Génie Logiciel

RAZAFIMAHATRATRA Hajarisena
Docteur en Informatique

Année: 2025

Définition du logiciel

Un **logiciel** (***software***) est l'ensemble des *programmes*, des *procédures* et des *documentations* nécessaires au fonctionnement d'un système informatique.

Critères de qualité du logiciel

❖**Exactitude**

Aptitude d'un logiciel à fournir des résultats voulus dans les conditions normales d'utilisation.

❖**Robustesse**

Aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation.

Exemple : IP (Internet Protocol). Le succès d'Internet est du à la robustesse du protocole de communication utilisé. Un datagramme IP arrive à destination même si un réseau local est inaccessible.

Critères de qualité du logiciel

❖**Extensibilité**

Facilité avec laquelle un programme pourra être adapté pour faire face à une évolution des besoins de l'utilisateur.

❖**Réutilisabilité**

Possibilité d'utiliser certaines parties du logiciel pour développer un autre logiciel répondants à d'autres besoins. Cette notion est souvent relié à l'orienté objet où une classe générale sera facilement réutilisable.

Critères de qualité du logiciel

❖**Portabilité**
Facilité avec laquelle on peut exploiter un logiciel dans différentes implémentations. Exemple Windows 95 ou Linux.

❖**Efficience**
Temps d'exécution, taille mémoire…

Caractéristiques du logiciel - I

- le logiciel est **facile à reproduire**
  - Tout le coût se trouve dans le développement
- le logiciel est **immatériel** et **invisible**
  - On ne peut l'observer qu'en l'utilisant
  - La qualité n'est pas vraiment apparente
  - Difficile d'estimer l'effort de développement
- développement **difficile à automatiser**
  - Beaucoup de main-d'œuvre nécessaire …

Caractéristiques du logiciel - 2

- le logiciel ne s'use pas, mais il **vieillit**

Détérioration suite aux changements :

- complexification indue
- duplication de code
- introduction d'erreurs

Mal conçu au départ :

- inflexibilité
- manque de modularité
- documentation insuffisante

Evolution du matériel

Différentes catégories de logiciels

- **sur-mesure** : pour un client spécifique
- **générique** : vendu sur un marché

Différents types de logiciels

- système d'**information** et d'**aide à la décision**
- logiciel **temps-réel**
- logiciel **embarqué**
- logiciel **distribué**

Problématique historique du logiciel

« D'une part le logiciel n'est **pas fiable**, d'autre part il est incroyablement difficile de réaliser dans les **délais prévus** des logiciels **satisfaisant leur cahier des charges** »

Le logiciel n'est pas fiable …

- Quelques **exemples célèbres** :
  - 1ère sonde **Mariner** vers Vénus

  → perdue dans l'espace *(erreur Fortran)*
  - **navette spatiale**

  → lancement retardé *(problème logiciel)*
  - **ATT**

  → appels longue distance suspendus *(changement de version)*
  - avion **F16**

  → retourné à l'équateur *(non prise en compte hémisphère sud)*
  - bug de l'**an 2000**

- **Tout système** comporte des **bugs** …

Délais et exigences non satisfaits …

- Quelques **exemples célèbres** :
  - **OS 360** d'IBM

  → en retard, dépassement mémoire et prix, erreurs
  - **Aéroport de Denver** (système de livraison des bagages)

  → 1 an de retard
  - **SNCF** (système Socrate)

  → difficultés importantes

Abandons …

- Quelques **exemples célèbres** :
  - **EDF** (contrôle-commande centrales 1400 MWatts)

    → renonce après plusieurs années d'efforts
  - **bourse de Londres** (projet d'informatisation)

    → abandon : 4 années de travail + 100 ML perdus
  - **Etats-Unis** (système de trafic aérien)

    → abandon

- La non rencontre des exigences et les dépassements de délais sont **fréquents** : 300 à 400 %

Naissance d'une nouvelle discipline

Historique

- Problématique apparue **dès les années 1960**
- **Conférence** parrainée par l'**OTAN** (**1968**)
- Naissance du « **Génie Logiciel** » *(software engineering)*

Naissance d'une nouvelle discipline

Objectif

- Démarche de développement **ingénierique**
- **Principes**, **méthodes**, **outils**
- Techniques de fabrication assurant :
  - respect des exigences
  - respect de la qualité
  - respect des délais et des coûts

Définition du « Génie Logiciel »

Processus visant :

- la **résolution** de **problèmes posés par un client**
- par le développement **systématique** et l'**évolution**
- de systèmes logiciels de **grande taille** et de **haute qualité**
- en respectant les **contraintes** de coûts, de temps, et autres

Résolution de problèmes posés par un client …

- **identifier** et **comprendre** le problème à résoudre
- solution **utile** résolvant un problème **concret**
- la solution peut être de ne **rien développer** !

