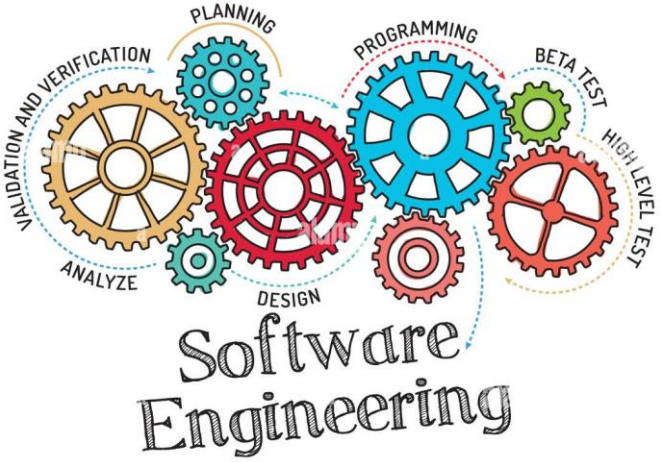
Université Sultan Moulay Slimane
Ecole Supérieure de Technologie Béni Mellal
Département ITG
DUT- IDIA 1 - S2
M122 - Ingénierie Logicielle avec Java et UML
Année universitaire: 2024-2025

Support de cours

Ingénierie logicielle avec Java et UML

Partie II: Génie Logiciel & Modélisation UML



Chapitre I: Introduction au Génie Logiciel

Pr. Siham BAKKOURI
Email: s.bakkouri@usms.ma

Plan

- **INTRODUCTION**
- **ETAT DE CONNAISSANCE DE GL**
- **PRINCIPES DU GÉNIE LOGICIEL**
  - **HISTORIQUE ET ÉVOLUTION DU GÉNIE LOGICIEL**
  - **DÉFINITION DU GÉNIE LOGICIEL**
- **DÉMARCHE DE DÉVELOPPEMENT ET CYCLE DE VIE D'UN LOGICIEL**
  - **DÉMARCHE DE DÉVELOPPEMENT LOGICIEL**
  - **MODÈLES DE CYCLE DE VIE TRADITIONNELS ET ITÉRATIF**
  - **LES MÉTHODES AGILE DE DÉVELOPPEMENT**
- **QUALITÉ DU LOGICIEL**

Introduction

→ **Un système d'information** : est l'ensemble des éléments participants à la gestion, au stockage, au traitement, au transport et à la diffusion de l'information au sein de l'organisation

→ **Un système informatique :** est l'ensemble des équipements destinés au traitement automatique de l'information permettant d'acquérir, de stocker, de traiter et de communiquer des données.

- **Systèmes informatiques**
  - **80**% de logiciel
  - **20**% de matériel

Depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement

- Le matériel est relativement fiable
- Le marché est standardisé

Les problèmes liés à l'informatique sont essentiellement des problèmes de Logiciel

Introduction

Un **logiciel** est un produit qui couvre un ensemble organisé de :

- Programmes
- Procédés
- Documentation
- Services

Logiciel = programme + utilisation

Caractéristiques du logiciel

- Un objet immatériel
- Le logiciel est facile à reproduire
- Ses fonctionnalités sont difficiles à figer au départ et souvent remises en cause
- La plupart des logiciels sont personnalisés

