



Université Paris-Sud
L3 MIAGE apprentissage
Année 2014-2015

Génie logiciel avancé

Introduction

Delphine Longuet

delphine.longuet@lri.fr

Logiciel : définitions

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 : caractéristiques

Environnement

- utilisateurs : grand public (traitement de texte),
spécialistes (calcul scientifique, finances),
développeurs (compilateur)
- autres logiciels : librairie, composant
- matériel : capteurs (système d'alarme),
réseau physique (protocole),
machine ou composant matériel contrôlé (ABS)

Spécification : ce que doit faire le logiciel, ensemble de critères que doivent satisfaire son fonctionnement interne et ses interactions avec son environnement

Spécification

Spécification fonctionnelle

- **Fonctionnalités** du logiciel, réponse aux besoins des utilisateurs

Spécification non fonctionnelle

- **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 et à faire évoluer le logiciel
- **Portabilité** : adaptabilité à d'autres environnements matériels ou logiciels

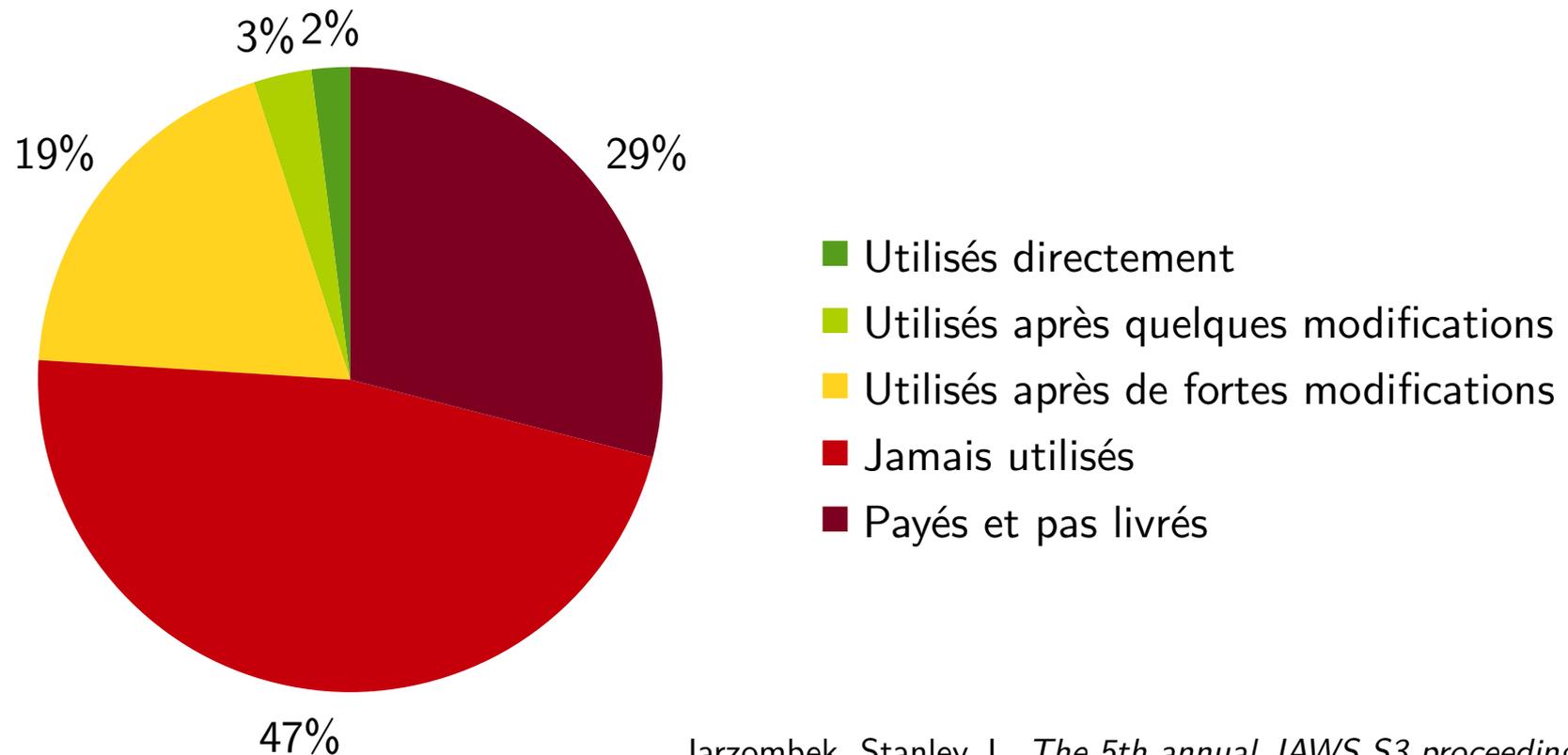
Crise du logiciel

Constat du développement logiciel fin années 60 :

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

Étude du DoD 1995

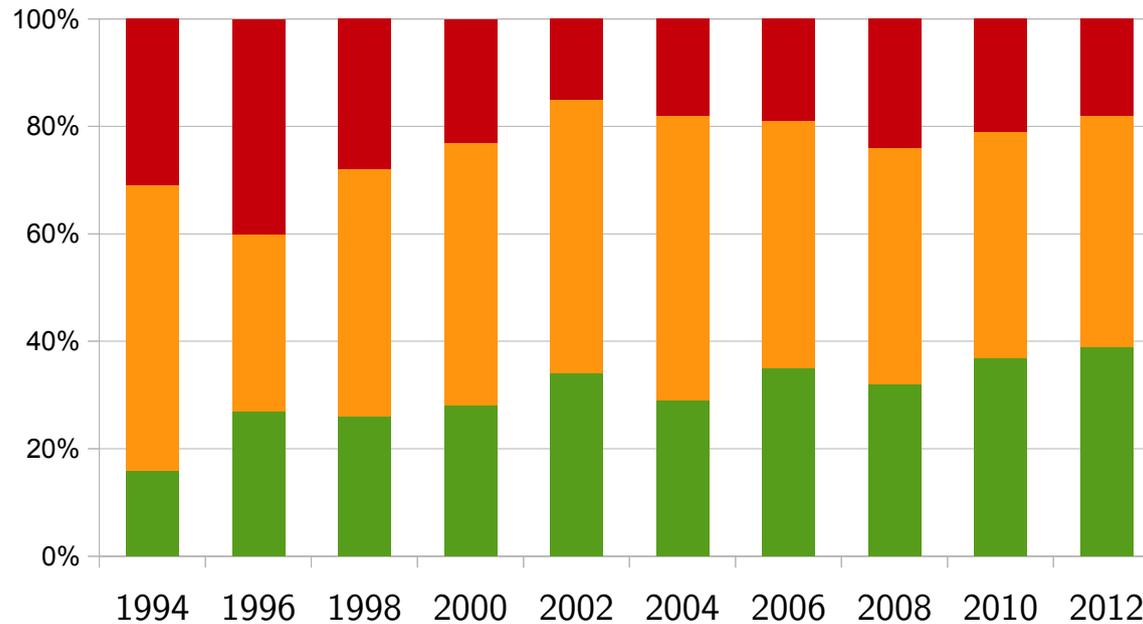
Étude du *Department of Defense* des États-Unis sur les logiciels produits dans le cadre de 9 gros projets militaires