Développement systématique et évolution …

- techniques **maîtrisées**, **organisation**, **rigueur**
- bonnes pratiques **standardisées** (IEEE, ISO)
- la plupart des projets consistent en une **évolution**

Systèmes de grande taille et haute qualité …

- travail en **équipe(s)**
- bonne **coordination** essentielle
- principal défi : **subdiviser** et **recomposer** harmonieusement
- produit final : **critères de qualités** bien établis

Respectant les contraintes …

- les ressources sont **limitées** (temps, argent, …)
- le bénéfice doit être **supérieur** aux coûts
- la productivité doit rester **concurrentielle**
- mauvaise estimation ⇒ échec

Transition …

Risques majeurs du développement de logiciels :

- ne pas remplir les exigences du client
- produire un logiciel non fiable
- dépasser les délais, les coûts et autres

contraintes

Méthodes du Génie Logiciel

Méthodologies de développement

Introduction

Comme pour tout produit manufacturé complexe :

- on **décompose** la production en « **phases** »
- l'ensemble des phases constitue un « **cycle de vie** »
- les phases font apparaître des **activités** clés

Activités du développement de logiciel

- Analyse des besoins
- Spécification
- Conception
- Programmation
- Intégration
- Vérification et validation

Analyse des besoins

- **But :**
  - → déterminer **ce que doit faire** (et ne pas faire) **le logiciel**
  - → déterminer les **ressources**, les **contraintes**

- **Caractéristiques :**
  - → parler **métier** et non info
  - → entretiens, questionnaires
  - → **observation** de l'existant, **étude** de situations similaires

- **Résultat :** ensemble de documents
  - → **rôle** du système
  - → future **utilisation**
  - → aspects de l'**environnement**
  - → (parfois) un manuel d'utilisation préliminaire

Spécification

- **But :**

→ établir une 1ère description du futur système
→ consigner dans un document qui fait référence

- **Données :**

→ résultats de l'analyse des besoins + faisabilité informatique

- **Résultat :** Spécification Technique de Besoin (STB)

→ **ce que fait** le logiciel, mais **pas comment**

- **Remarques :**

→ pas de choix d'implémentation
→ (parfois) un manuel de référence préliminaire

Conception

- **But :**

→ décrire **comment** le logiciel est construit
→ décider **comment** utiliser la techno. pour répondre aux besoins

- **Travail :**

→ enrichir la description de **détails d'implémentation**
→ pour aboutir à une description **très proche** d'un programme

- **2 étapes :**

→ **conception architecturale**
→ **conception détaillée**

Conception architecturale

- **But :** décomposer le logiciel en composants

→ mettre au point l'**architecture** du système

→ définir les **sous-systèmes** et leurs **interactions**

→ concevoir les **interfaces** entre composants

- **Résultat :**

→ **description** de l'architecture globale du logiciel

→ **spécification** des divers composants

Conception détaillée

- **But :** élaborer les éléments internes de chaque sous-syst.

- **Résultat :**

  → pour **chaque composant**, description des

  → **structures de données**, **algorithmes**

- **Remarque :**

  → si la **conception** est **possible**, la **faisabilité** est **démontrée**

  → sinon, la **spécification** est **remise en cause**

Programmation

- **But :**

→ passer des structures de données et algorithmes à un **ensemble de programmes**

- **Résultat :**

→ ensemble de **programmes**
→ ensemble de **bibliothèques** / **modules**
→ **documentés** (commentaires)

- **Remarque :**

→ activité la mieux **maîtrisée** et **outillée** (parfois automatisée)

Gestion de configurations et Intégration

- **Gestion de configurations :**

  → **gérer les composants** logiciels (programmes, bibliothèques, …)

  → **maîtriser leur évolution** et leurs mises à jour

- **Intégration :**

  → **assembler** les composants pour obtenir un système exécutable

Vérification

- **But :** vérifier par rapport aux spécifications
  → **vérifier** que les descriptions successives
  → **(**et *in fine* le logiciel) **satisfont la STB**

- **Moyens :** revues de projet, tests
  → test = **chercher des erreurs** dans un programme
  → exécution sur un sous-ensemble fini de données

- **4 types de tests :**
  → **test unitaire** : composants isolés
  → **test d'intégration** : composants assemblés
  → **test système** : système installé sur site
  → **test d'acceptation** : testé par l'utilisateur

Validation

- **But :** vérifier par rapport aux utilisateurs

- **Moyen :** revues de projet

Maintenance

But :

→ **corriger** des défauts
→ **améliorer** certaines caractéristiques
→ **suivre les évolutions** (besoin, environnement)

Catégorie :

→ **maintenance corrective d'où les changements corrigent des bogues dans le code**
→ **maintenance adaptative d'où les changements permettent à un système de s'exécuter dans une nouvelle infrastructure technique.**
→ **maintenance perfective qui permet les autres améliorations prévues à améliorer le système**

→ **maintenance préventive apporte des changements pour faciliter la future maintenance et l'évolution du système**

Remarque :

→ peut remettre en cause les fonctions du système
→ peut impliquer un re-développement *(re-ingeneering)*

Maquettage / Prototypage

- **But :**

  → **préciser les besoins** et les souhaits des utilisateurs

  → **développer** rapidement une **ébauche** du système

- **2 types de maquettes :**

  → **maquette exploratoire** : spécification

  → **maquette expérimentale** : conception

Répartition des activités

Actuellement, dans un **projet logiciel** bien conduit :

40 %	**Analyse, Spécification, Conception**
20 %	**Programmation**
40 %	**Intégration, Vérification, Validation**

Résumé

- analyse des besoins
- spécification
- *(maquettage)*
- conception
  - → architecturale
  - → détaillée

- programmation

- config. et intégration
- vérif. et validation

- maintenance

descriptions
de plus en plus
précises

=

raffinements

↓

Exécutable + Doc.

