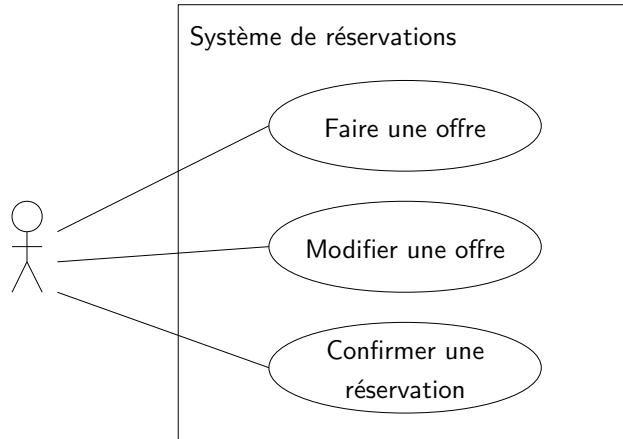


Diagramme de cas d'utilisation

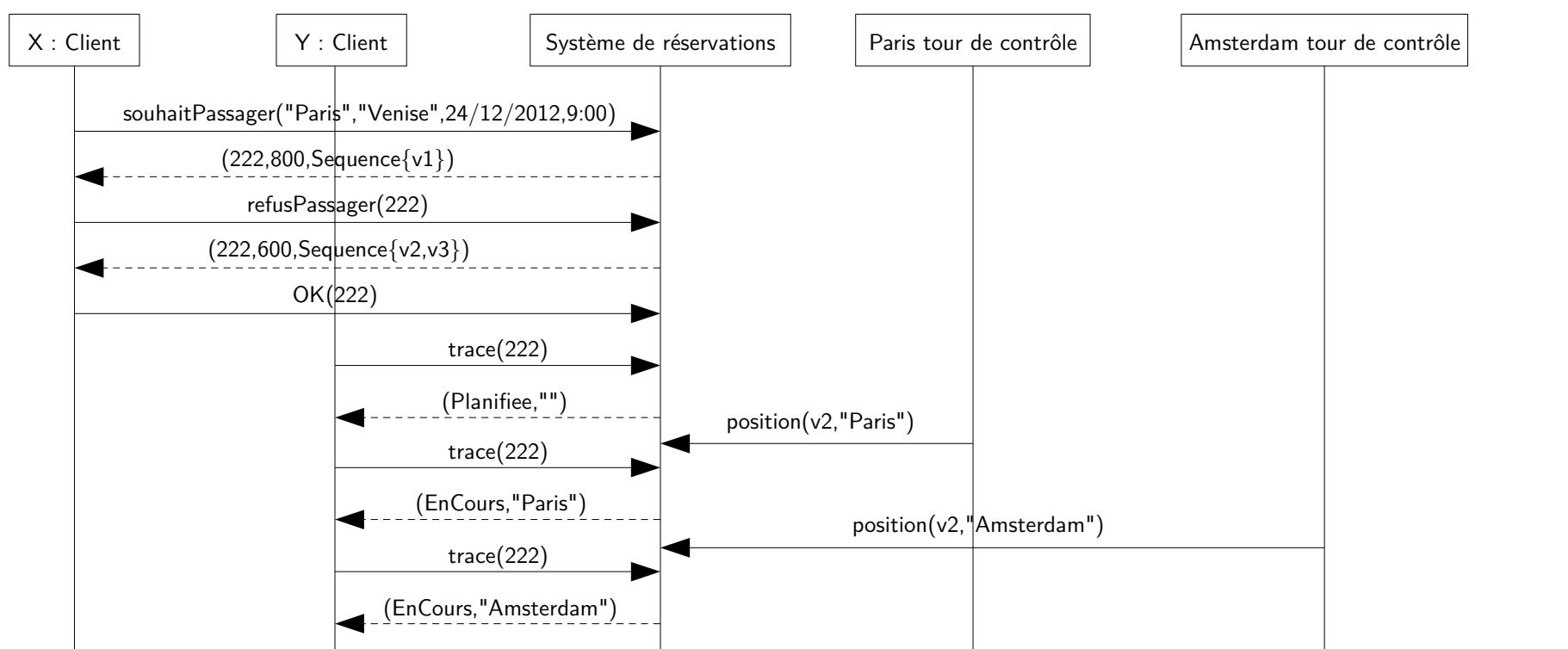


