

# Génie logiciel avancé

## Introduction

### Génie logiciel, validation et vérification

Delphine Longuet

[delphine.longuet@lri.fr](mailto:delphine.longuet@lri.fr)

# Organisation du cours

Modalités de contrôle des connaissances :

- TP noté : semaine du 10 octobre
- Partiel : semaine du 24 octobre
- Examen : semaine du 12 décembre

Note du contrôle continu (CC) = 40 % TP + 60 % partiel

Note finale = 40 % CC + 60 % examen

Note de 2<sup>e</sup> session remplace la note finale (pas de report de CC)

Seuls les transparents du cours sont autorisés au partiel et aux examens

Page web du cours :

<http://www.lri.fr/~longuet/Enseignements/16-17/L3-GLA>

# Génie logiciel

## Définitions et processus de développement

# 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**, **rigoureuses** et **mesurables** 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

# 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 météorologique),  
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

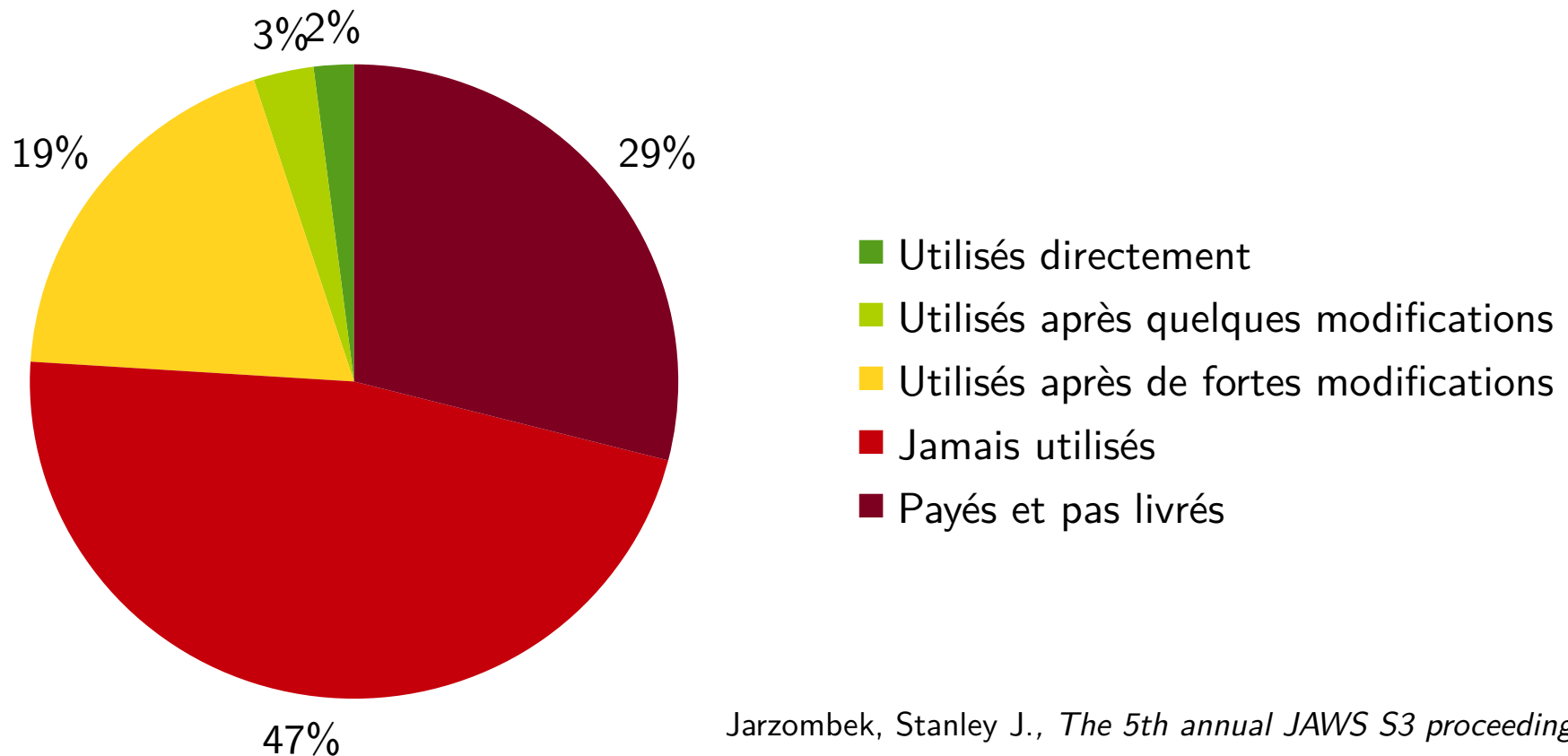
# Crise du logiciel

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

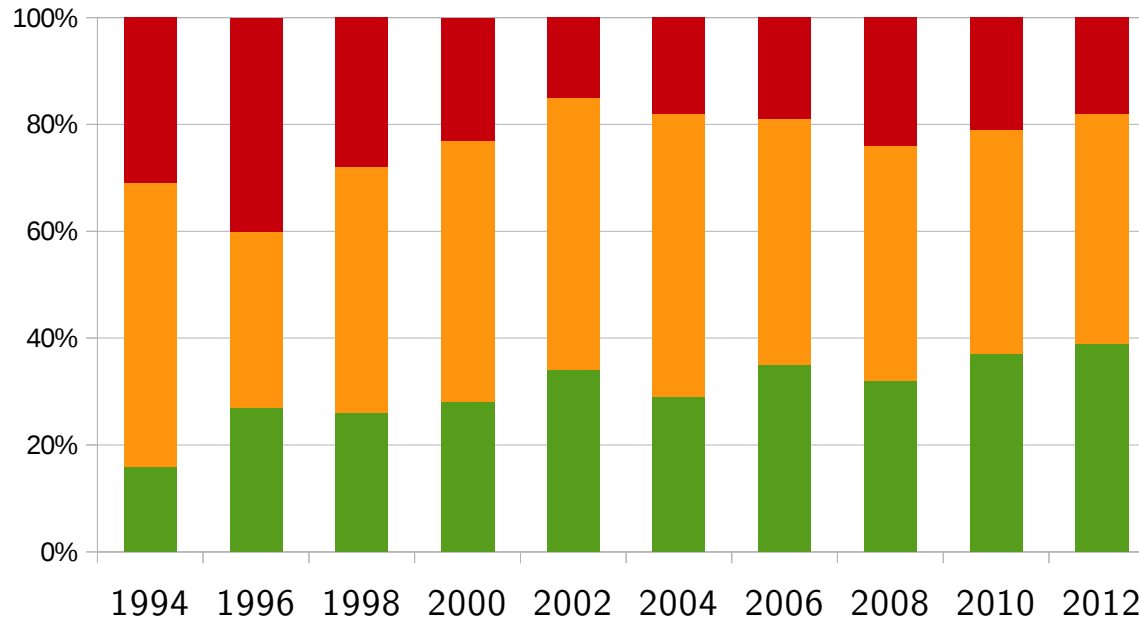


Jarzombek, Stanley J., *The 5th annual JAWS S3 proceedings*, 1999