Introduction

- Modèle de développement ?

→ **enchaînements** et **interactions** entre les activités

- But pour le projet : ne pas s'apercevoir des problèmes qu'à la fin

→ **contrôler l'avancement** des activités en cours

→ **vérifier** / **valider** les résultats **intermédiaires**

- Objectif général : obtenir des processus de développement

→ **rationnels**

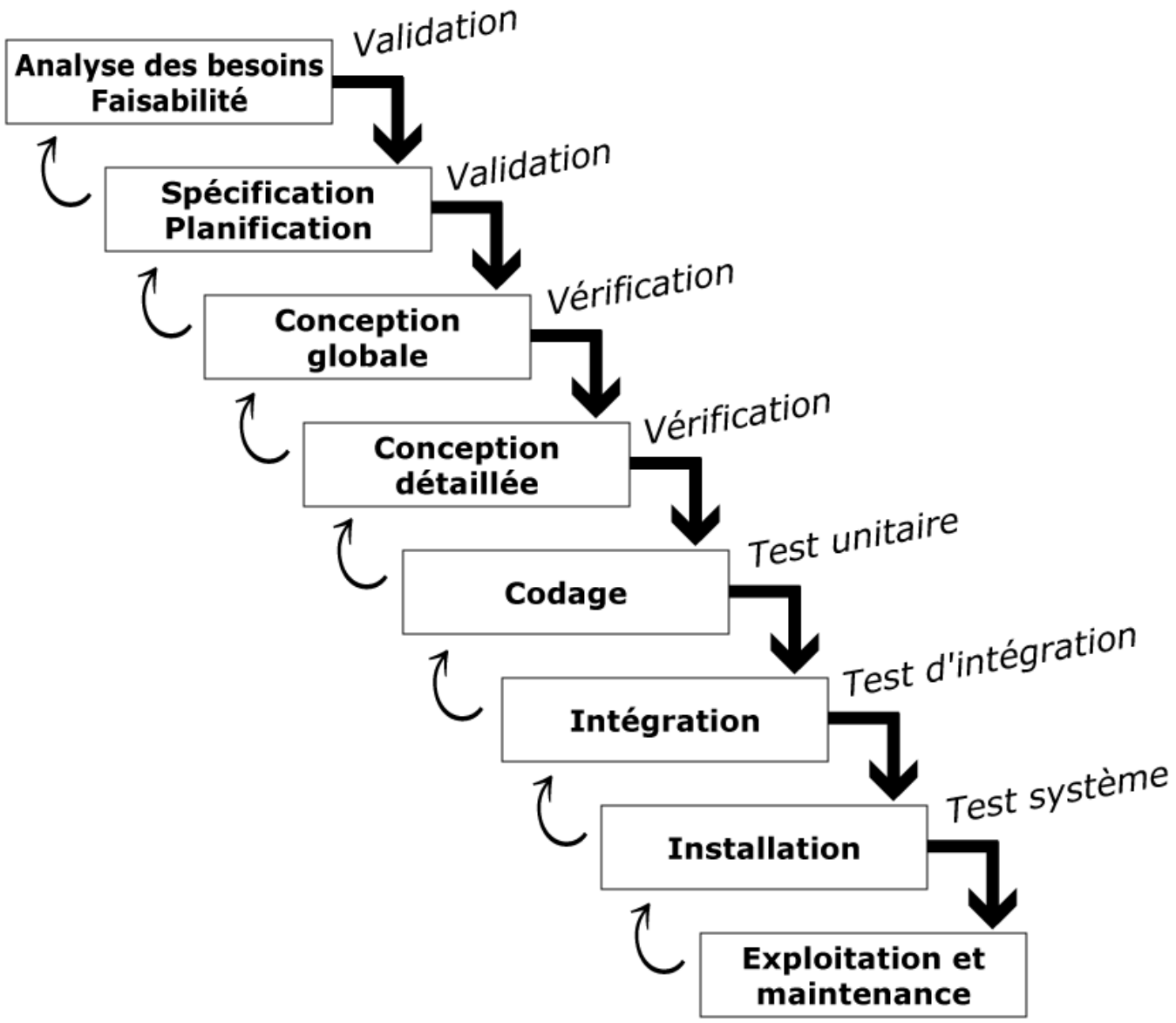
→ **contrôlables**

→ **reproductibles**

Modèles de développement logiciel

- modèle en **cascade** (fin des années 1960)
- modèle en **V** (années 1980)
- modèle en **spirale** (Boehm, 1988)