Jarzombek, Stanley J., *The 5th annual JAWS S3 proceedings*, 1999

Étude du Standish group

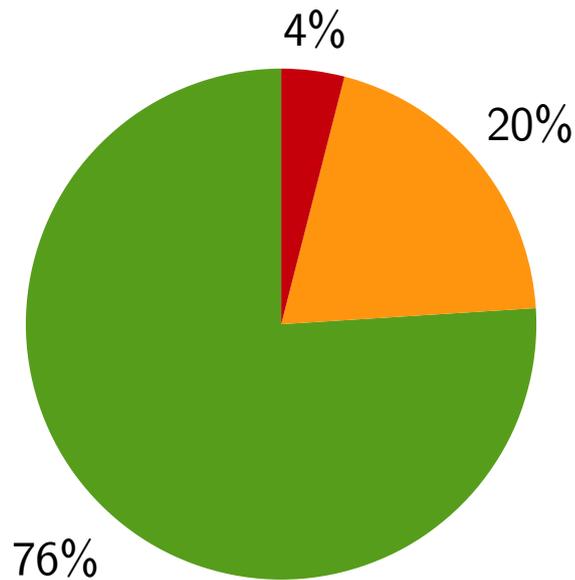
Enquête sur des milliers de projets, de toutes tailles et de tous secteurs



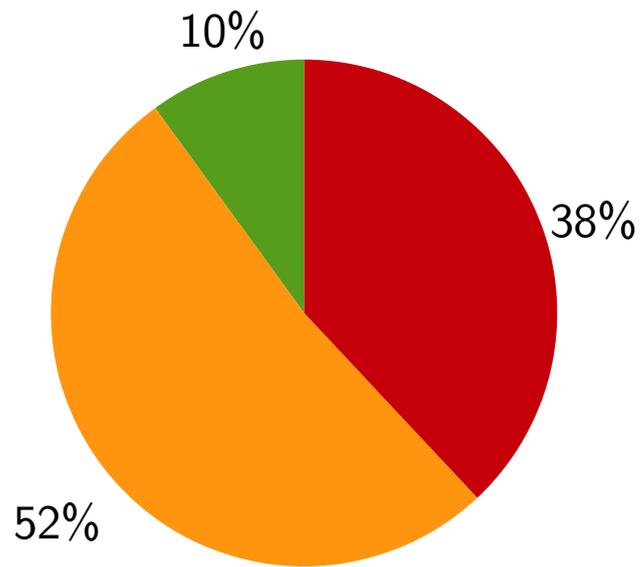
Standish group, *Chaos Manifesto 2013 - Think Big, Act Small*, 2013

- **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 ou livrés mais jamais utilisés