# Étude du Standish group

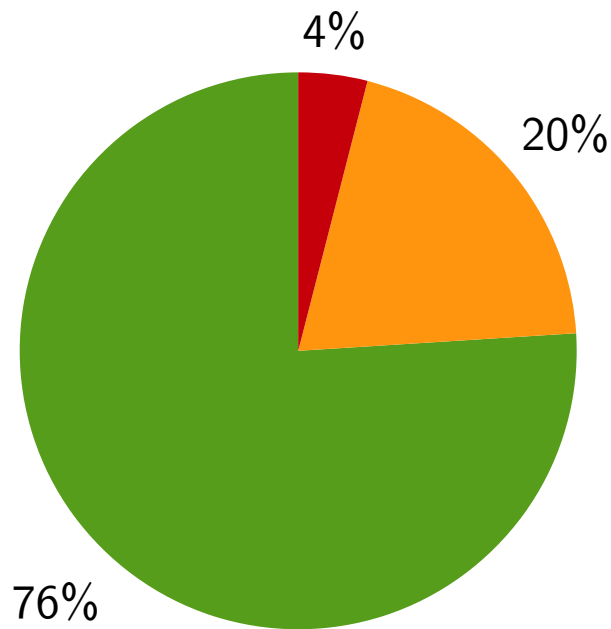
Enquête sur des milliers de projets, de toutes tailles et de tous secteurs



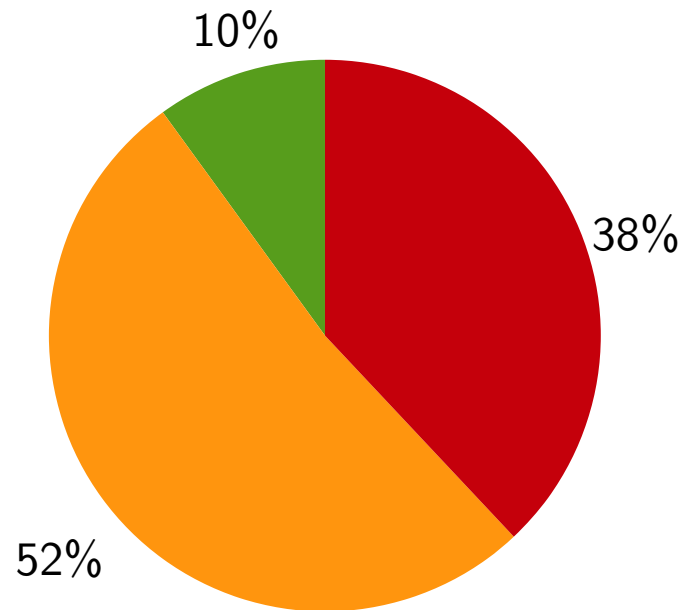
Standish group, *Chaos Manifesto 2013 - Think Big, Act Small*, 2013

- **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

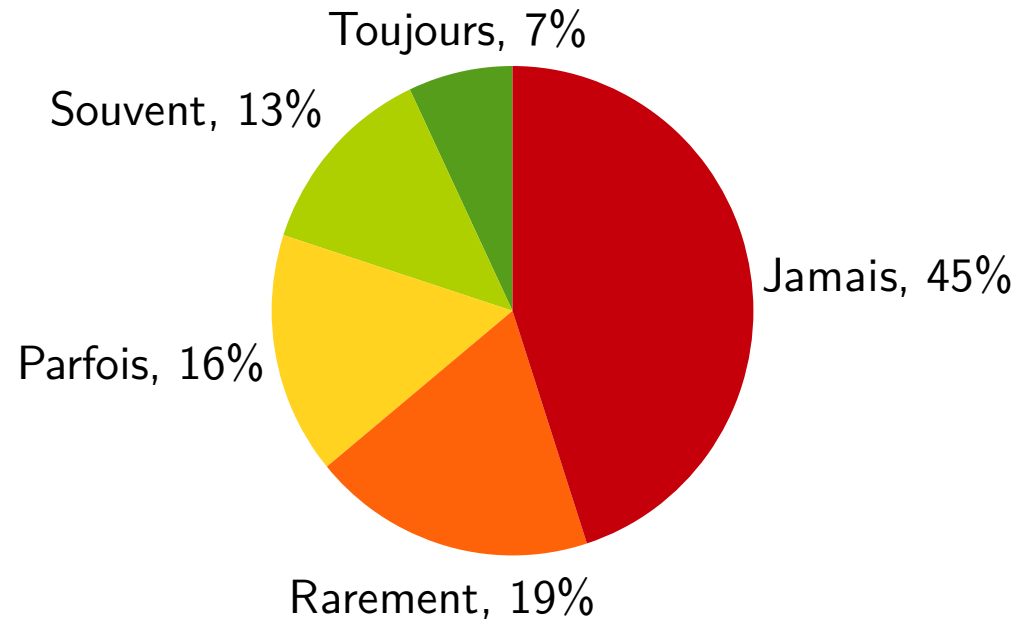


Grands projets  
budget  $\geq$  \$10 millions

- 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

« La **satisfaction** du client et la **valeur** du produit sont plus grandes lorsque les **fonctionnalités** livrées sont bien **moins nombreuses** que demandé et ne remplissent que les **besoins évidents**. »

Standish group, *Chaos Report 2015*, 2015

# Raisons de la faible qualité des logiciels

## Tâche complexe :

- Taille et complexité des logiciels
- Taille des équipes de conception/développement

## Manque de méthodes et de rigueur :

- Manque de méthodes de conception
- Négligence et manque de méthodes et d'outils des phases de validation/vérification

## Mauvaise compréhension des besoins :

- Négligence de la phase d'analyse des besoins du client
- Manque d'implication du client dans le processus