Modèle en cascade



Modèle en cascade

- **principe** : le développement se divise en étapes

  → une étape se **termine** à une certaine **date**

  → des **docs** ou **prog** sont produits à la fin de chaque étape

  → les résultats d'étapes sont **soumis à revue**

  → on passe à l'étape suivante **si** l'examen est **satisfaisant**

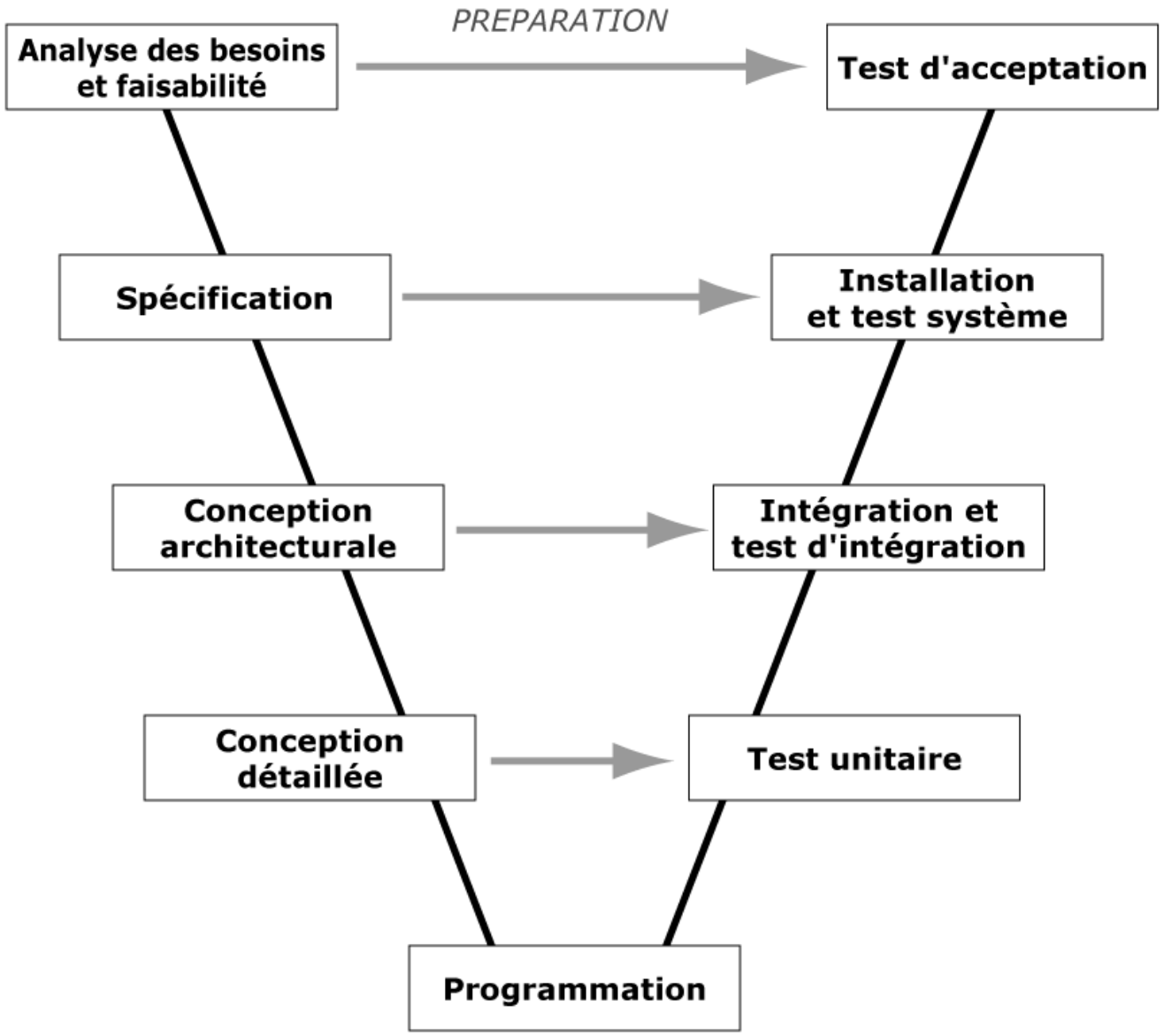
  → une étape ne remet en cause que la **précédente**

- **commentaire** :

  → modèle séduisant car **simple**

  → **moyennement réaliste** (trop séquentiel)

Modèle en V


PREPARATION
Analyse des besoins et faisabilité
Test d'acceptation
Spécification
Installation et test système
Conception architecturale
Intégration et test d'intégration
Conception détaillée
Test unitaire
Programmation

Modèle en V

- **principe** : les premières étapes préparent les dernières

- **interprétation** : 2 sortes de dépendances entre étapes

→ en V, enchaînement **séquentiel** (modèle en cascade)

→ de gauche à droite, les **résultats** des étapes de départ sont **utilisés par les étapes d'arrivée**

- **commentaire** :