Petits vs grands projets



Petits projets
budget \leq \$1 million

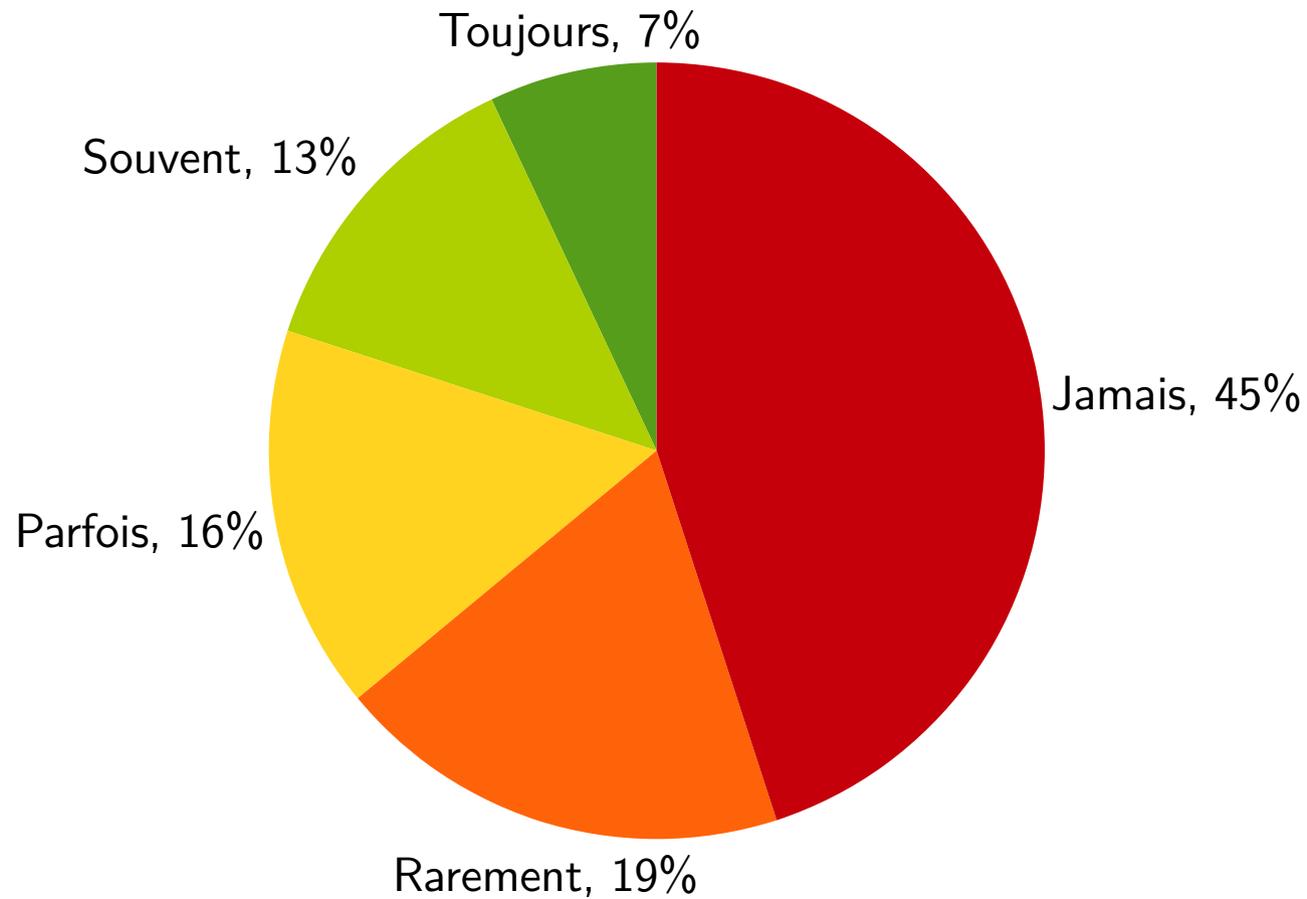


Grands projets
budget \geq \$10 millions

- Projets réussis
- Projets mitigés
- Projets ratés

Standish group, *Chaos Manifesto 2013 - Think Big, Act Small*, 2013

Utilisation des fonctionnalités implantées



Standish group, *Chaos Manifesto 2002*, 2002

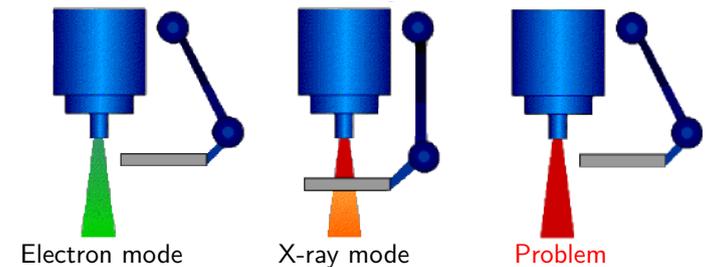
Bugs célèbres

Sonde Mariner 1, 1962

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

Therac-25, 1985-87

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



Processeur Pentium, 1994

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



Ariane V vol 501, 1996

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



Plus récemment

PlayStation Network, avril 2011

- Des millions de données personnelles et bancaires piratées
- Pertes financières de plusieurs milliards de dollars
- Vulnérabilité du réseau connue mais conséquences mal évaluées ?



Outil de chiffrement OpenSSL, mars 2014

- 500 000 serveurs web concernés par la faille
- Vulnérabilité permettant de lire une portion de la mémoire d'un serveur distant



De manière générale, **fiabilité, sûreté et sécurité des logiciels**

- Transports automobile, ferroviaire, aéronautique
- Contrôle de processus industriels, nucléaire, armement
- Médical : imagerie, appareillage, télé-surveillance
- e-commerce, carte bancaire sans contact, passeport électronique

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

Principes d'ingénierie pour le logiciel

Sept principes fondamentaux

- **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** : extraire des **concepts généraux** sur lesquels **raisonner**, puis instancier les solutions sur les cas particuliers
- **Décomposition en sous-problèmes** : traiter **chaque aspect séparément**, chaque **sous-problème plus simple** que problème global
- **Modularité** : **partition** du logiciel en **modules interagissant**, remplissant une **fonction** et ayant une **interface** cachant l'implantation aux autres modules

Principes d'ingénierie pour le logiciel

- **Construction incrémentale** : construction pas à pas, intégration progressive
- **Généricité** : proposer des solutions plus générales que le problème pour pouvoir les réutiliser et les adapter à d'autres cas
- **Anticipation des évolutions** : liée à la généricité et à la modularité, prévoir les ajouts/modifications possibles de fonctionnalités

De façon transversale

- **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

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

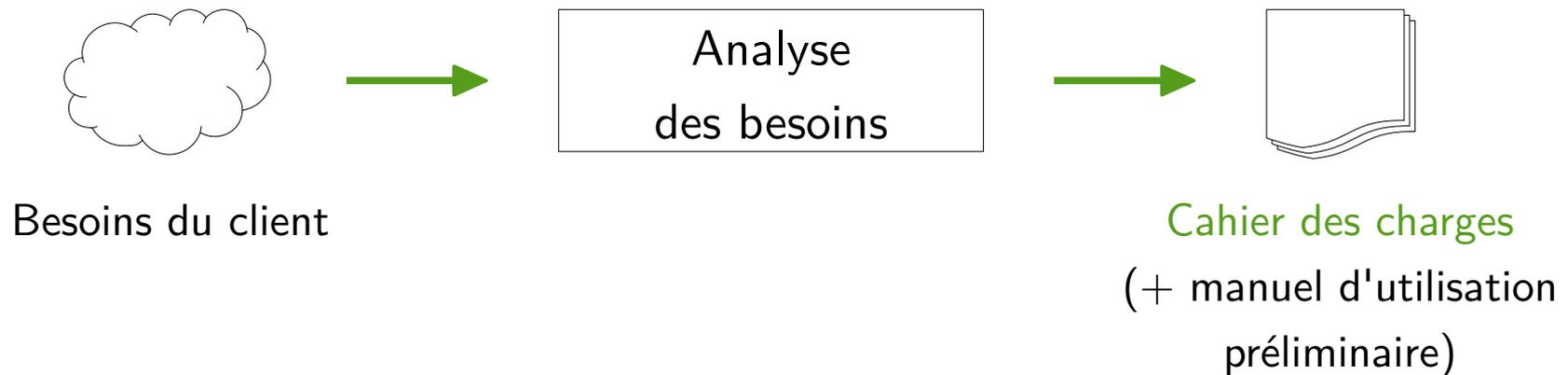
Pour chaque activité : Utilisation et production de documents