# Raisons de la faible qualité des logiciels



Ce que le client a expliqué



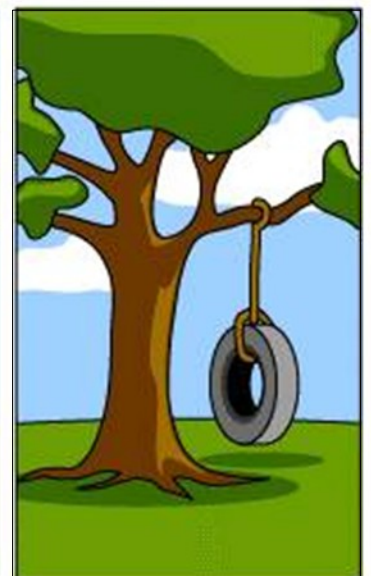
Ce que le chef de projet a compris



Ce que l'analyste a proposé



Ce que le programmeur a écrit



Ce dont le client avait vraiment besoin

# Raisons de la faible qualité des logiciels

## Difficultés spécifiques 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

# Importance de la qualité des logiciels

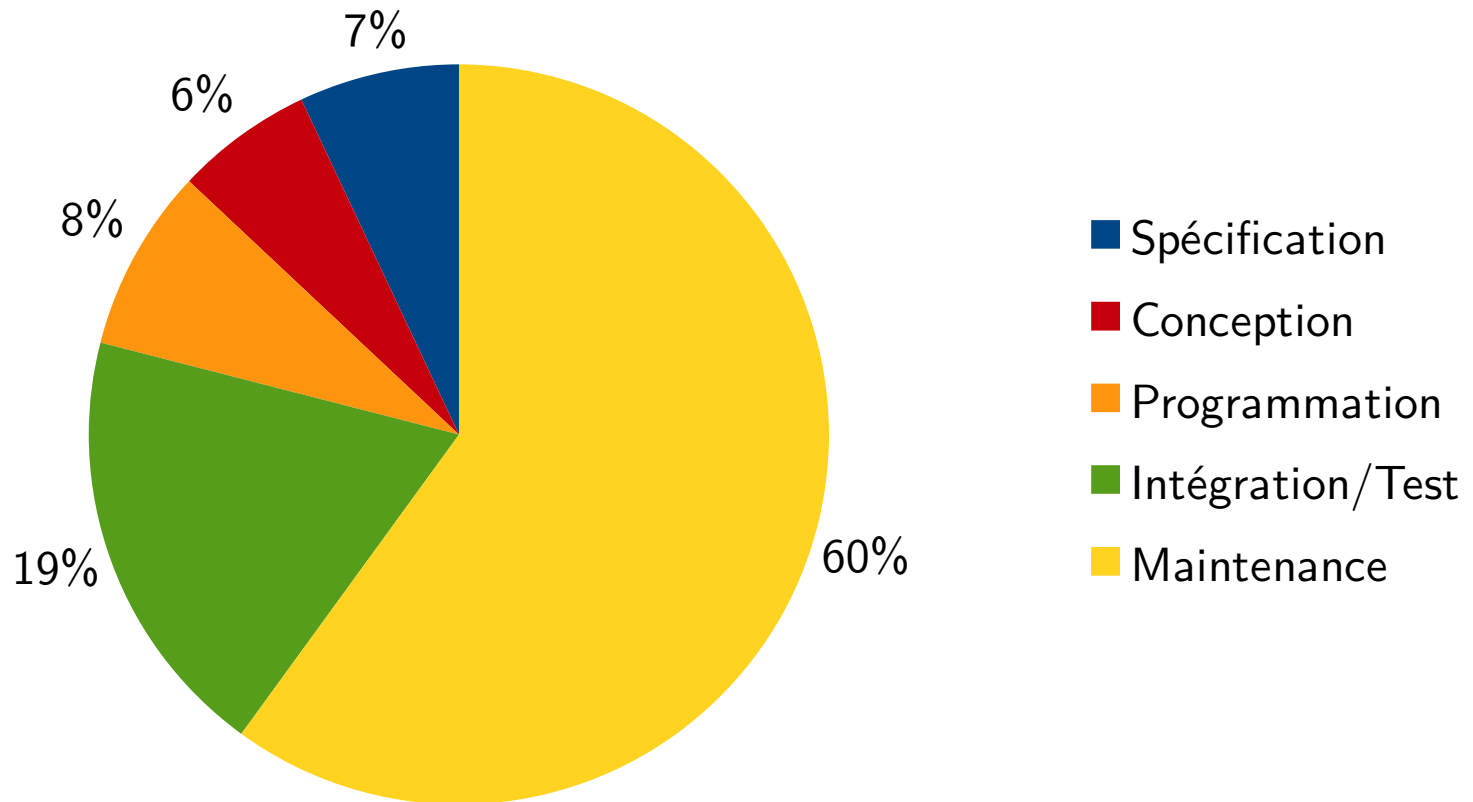
## Fiabilité, sûreté et sécurité des logiciels

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

## Raisons économiques : coût d'un bug

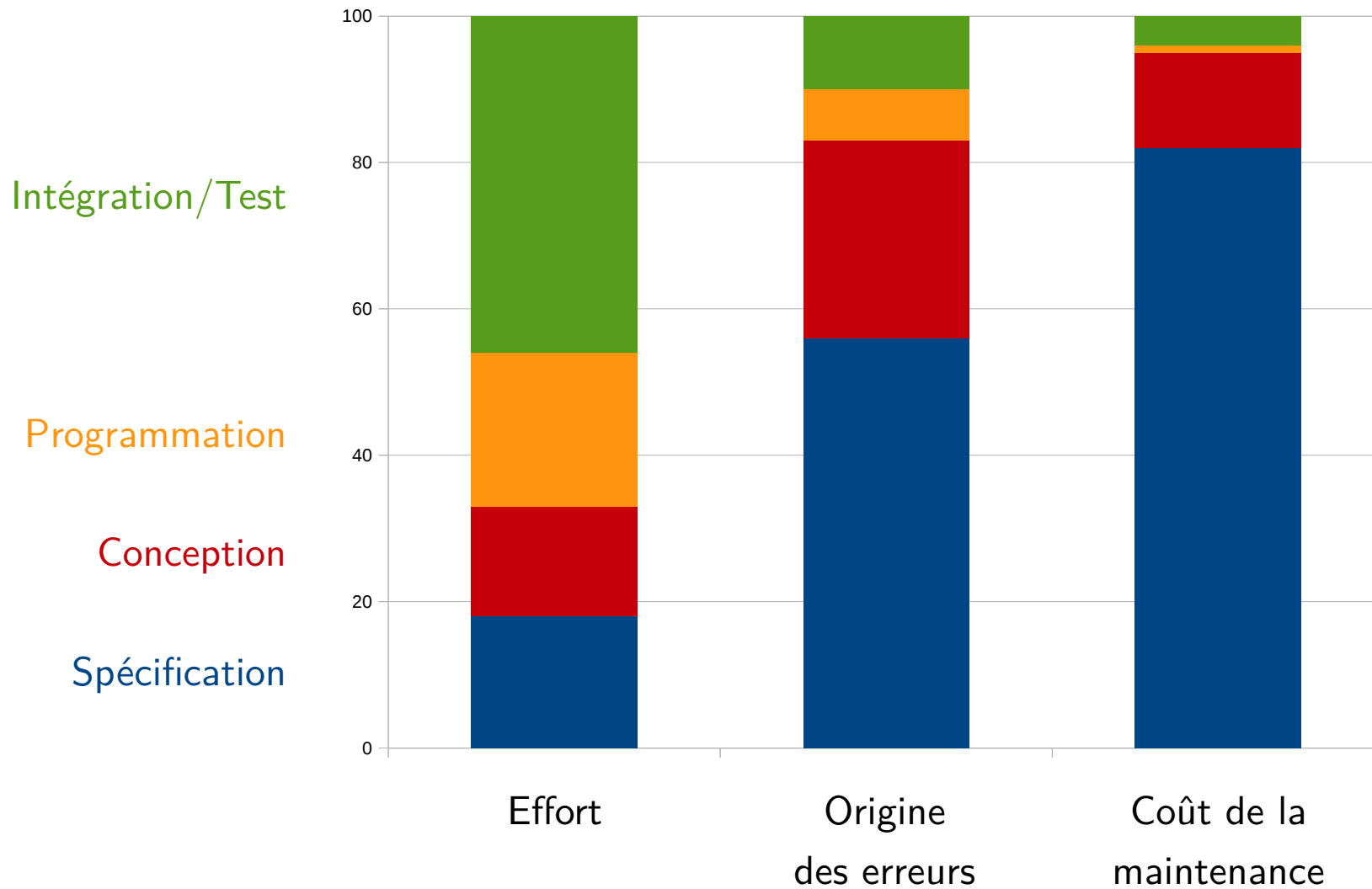
- coût de la correction, du rappel des appareils défectueux
- coût de l'impact sur l'image, de l'arrivée tardive sur le marché
- coût en vies, coût de l'impact écologique