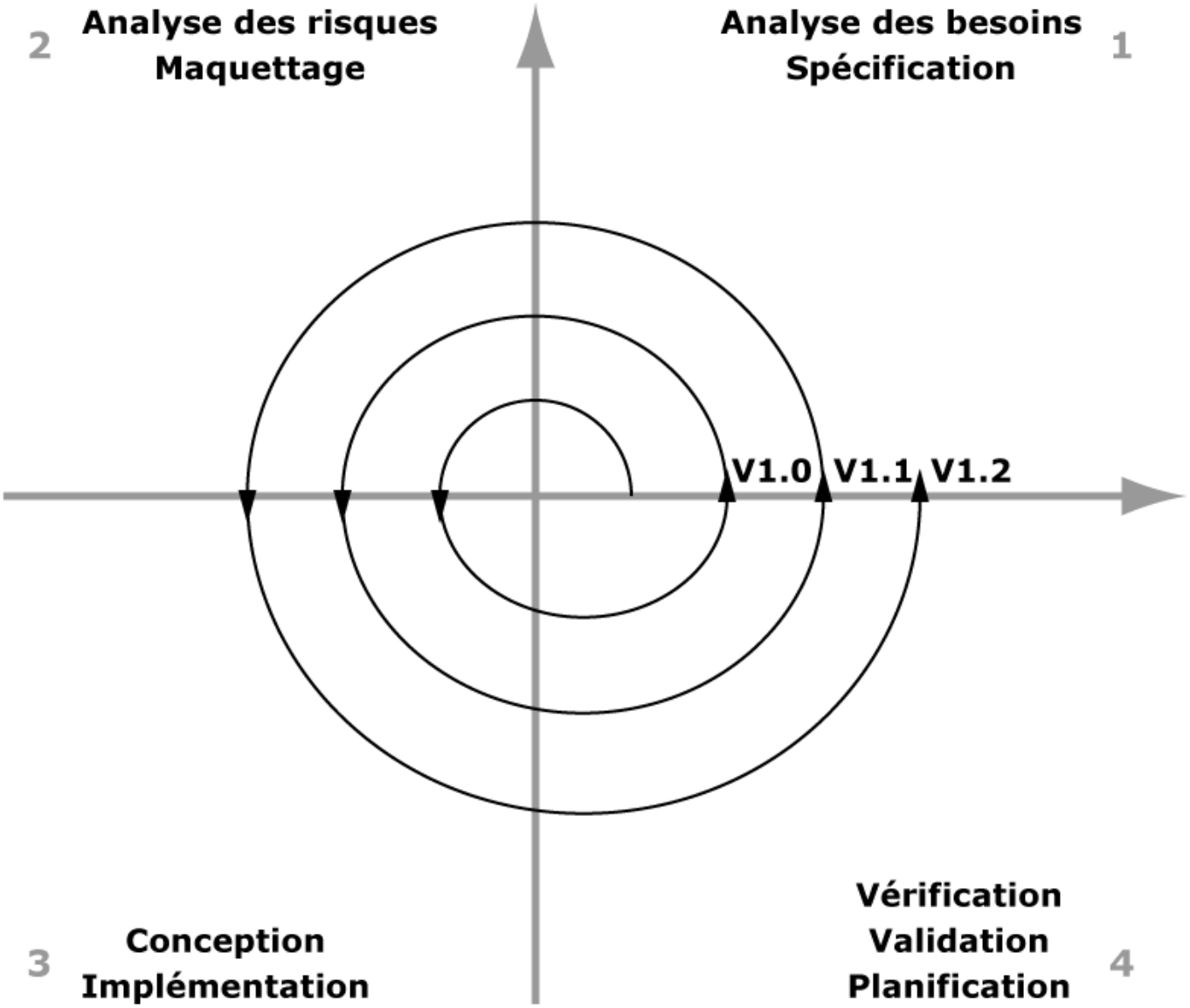
→ avec la **décomposition** est écrite la **recomposition**

→ vérification **objective** des spécifications

→ modèle plus **élaboré** et **réaliste**

→ **éprouvé** pour de grands projets, **le plus utilisé**

Modèle en spirale



Modèle en spirale

- **principe** : développement itératif (prototypes)
- **interprétation**: chaque mini-cycle se déroule en 4 phases

1. Analyse des **besoins**, **Spécification**
2. Analyse des **risques**, **Alternatives**, **Maquettage**
3. **Conception** et **Implémentation** de la solution retenue
4. Vérification, Validation, **Planification** du cycle suivant

- **commentaire** :

→ **nouveau** : analyse de risques, maquettes, prototypage
→ modèle **complet**, **complexe** et **général**
→ **effort important** de mise en œuvre
→ utilisé pour **projets innovants** ou **à risques**

Résumé

- **modèles** : enchaînements et interactions entre étapes
- **passage d'une étape à la suivante** :

  → **documents**, **tests**
  → **vérifiés** et **validés**

- **3 modèles** : cascade, V, spirale (**séquentiels** et **itératif**)
- **cascade** : **simple** mais **pas très réaliste**
- **spirale** : nouvelles notions, très **complet** mais **lourd**
- **V** : **assez réaliste**, le plus **éprouvé** et **utilisé**