Introduction

→ Domaines d'application du logiciel

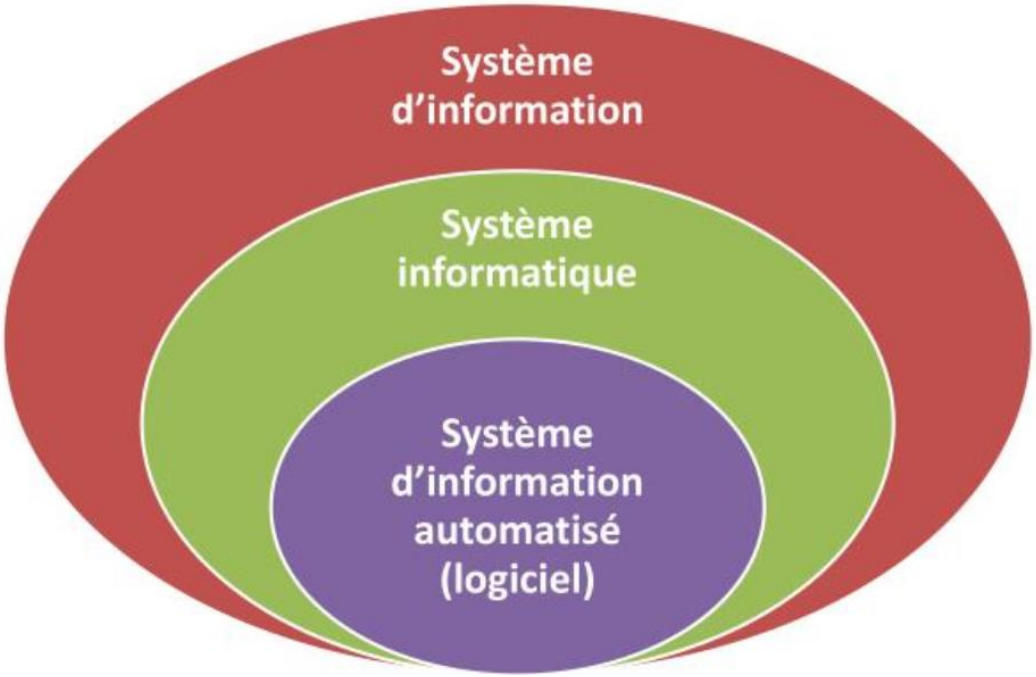
- **Système**
  - Compilateurs (A-0 System)
  - Editeurs
  - Gestion de fichiers
- **Temps réels (Real-time)**
  - Contrôle de machine
- **Affaires (Data processing)**
  - SGBD (Oracle)
  - ERP (SAP)
- **Embarqué (Embedded)**
  - Programme FPGA
  - Auto-contrôle
- **Scientifique**
  - Simulation
  - Conception assisté par ordinateur
  - Calcul numérique intensif
- **Bureautique**
- **Intelligence artificielle**
  - Système expert

Introduction

→ Un **logiciel** est un système d'information automatisé

→ Un **système d'information automatisé** est l'ensemble des moyens et des méthodes qui se rapporte au traitement automatisé des données.

- Il constitue la partie logicielle du système informatique

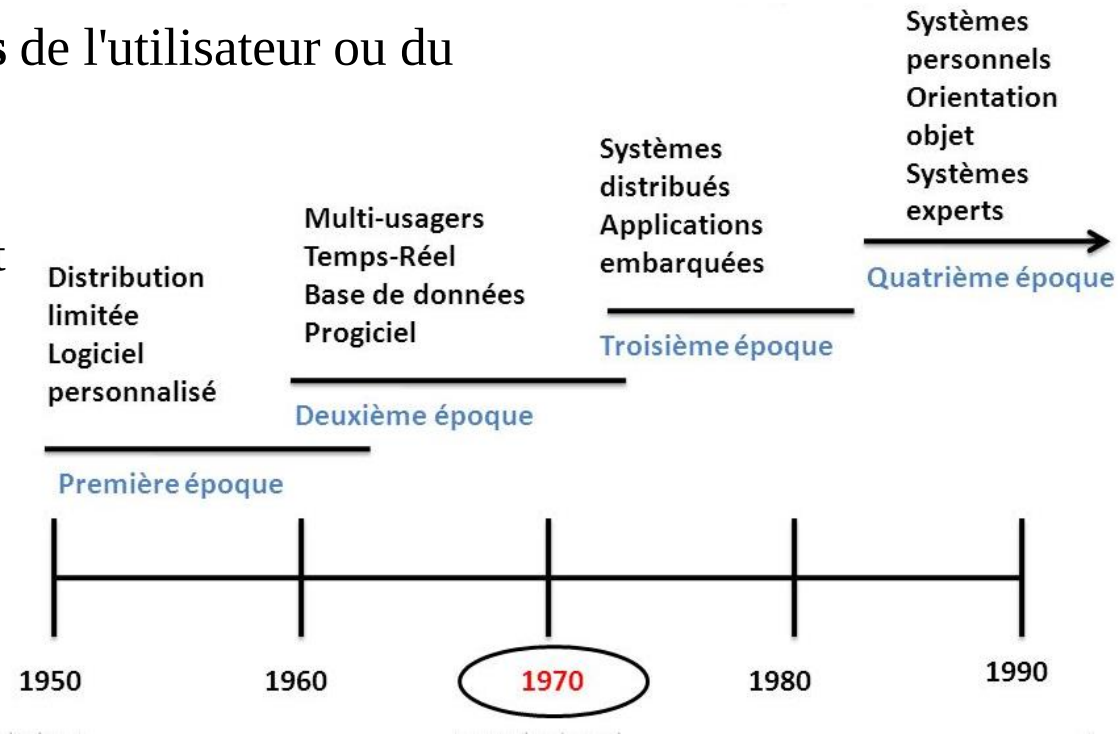


État des connaissances en GL

→ La crise du logiciel

→ Constat du développement logiciel fin années 60 :