Analyse des besoins



Objectif : Comprendre les besoins du client

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



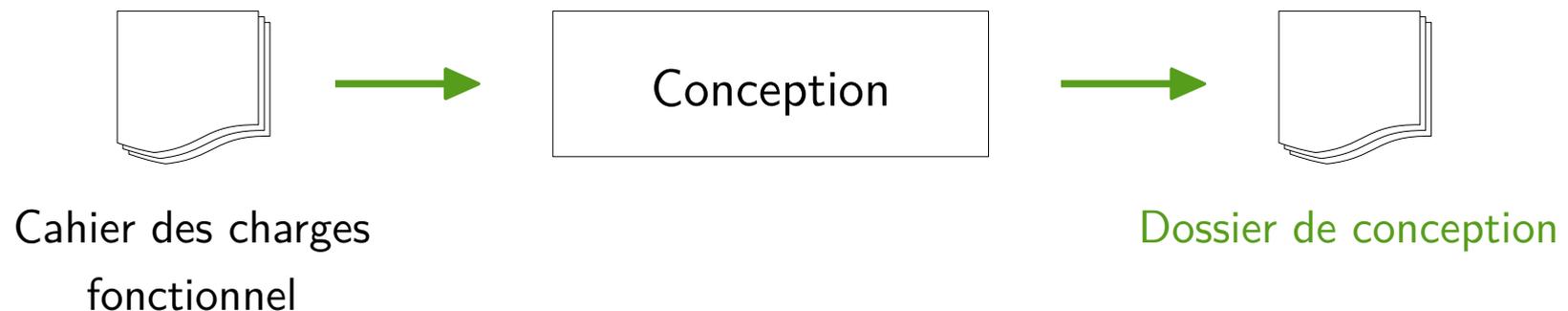
Spécification

-  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)



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...)

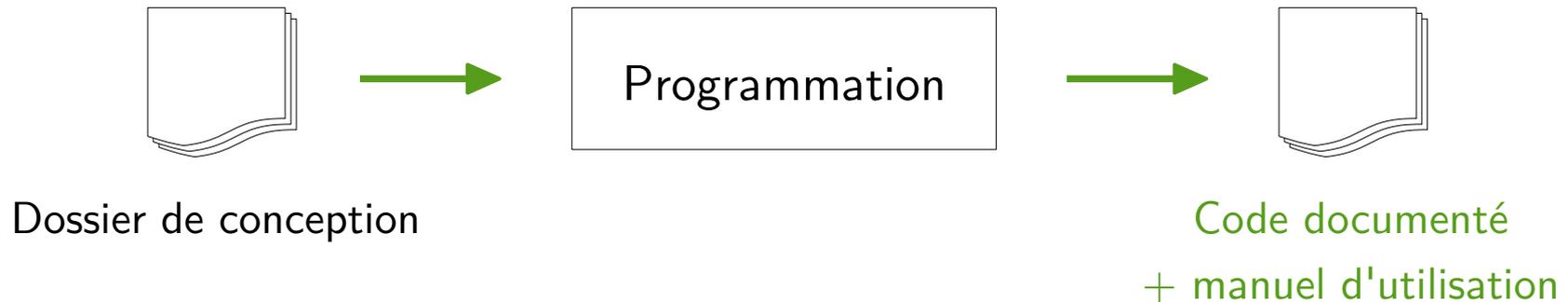


Programmation



Objectif : **Implantation** de la solution conçue

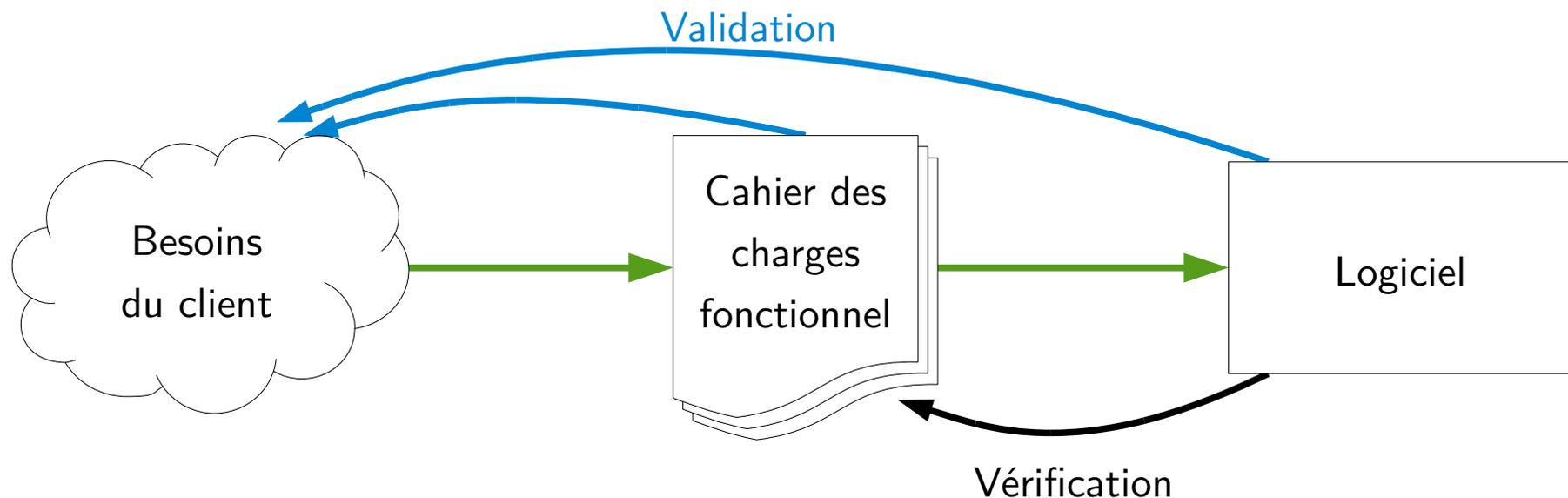
- Choix de l'environnement de développement, du/des langage(s) de programmation, de normes de développement...