# Répartition de l'effort





# Rapport effort/erreur/coût



# 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

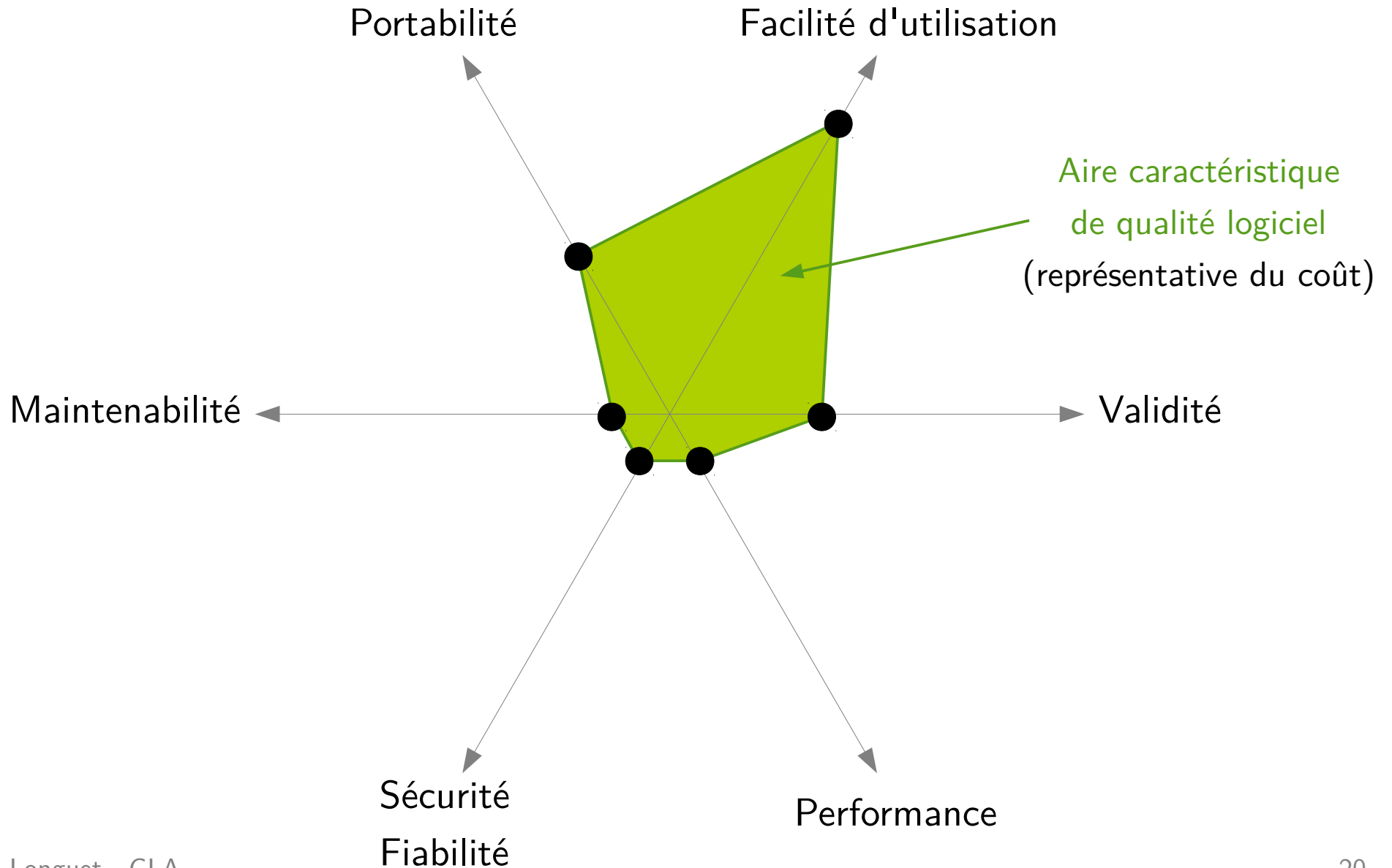
Génie civil, naval, aéronautique, mécanique, chimique...