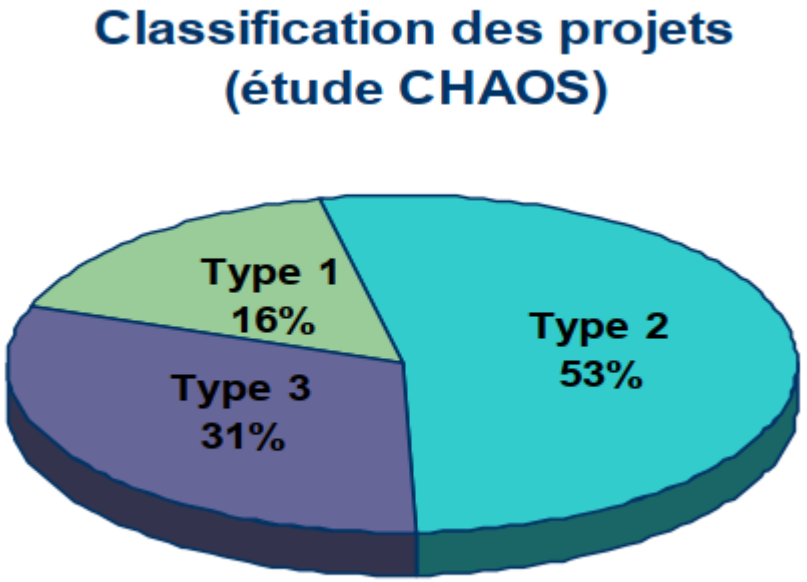
- Délais de livraison **non respectés**
- Budgets **non respectés**
- **Ne répond pas aux besoins** de l'utilisateur ou du client
- **Difficile** à utiliser, maintenir, et faire évoluer



État des connaissances en GL

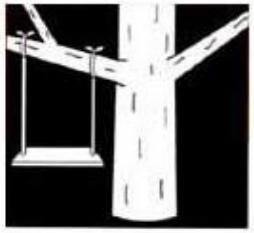
→ La crise du logiciel

- **Type 1**: Projets réussis à temps dans le budget prévu et réalisant les fonctionnalités demandées
- **Type 2**: Projets terminés mais hors budget, hors planning et n'offrant pas toutes les fonctionnalités requises
- **Type 3**: Projets abandonnés en cours de réalisation

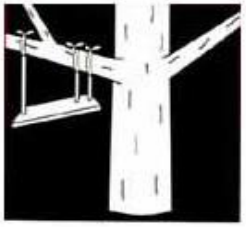


État des connaissances en GL

→ La crise du logiciel

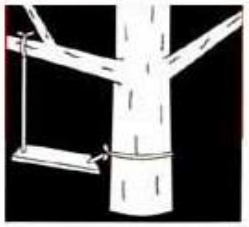


Ce que voulait
le client...

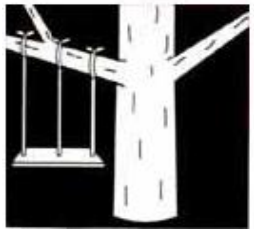


Ce qu'a compris
le chef de projet...

Ce qu'a spécifié
le client...



Ce qu'a compris
le concepteur...



Ce qu'a promis
l'ingénieur commercial...