Validation et vérification

Objectifs :

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



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 (vérification de modèles)

Démarche de test

Plan de test

- Description des **exigences de test** (couverture des exigences fonctionnelles et non fonctionnelles)
- Choix d'une **stratégie de test** et **planification** des tests

Cahier de tests

- Description des **cas de test** (couverture des exigences de test)
- Élaboration des **procédures de test**

Dossier de tests

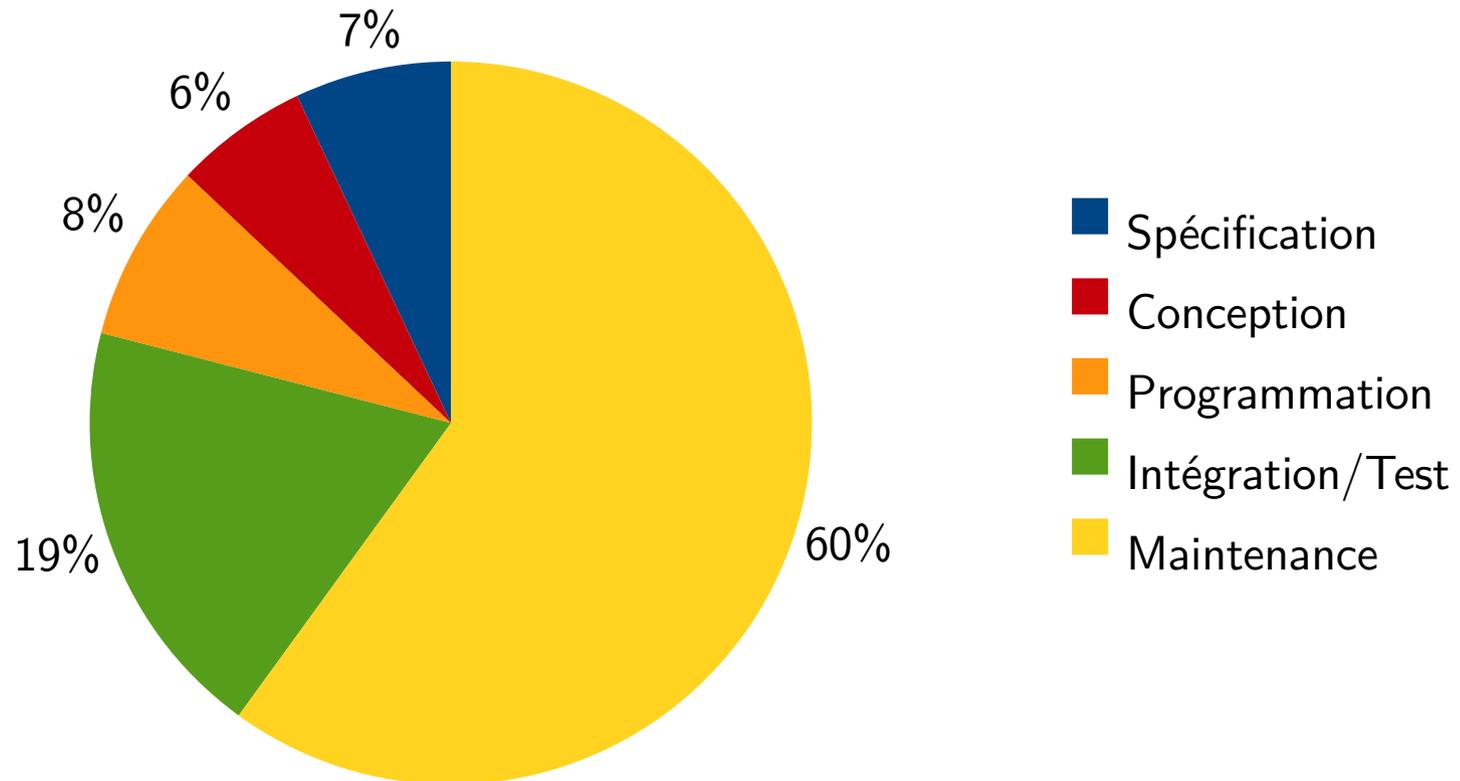
- Implémentation et **exécution des tests**
- Évaluation de l'exécution des tests et **analyse des résultats**
- **Rapport de test**

Maintenance

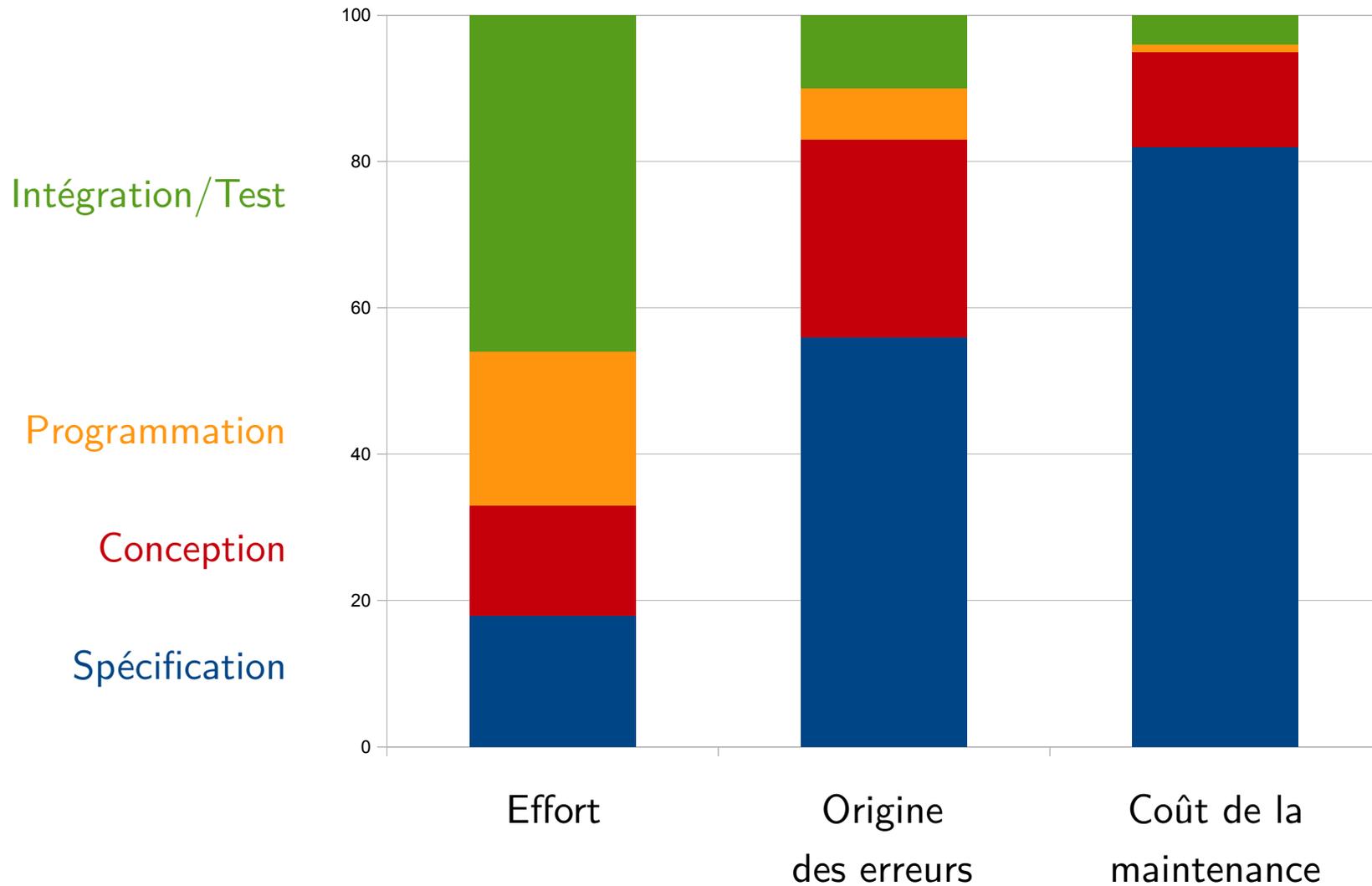
Types de maintenance :