Maturité du processus de développement

- **Notion définie par le SEI** *(Software Engineering Institute)*

- ***Capacity Maturity Model (CMM)***

- **5 niveaux de maturité :**

  → **Initial**

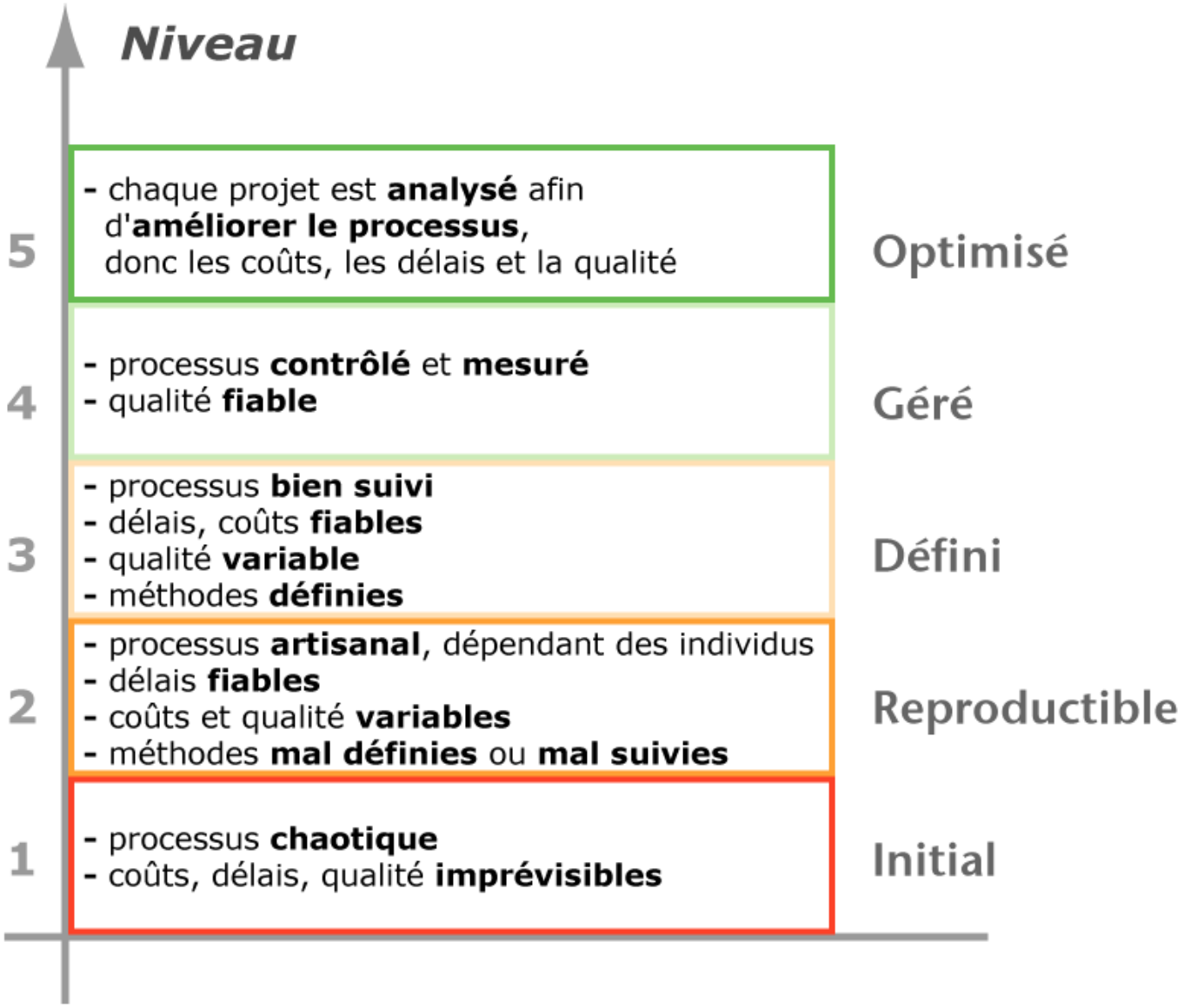