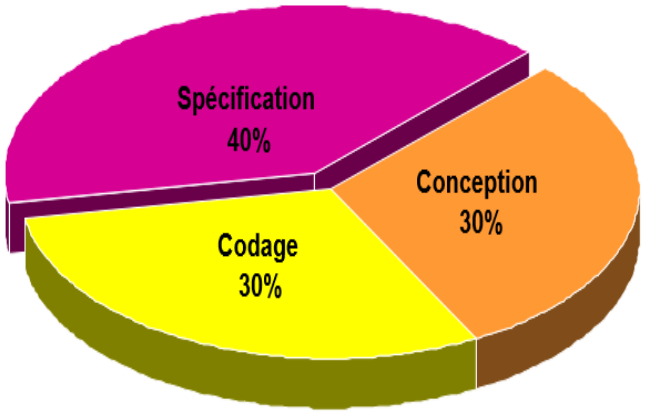
Ce qui a été livré :
Version 1.0

Ce qui fonctionne
actuellement :
Version 1.2.5 + patch

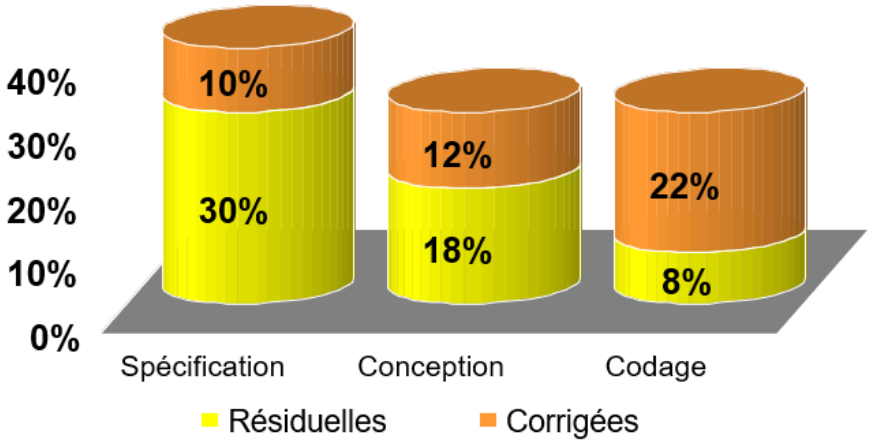
État des connaissances en GL

→ La crise du logiciel en chiffre

→ Répartition des erreurs



→ Ténacité des erreurs



État des connaissances en GL

Exemples d'échec de logiciel

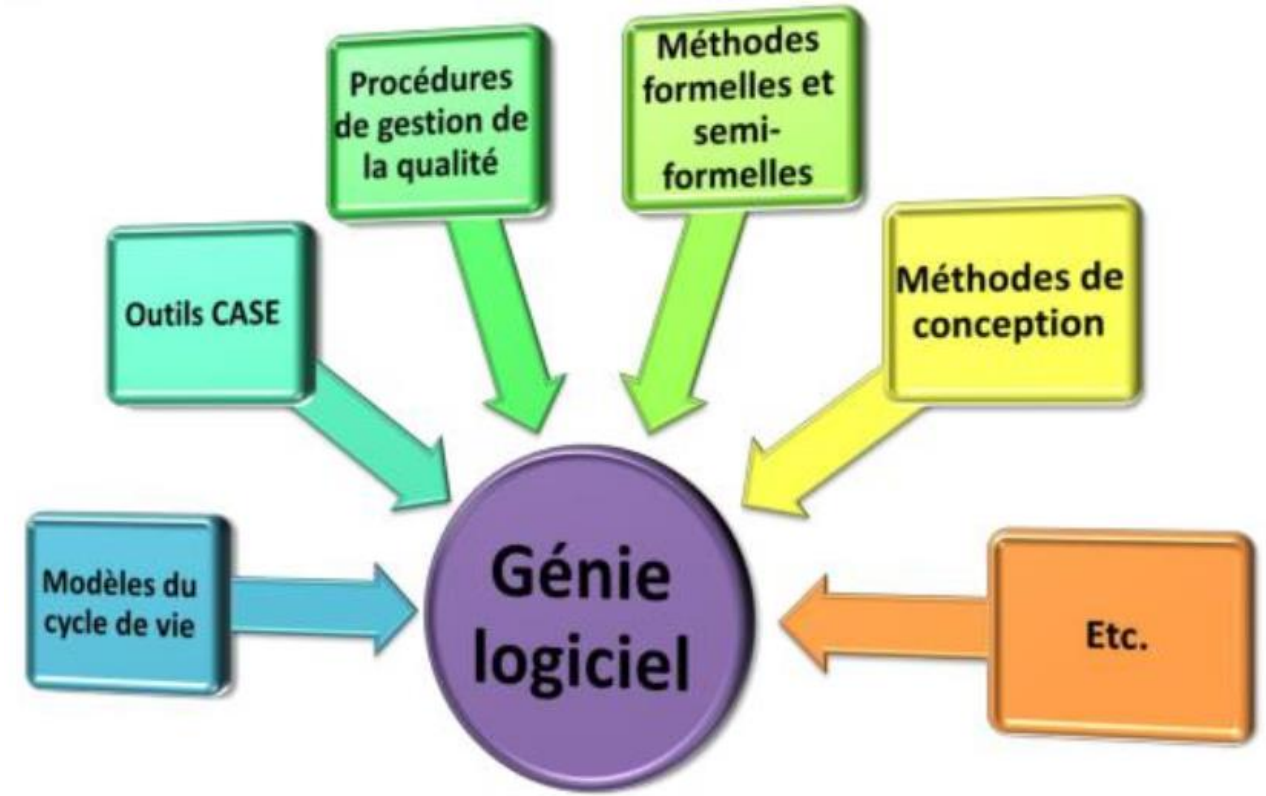
- Perte de la première sonde Mariner vers Venus suite à une erreur de programmation dans un programme Fortran
- Abandon du projet d'informatisation de la bourse londonienne après 4 ans de travail et 100 M£ de pertes
- Retard (2 ans) du premier vol de l'Eurofighter