# Qualité du logiciel

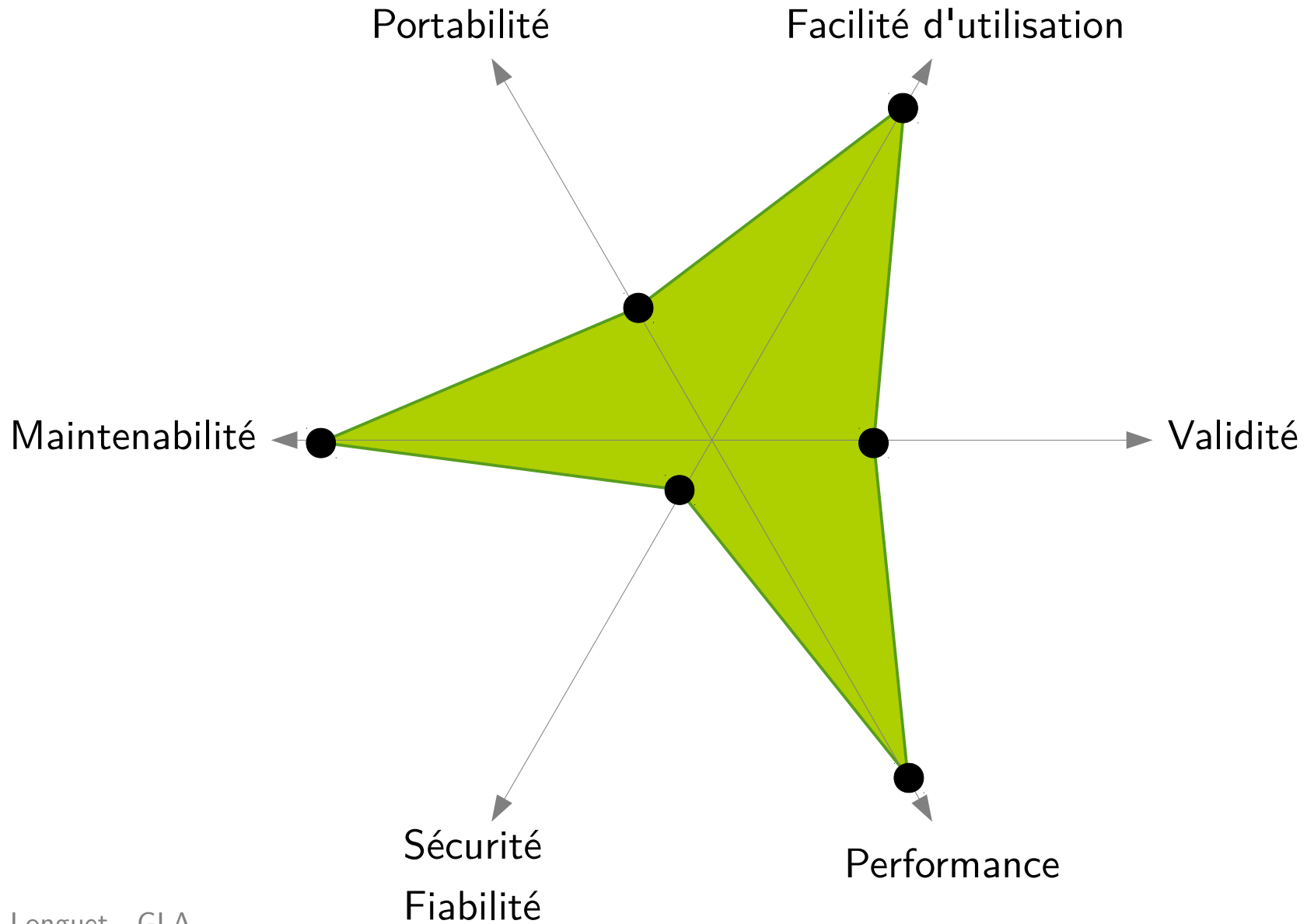
## Critères de qualité

- **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

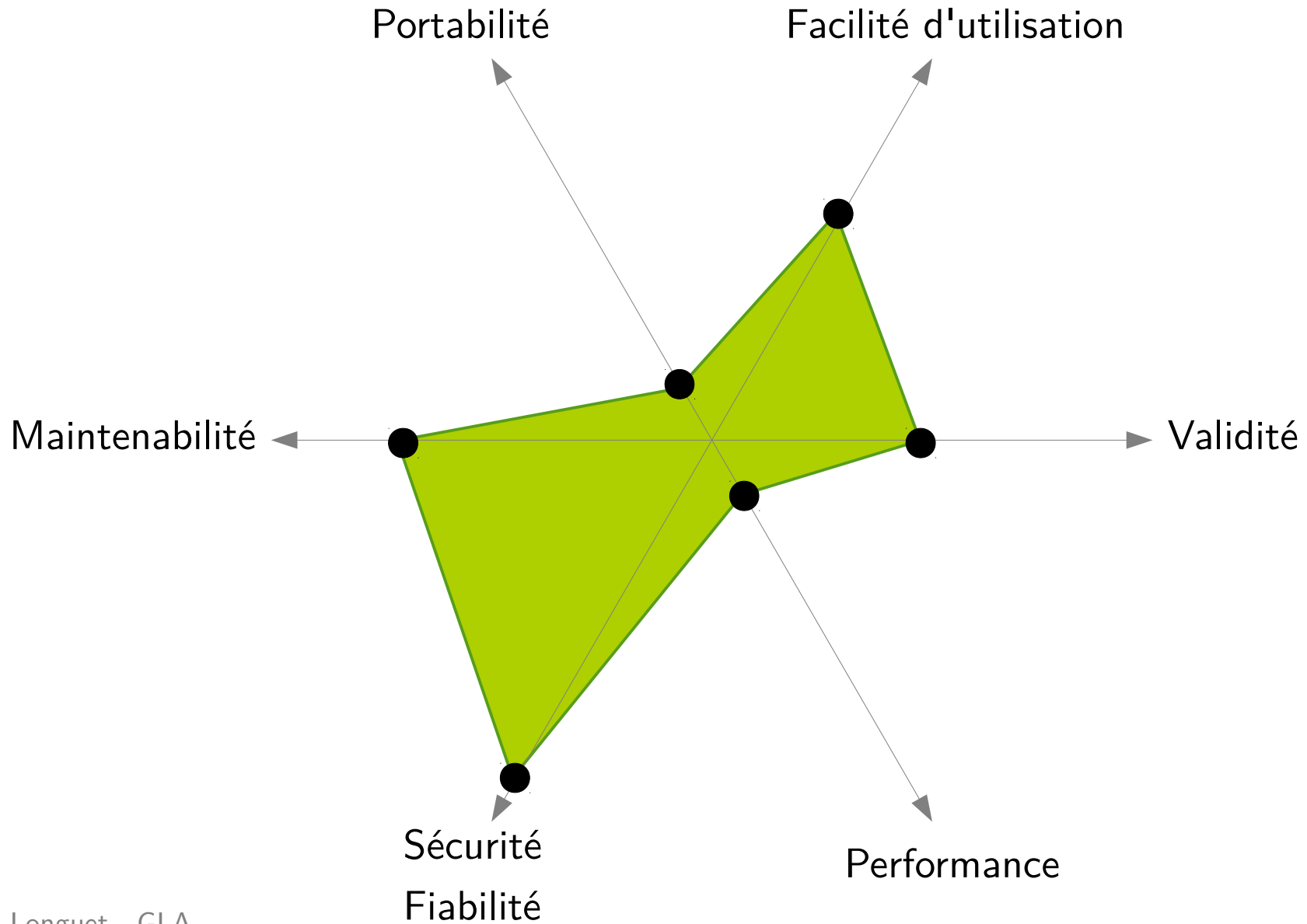
# Contrôleur de télécommande



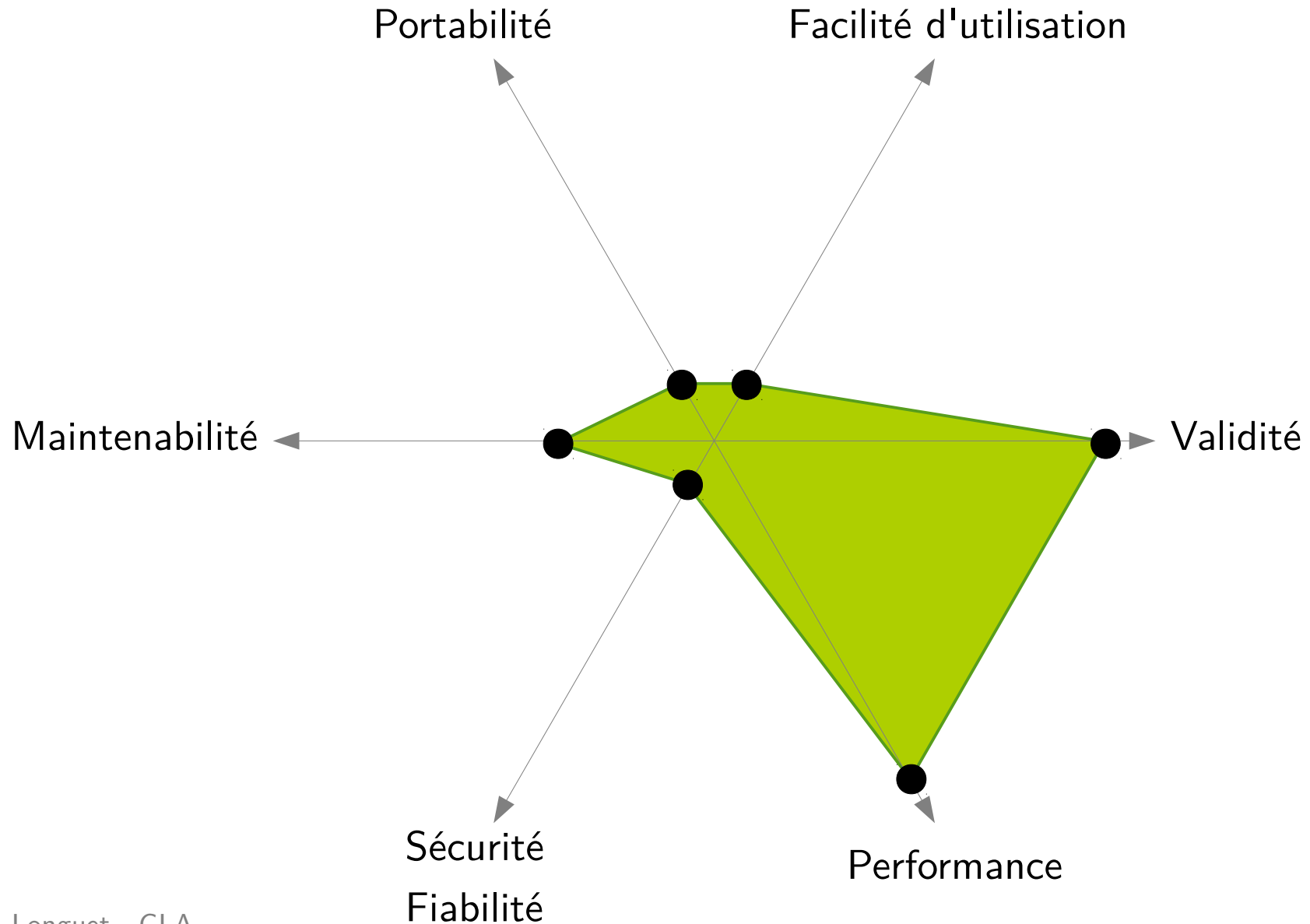
# Jeu vidéo



# Client mail



# Simulateur pour Météo France



# 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



# Activités du développement logiciel

**Analyse des besoins** : Comprendre les **besoins du client**