  → **Reproductible**

  → **Défini**

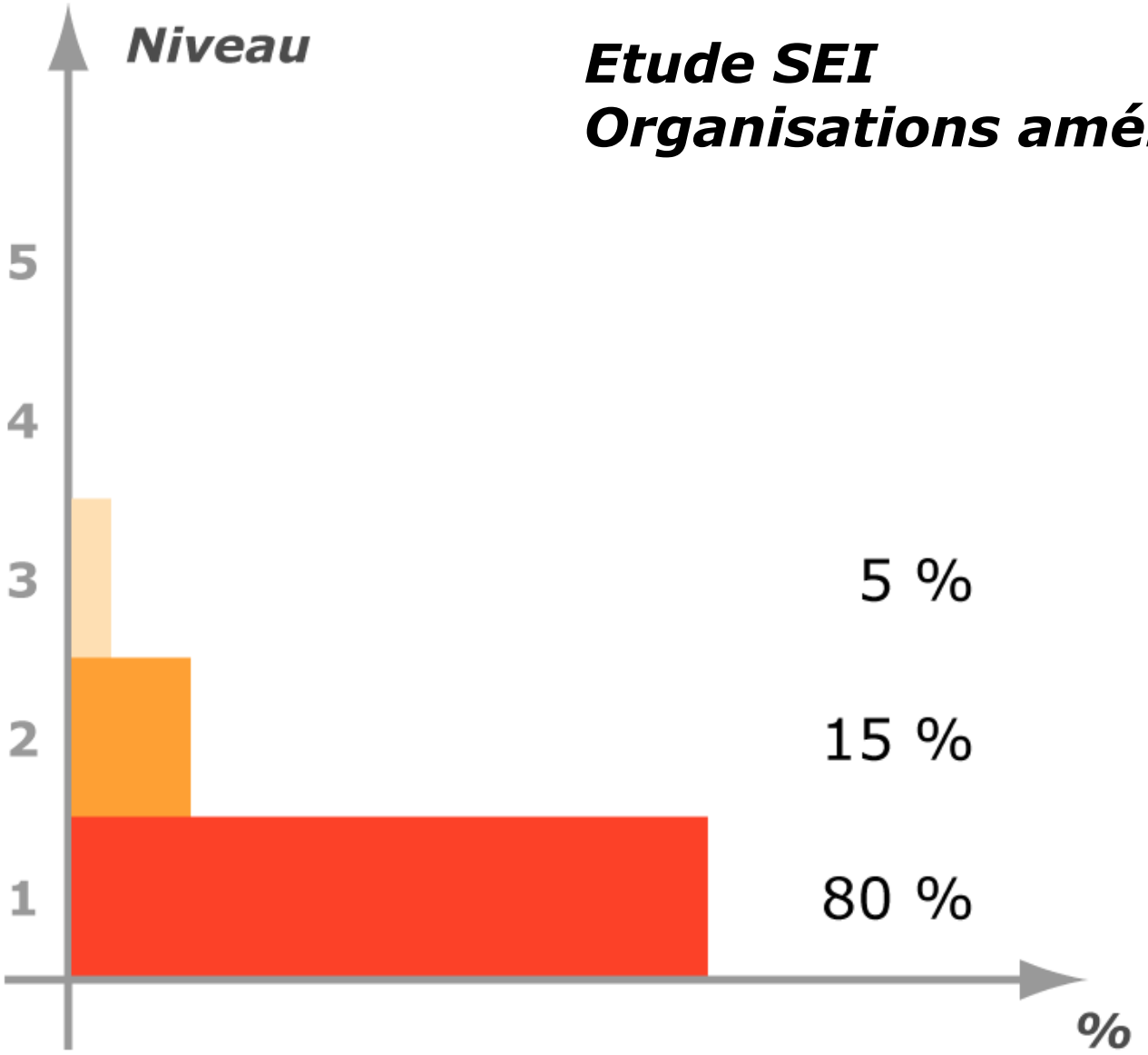
  → **Géré**

  → **Optimisé**

Niveaux de maturité

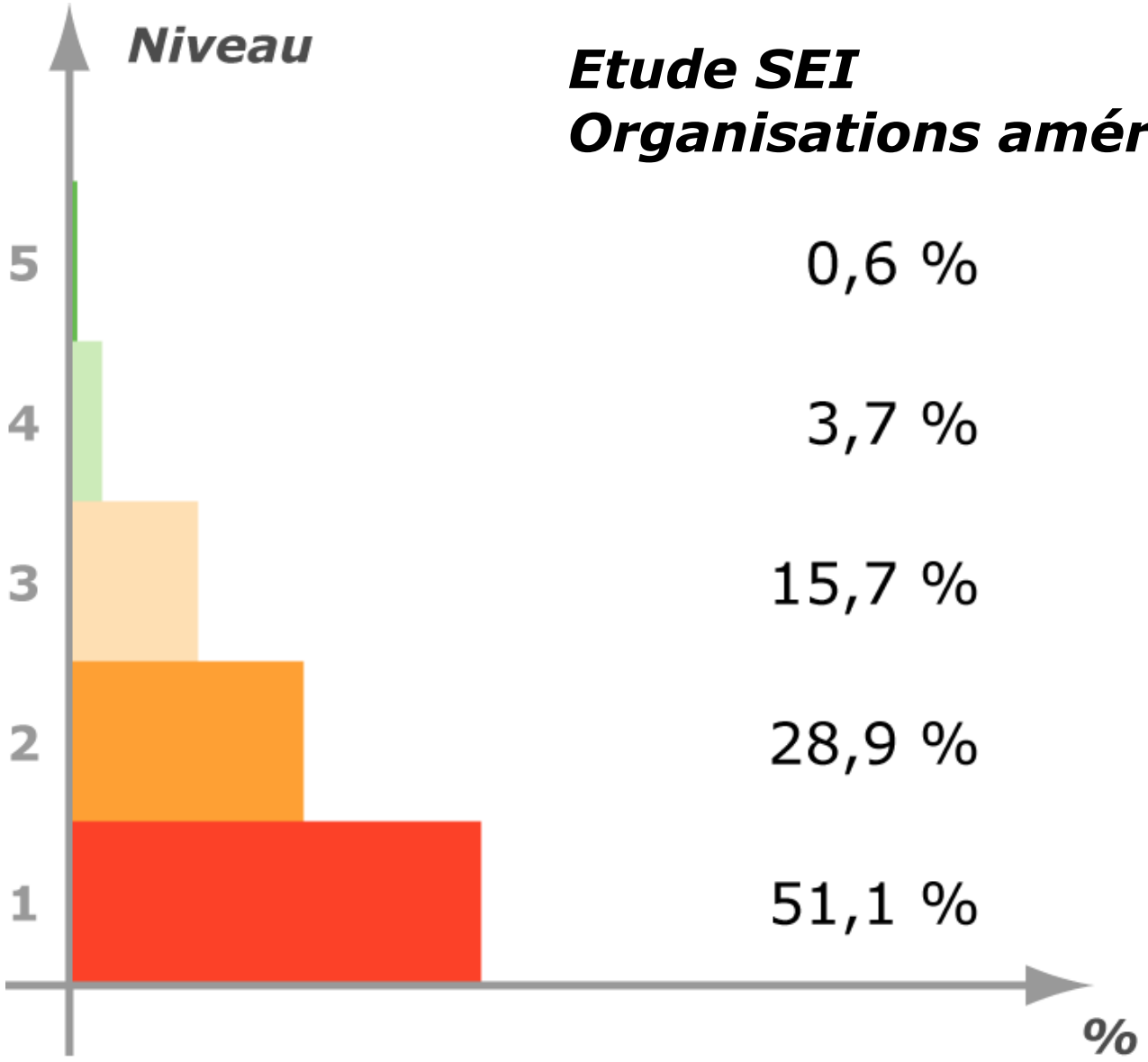


Etude américaine (1991)