- **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

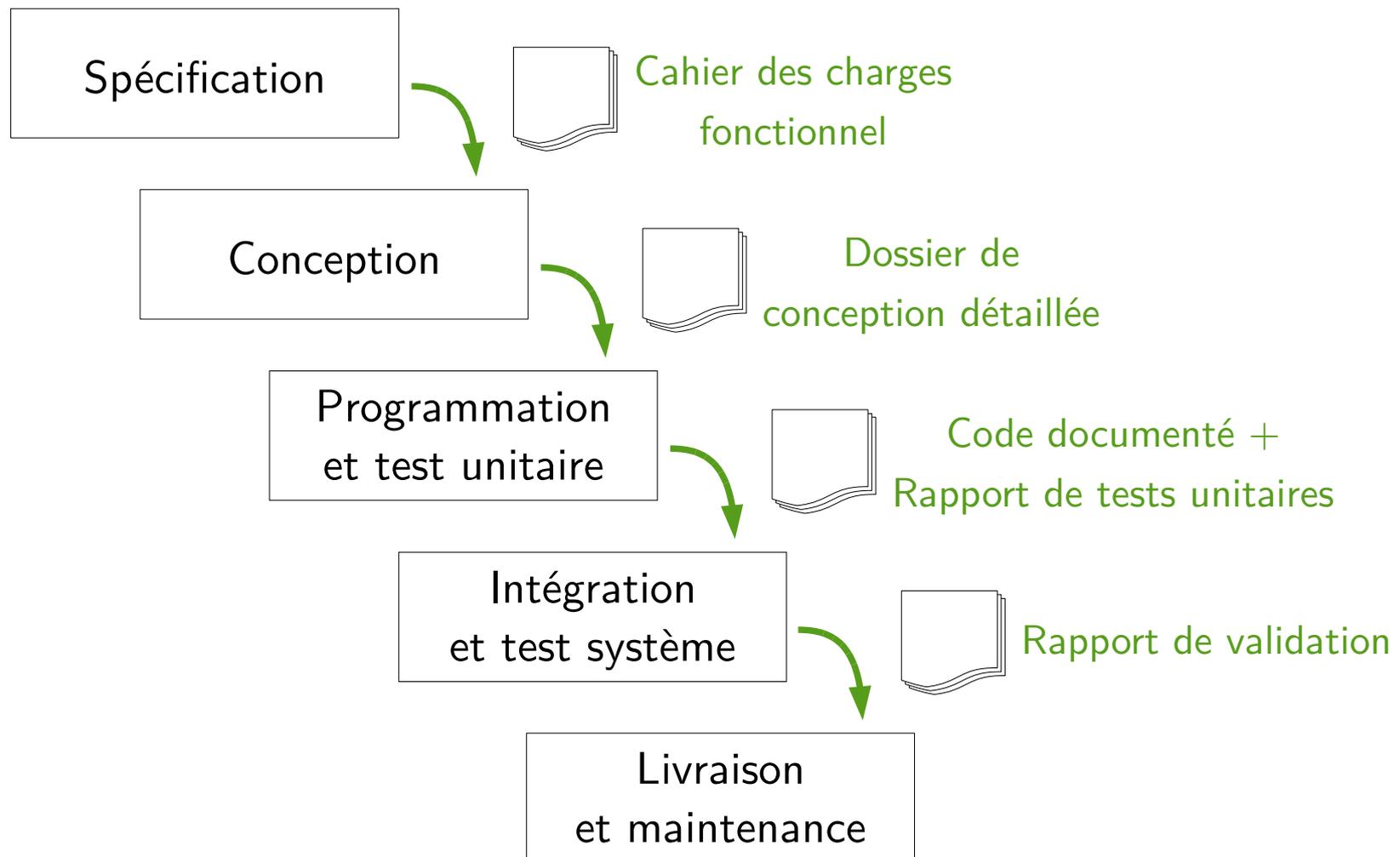


Rapport effort/erreur/coût



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

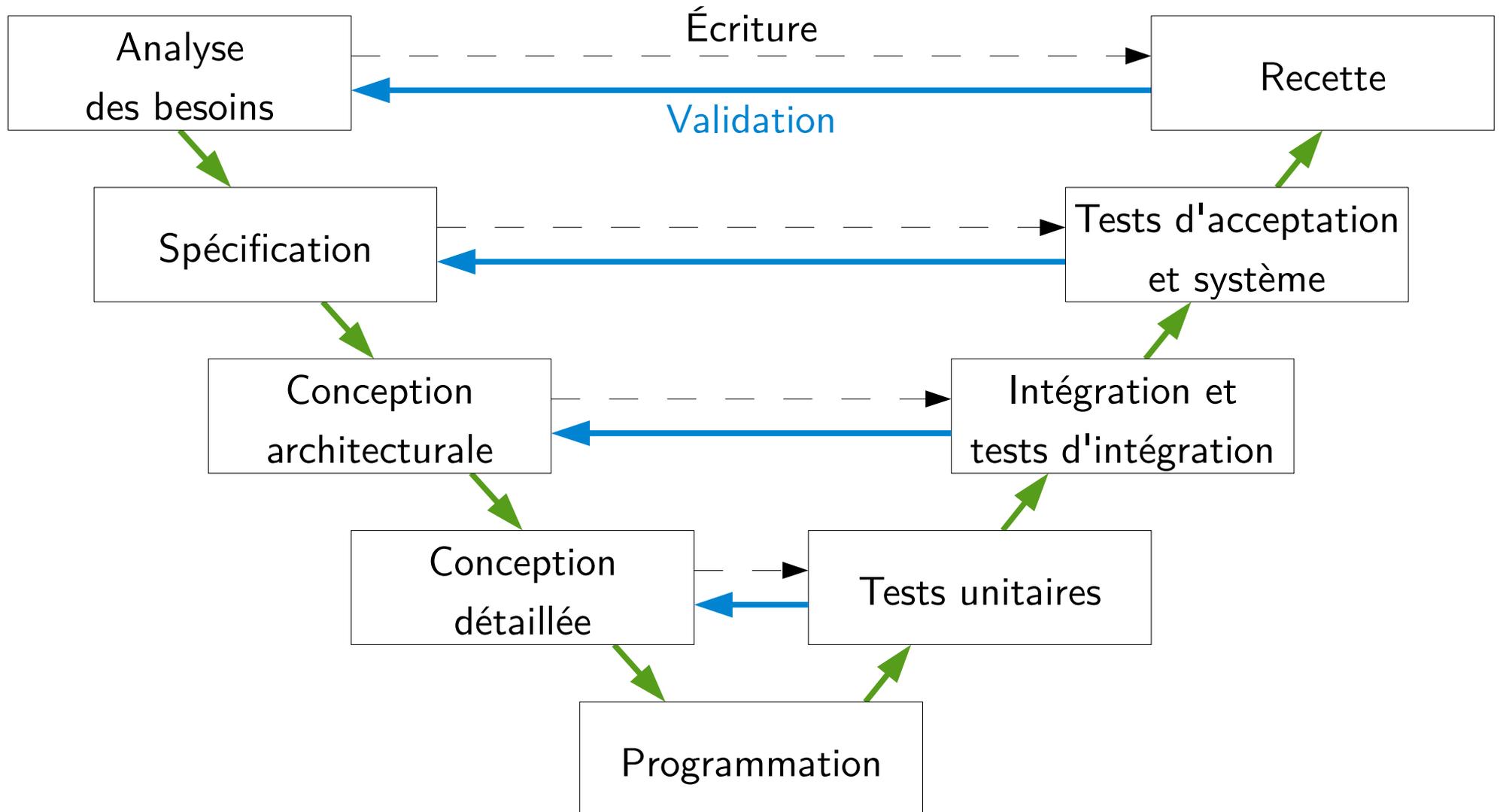
Caractéristiques :

- 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

Inconvénients : les mêmes que le cycle en cascade

- **Processus lourd**, difficile de revenir en arrière