- Objectifs généraux, environnement du futur système, ressources disponibles, contraintes de performance...
- Fournie par le **client** (expert du domaine d'application, futur utilisateur...)

**Spécification** :

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

# Activités du développement logiciel

**Conception** : 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...)

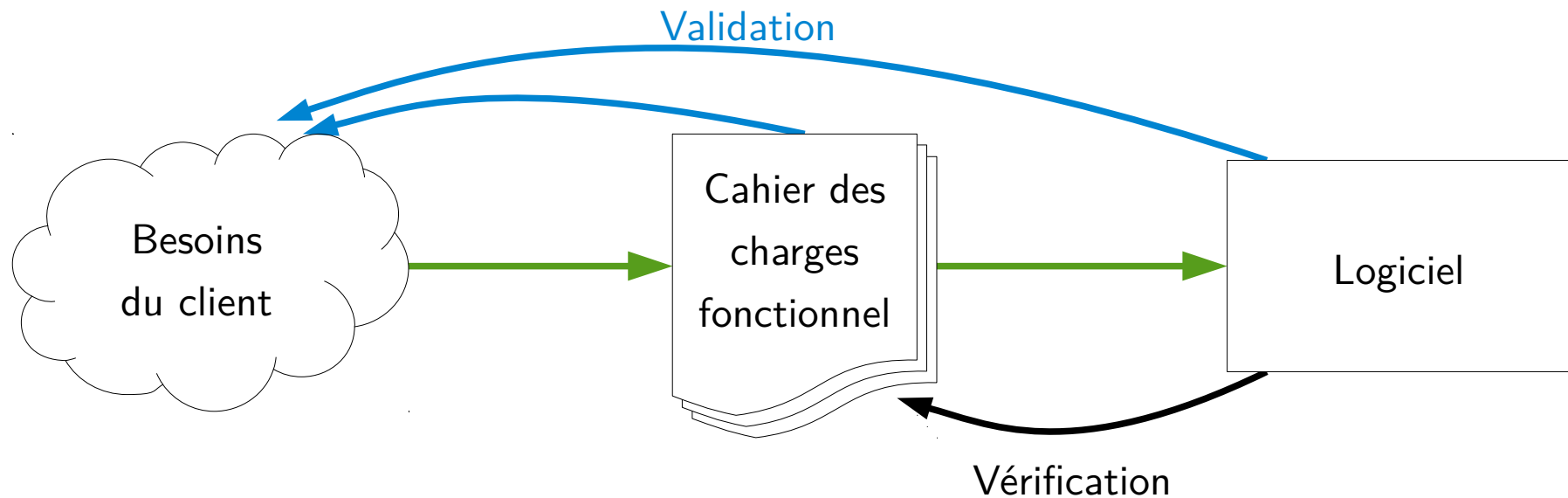
**Programmation** : **Implantation** de la solution conçue