Etude SEI
Organisations américaines

Etude américaine (1999)



Principes du Génie Logiciel

Vers une assurance qualité …

Différentes perceptions de la qualité …

Facteurs de qualité - I

Qualité **externe** :

- **complétude fonctionnelle**
  - → réalise toutes les tâches attendues
- **ergonomie** / **convivialité**
  - → facile d'utilisation
  - → apprentissage aisé
- **fiabilité** / **robustesse**
  - → tâches effectuées sans problème
  - → fonctionne même dans des cas atypiques

Facteurs de qualité - 2

Qualité **interne** :

- **adaptabilité**
  → facile à modifier, à faire évoluer
- **réutilisabilité**
  → des parties peuvent être réutilisées facilement
- **traçabilité** / **compréhension**
  → le fonctionnement du logiciel est facile à comprendre
- **efficacité** / **performance**
  → bonne utilisation des ressources (mémoire, cpu, …)
- **portabilité**
  → capacité à marcher dans un autre contexte ou env.

Comment agir sur la qualité logicielle ?

La **qualité** est atteinte ou **améliorée** en appliquant certains **principes** :

- rigueur et formalisme
- séparation des préoccupations
- modularité
- généralité / abstraction
- incrémentalité
- anticipation des changements

Rigueur et formalisme

- **rigueur** = précision, exactitude *(confiance en la fiabilité)*
- **formalisme** = le plus haut degré de rigueur *(mathématiques)*
  - nécessaire pour les parties critiques (haut risque)
  - peut être utilisé dans chaque phase

→ spécification formelle
→ vérification formelle (preuve)
→ analyse de complexité d'algorithmes
→ …

Séparation des préoccupations - I

- **principe :** traiter séparément les ≠ aspects d'un problème → **diviser pour régner**

- **résultat :** réduit la quantité de complexité à contrôler

Séparation des préoccupations - 2

- **différentes sortes de séparations :**
  - **séparation de domaine**

    → domaine de *problème* : *quoi* résoudre ?
    → domaine de *solution* : *comment* résoudre ?

  - **séparation de temps** : phases du cycle de vie
  - **séparation de qualité**

    → maquettes, prototypes
    → conception globale, détaillée

  - **vues séparées sur le logiciel** : modélisation en UML

    → cas d'utilisation, structure statique
    → comportement dynamique, architecture

  - **séparation de responsabilités** : org. en équipes projet

Modularité

- **principe :** séparer le système en composants logiques

  → **modules**

- **avantages :**

- **réduction de complexité**

- **les modules peuvent être**

  → conçus et construits séparément
  → réutilisés

- système modifié en changeant un nombre **limité** de modules

Généralité / Abstraction

- **principe :**

  → généraliser un problème particulier
  → le rendre réutilisable dans d'autres contextes

- **exemple :**

- **logiciel générique** vs **logiciel sur mesure**

- **design** : des solutions généralisées pour des problèmes typiques de conception

Incrémentalité

- **principe :**
  - → développer le logiciel en une série d'incréments
  - → se rapprocher de la solution par raffinements successifs

- **exemple :**
  - **phase de conception**
  - **cycle de vie en spirale**

Anticipation des changements

- le logiciel **évolue constamment** pour différentes

raisons :

- **réparation des erreurs détectées**
- **adaptation à de nouveaux environnements**
- **traitement de nouvelles exigences**
- **changements dans les exigences**
- **changement des formats de données**
- **changement d'exigences non-fonctionnelles**

- avant de développer, **poser les**

questions :

→ quels changements, où ?
→ comment les rendre plus faciles à appliquer ?

Résumé

- la qualité du logiciel est **fondamentale**
- elle est aperçue différemment selon les **points de vue** :
  - **qualité externe** : client, utilisateurs
  - **qualité interne** : développeurs, gestionnaires
- pour l'atteindre, on adopte des **principes**
  - participation des **activités** et **modèles de développement**
  - l'utilisation de l'**approche OO** participe aussi beaucoup à remplir ces objectifs

Résumé général

- **logiciel** ≠ programme
- **problèmes** : → **pas fiable**
  → **dépassements** (délais, coûts)
  → **non conforme** au cahier des charges
- **génie logiciel** : → démarche **ingénierique**
  → **méthodes**, **principes**, **outils**
- **méthodes** : → **processus** de développement
  → **activités** et **modèles** (cascade, V, spirale)
- **principes** : pour atteindre des objectifs de **qualité**
- **outils** : Ateliers de Génie Logiciel (AGL)

Conclusion

- situation en progrès : → logiciel **plus fiable**
  → plus facilement **modifiable**
  → **satisfait** mieux les utilisateurs
- en contrepartie, les problèmes sont plus **critiques**,
  → centr. téléph., centrales nucléaires
  → avions, spatial, ferroviaire
  → banque, bourse, …
- plus **complexes**,
  → de plus en plus de **fonctionnalités**
  → **systèmes distribués**
  → **mutations** technologiques **rapides**
- et les **exigences** plus fortes (fiabilité, permanence du service)
- la maîtrise du logiciel reste un **défi scientifique** et technologique **majeur**