Diagramme de séquence

Dans ce cours

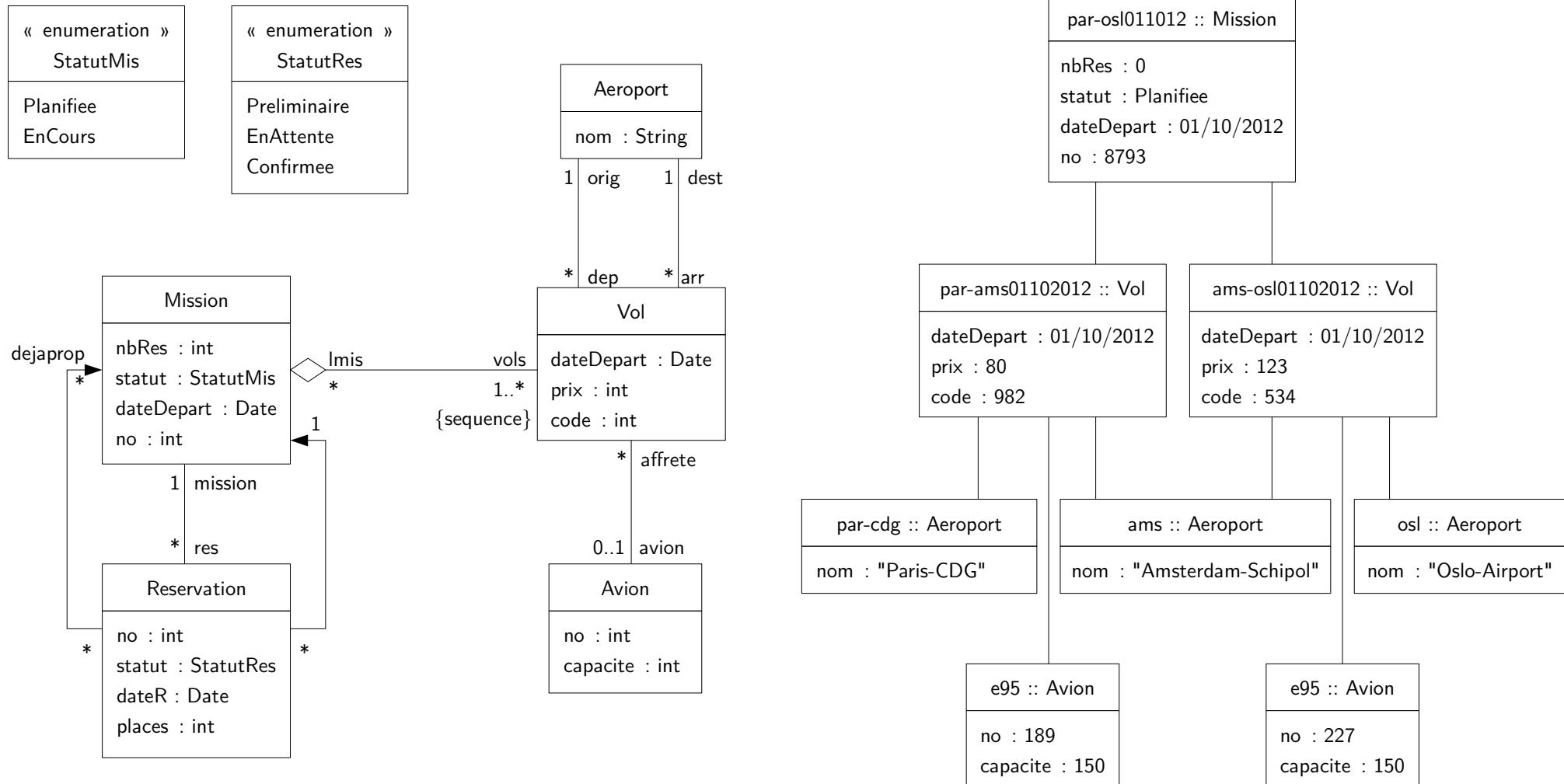


Diagramme de classes