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

# Validation et vérification

**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**



# Maintenance

## Types de maintenance :

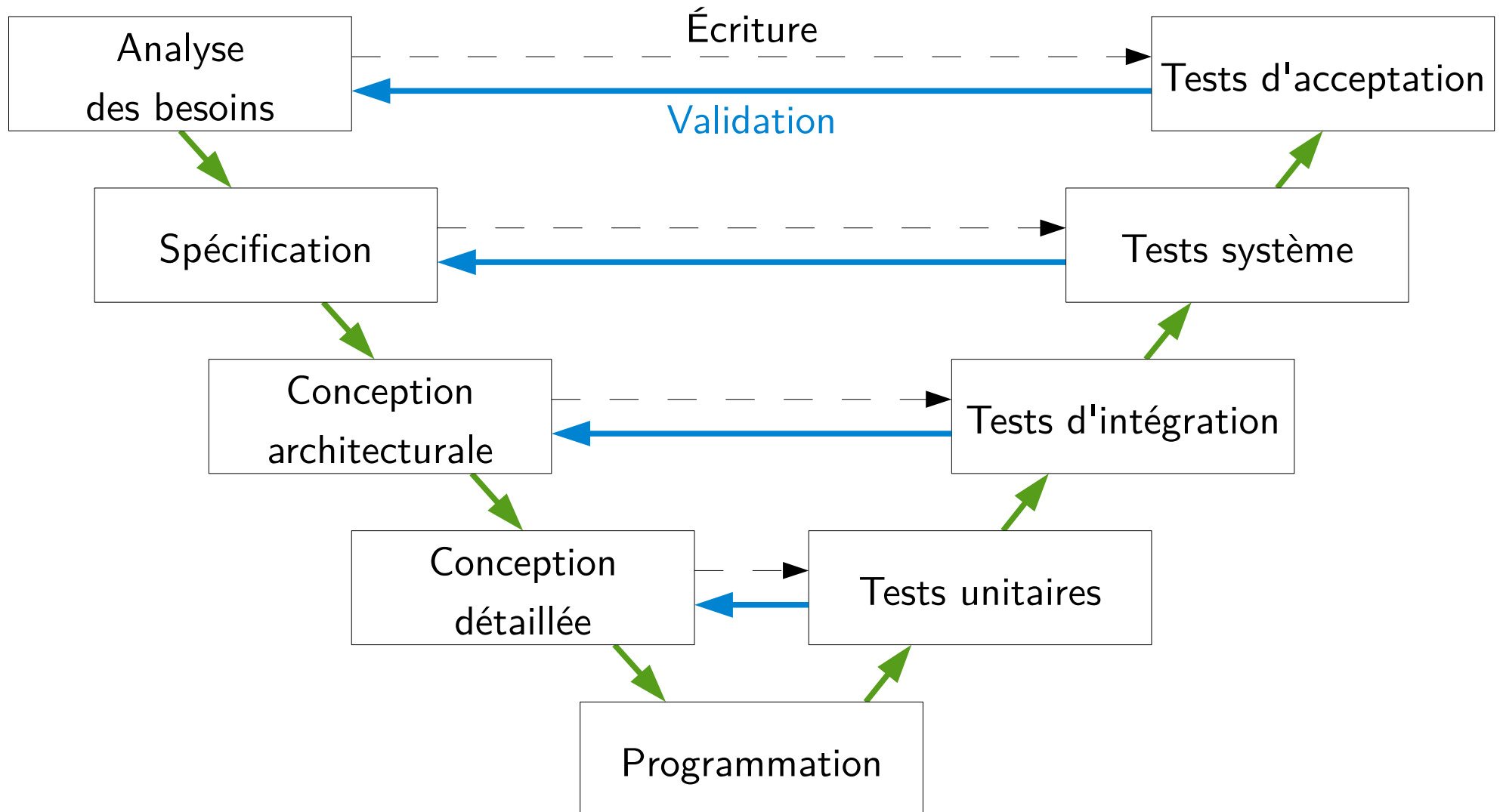
- **Correction** : identifier et corriger des erreurs trouvées après la livraison
- **Adaptation** : adapter le logiciel aux changements dans l'environnement (format des données, environnement d'exécution...)
- **Perfection** : améliorer la performance, ajouter des fonctionnalités, améliorer la maintenabilité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

# 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

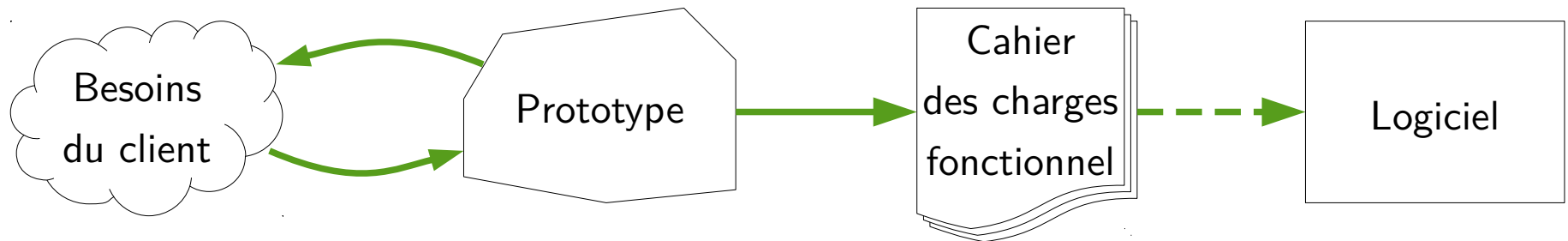
# Processus en V



# 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

# 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 de spécification et conception

# 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

# 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

## Dans ce cours :

- UML + langage mathématique orienté objet
- Langage de spécification de programmes

# Validation et vérification

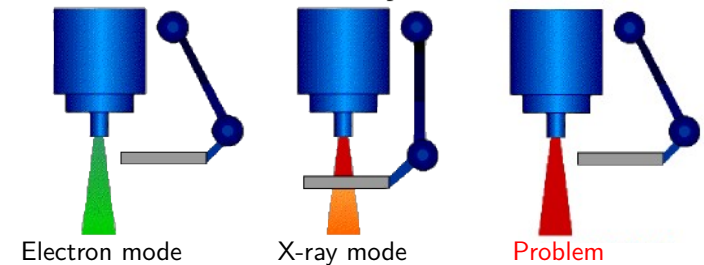
# Bugs célèbres

## 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



# Pourquoi des méthodes pour valider et vérifier ?

## Pour réduire le coût

- Vérification et validation =
  - environ 30% du développement d'un logiciel standard
  - plus de 50% du développement d'un logiciel critique
- Phase de test souvent plus longue que les phases de spécification, conception et implantation réunies

## Pour certifier la qualité

- Processus de certification, de normalisation
- Obligations légales

# Validation et vérification

**Validation** : assurer que les attentes du client sont satisfaites

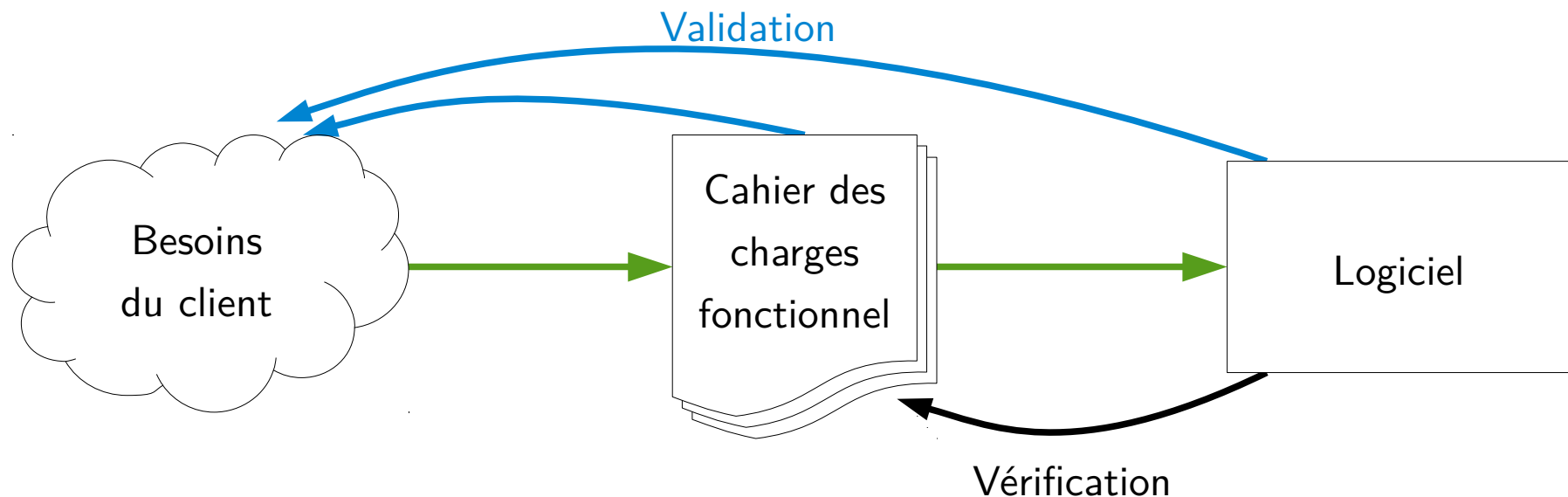
« *Are we building the right product ?* »