- Nécessite des **spécifications précises et stables**
- Beaucoup de documentation

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)

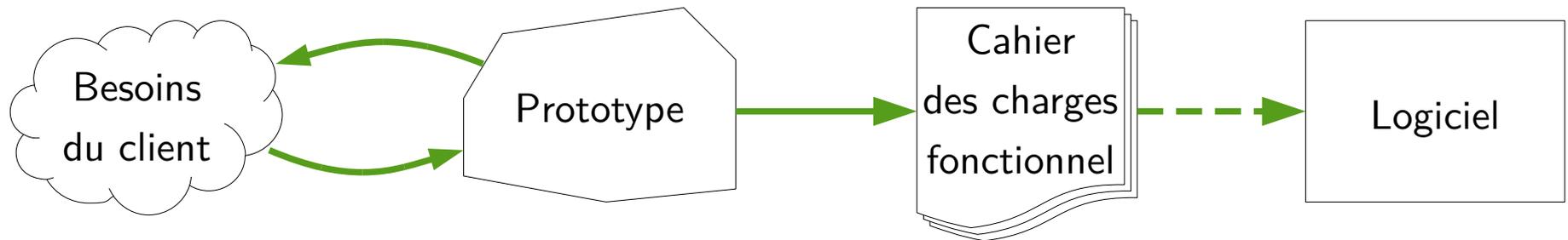
Test d'acceptation et système : test du **système complet** par rapport à son cahier des charges

Recette : faite par le **client**, validation par rapport aux **besoins initiaux** (tests, validation des documents, conformité aux normes...)