Diagramme d'objets

Dans ce cours

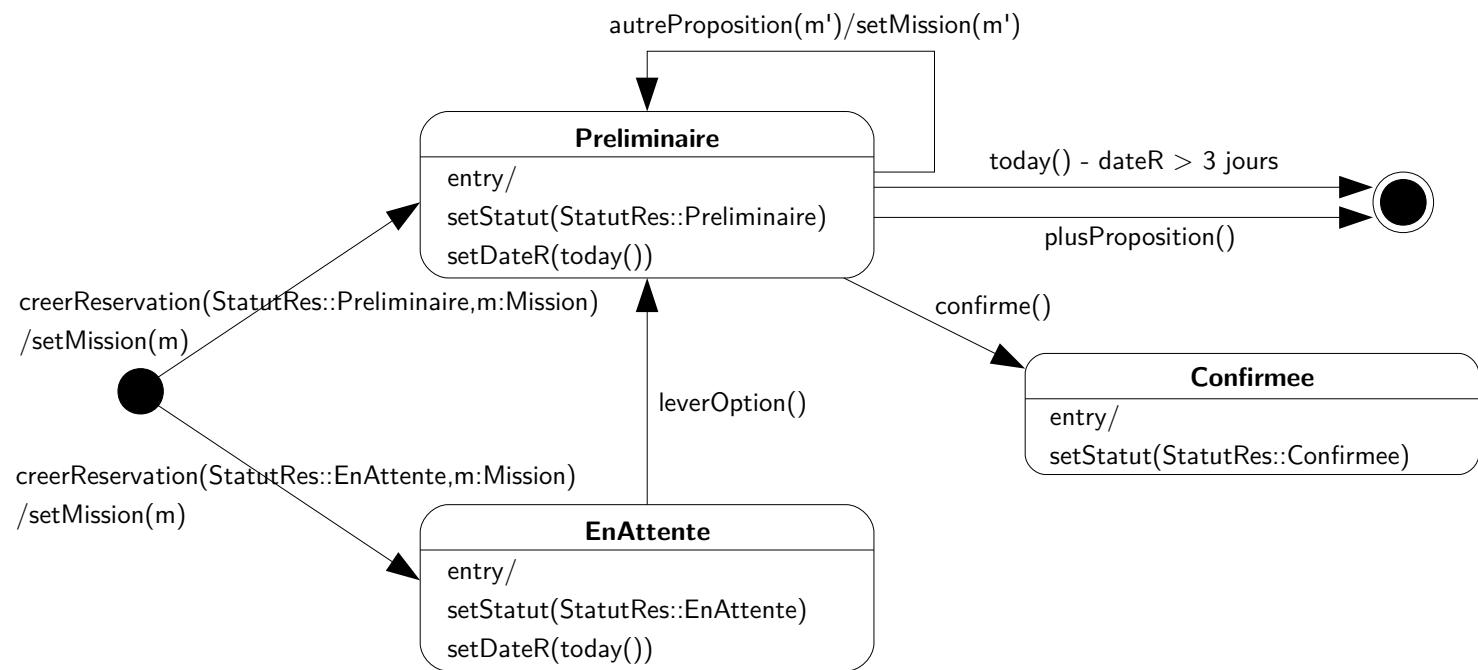


Diagramme états-transitions