Assurance d'un certain niveau de confiance dans le système

**Vérification** : assurer que le système satisfait sa spécification

« *Are we building the product right ?* »

Preuve formelle de propriétés sur un modèle du système



# Méthodes de validation et vérification

## Méthodes de validation :

- Revue de spécification, de code
- Prototypage rapide
- Développement de tests exécutables (*extreme programming*)
- Recette (tests d'acceptation faits par le client)

## Méthodes de vérification :

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

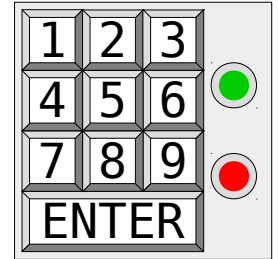
# Quelques méthodes de V&V

**Test** : 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)

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

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

# Exemple : test

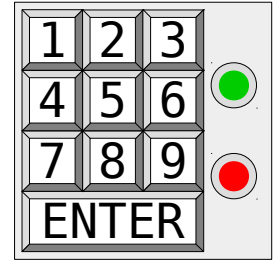


Code d'accès :

- 4 chiffres consécutifs dans l'ordre croissant
- Saisie des chiffres puis validation.
- Accès autorisé s'il existe une suite de 4 chiffres consécutifs dans la suite saisie.

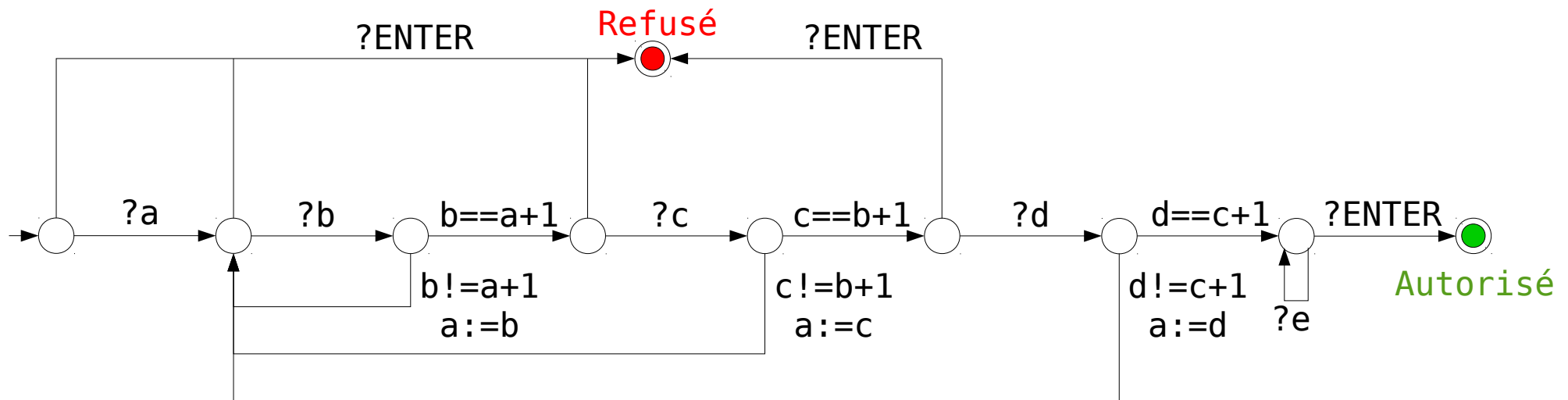
Objectif de test	Données d'entrée	Résultat attendu
4 chiffres consécutifs	1234 + ENTER	Autorisé
Avec des chiffres avant	846789 + ENTER	Autorisé
Avec des chiffres après	3456298 + ENTER	Autorisé
Avec des chiffres avant et après	32345698 + ENTER	Autorisé
Dans l'ordre décroissant	6543 + ENTER	Refusé
Pas consécutifs	2356 + ENTER	Refusé
Moins de 4 chiffres	345 + ENTER	Refusé
Vide	ENTER	Refusé
Oubli de ENTER	5678	Aucun

# Exemple : model-checking



Code d'accès :

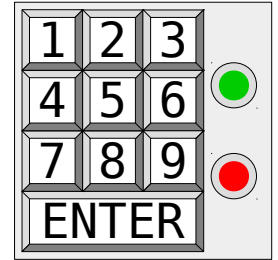
- 4 chiffres consécutifs dans l'ordre croissant
- Saisie des chiffres puis validation.
- Accès autorisé s'il existe une suite de 4 chiffres consécutifs dans la suite saisie.



Propriété à démontrer :  $\text{Autorisé} \Rightarrow a \neq 0 \wedge b = a + 1 \wedge c = a + 2 \wedge d = a + 3 \wedge \text{ENTER}$

(Variables :  $a, b, c, d, e = 0$  ;  $\text{ENTER} = \text{false}$ )

# Exemple : preuve



Code d'accès :

- 4 chiffres consécutifs dans l'ordre croissant
- Saisie des chiffres puis validation.
- Accès autorisé s'il existe une suite de 4 chiffres consécutifs dans la suite saisie.

```
/*@ requires \valid(t+(0..n-1)) && 1 <= n;
   @ ensures \result <==> \exists integer k; 0 <= k < n
               ==> \forall integer j; 0 < j <= 3 ==> t[k] == t[k-j]+j;
   @*/

int verifCode(int t[], int n) {
    int res = 0;
    int i = 1;
    /*@ loop invariant \forall integer j; 0 < j <= res ==> t[i-1] == t[i-1-j]+j;
       @ loop invariant 0 <= res <= 3 && 1 <= i <= n;
       @*/
    while(res < 3 && i < n) {
        if(t[i] == t[i-1]+1) {
            res = res+1; i = i+1;
        } else {
            res = 0; i = i+1;
        }
    }
    return(res == 3);
}
```



# Comparaison des méthodes de V&V

## Test :

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

## Model-checking :

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

## Preuve :

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

# Comparaison des méthodes de V&V

## Méthodes complémentaires :

- **Test** non exhaustif mais facile à mettre en œuvre
- **Preuve** exhaustive mais très technique
- **Model-checking** exhaustif et moins technique que la preuve

## Mais :

- Preuve et model-checking **limités par la taille du système** et vérifient des propriétés sur un **modèle du système** (distance entre le modèle et le système réel ?)
- Test repose sur l'**exécution du système réel**, quelle que soit sa taille et sa complexité

# Plan du cours

- I. Rappels d'UML, langage de contraintes et spécification d'opérations
- II. Test de logiciels : test fonctionnel et test structurel (2 TP)
- III. Preuve de programmes (1 TP)