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

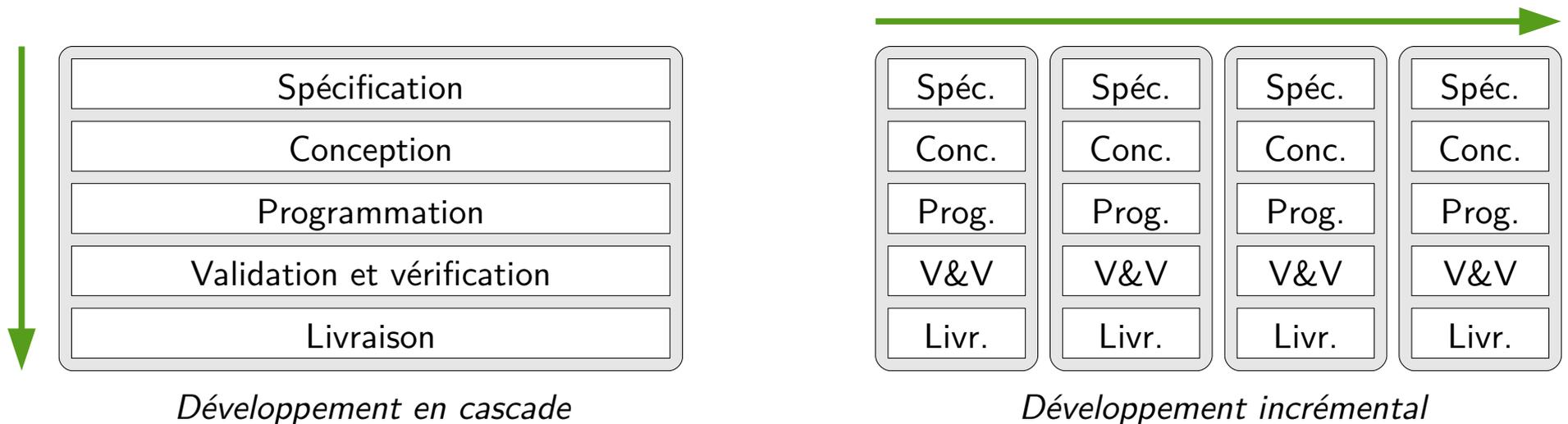


Avantage : Validation concrète des besoins, moins de risques d'erreur de spécification

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



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)

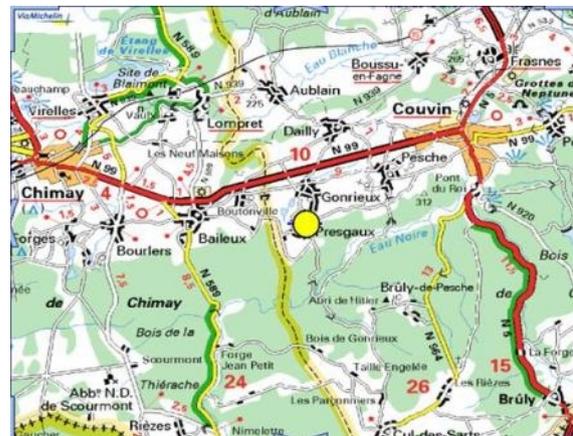
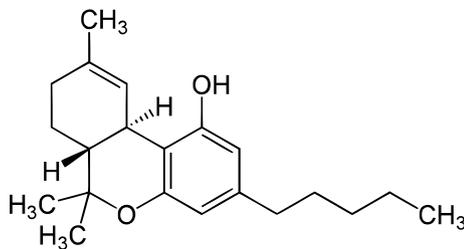
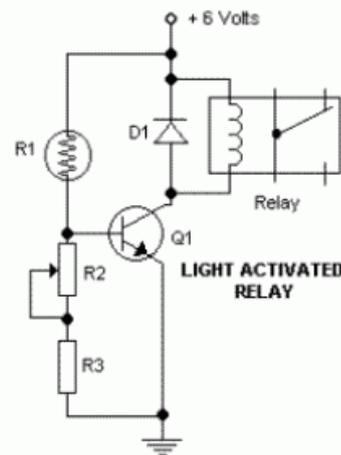
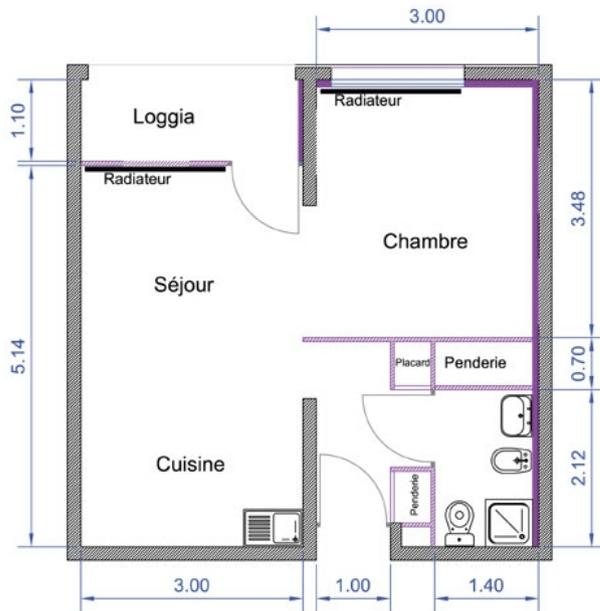
En génie logiciel :

- 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 **graphique, semi-formel**
- **Standard** très utilisé

Conception orientée objet

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

Conception orientée objet avec UML

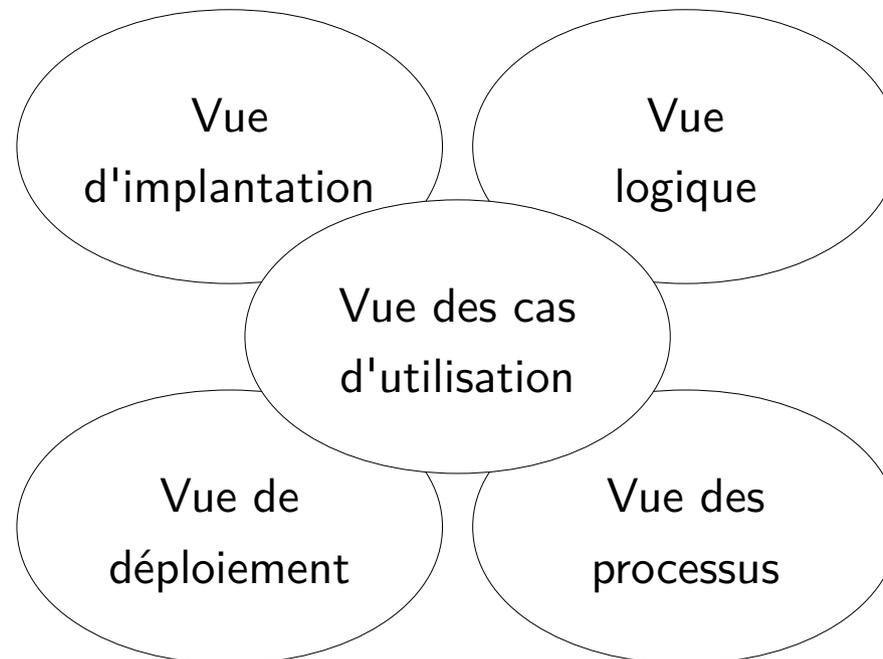
UML n'est pas une méthode de conception

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

Spécification

- 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** ou **états-transitions** : enchaînement d'actions représentant un **comportement** du logiciel

Conception

- 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