❑ Les causes

- Difficulté de maîtrise des coûts (200 millions de dollars pour fabriquer OS-360)
- Difficulté de maîtrise des délais de réalisation (2 ans de retard pour les premiers compilateurs PL/1, Algol 68, ADA)

État des connaissances en GL

Solution: En octobre 1968 lors d'une conférence de l'OTAN à Garmisch-Partenkirchen en Allemagne.



PRINCIPES DU GÉNIE LOGICIEL

Principes du Génie Logiciel

→ Historique et l'évolution de Génie Logiciel

- Le terme **Génie Logiciel** est né entre le 7 et le 11 octobre 1968 à Garmish-Partenkirchen sous le nom de software engineering sous le parrainage de l'OTAN

- Défini par un groupe de scientifiques pour répondre à un problème bien défini s'énonçant en 2 constatations :

    - Le logiciel n'était pas fiable

    - Il était incroyablement difficile de réaliser dans des délais prévus des logiciels satisfaisant leur cahier des charges

- Appliquer les **méthodes classiques d'ingénierie** au domaine du **Logiciel**

Principes du Génie Logiciel

→ Définition du Génie Logiciel

- **Génie Logiciel** ou ingénierie logicielle est une discipline de l'informatique qui se consacre à l'étude, la création, la gestion et la maintenance de logiciels de manière systématique, contrôlée et efficace.
- Méthodologie de construction en équipe d'un logiciel complexe et à multiples versions.
- Il englobe l'ensemble des activités liées au développement de logiciels, depuis leur conception initiale jusqu'à leur mise en production et leur évolution ultérieure.

- **Programmation vs. Génie logiciel**
  - Programmation = Activité personnelle
  - Génie logiciel = Activité d'équipe structurée organisée autour d'un projet
- Suivant les projets, la partie programmation (***codage***) ne représentera qu'entre **10%** et **30%** du coût total.

DÉMARCHE DE DÉVELOPPEMENT ET CYCLE DE VIE D'UN LOGICIEL

Démarche de développement logiciel

→ Une démarche de développement repose sur :

- **Un formalisme** (exemple : orientation objet)
- **Une méthode** (Merise , SADT , FUSION , OMT , RUP / UML , …)
- **Un processus et un cycle de vie**

→ Notion de processus de développement :

Un processus de développement représente un ensemble d'étapes ou activités successives permettant la production d'un système logiciel dans les délais (et le budget) fixés par le client et répondant aux besoins des utilisateurs .

Démarche de développement logiciel

→ Les étapes d'un processus de développement

- **Spécification** : établissement du cahier des charges et des contraintes du système (capture des besoins).
- **Analyse** : Détermination des éléments constituant le système.
- **Conception** : Production d'un modèle du système final tel qu'il doit fonctionner.
- **Implémentation** : réalisation du système (codage des composants et assemblage).
- **Test :** Vérification de l'adéquation entre les fonctionnalités du système et la description des besoins .
- **Installation :** livraison du système au client et vérification de son fonctionnement.
- **Maintenance :** réparation des fautes dans le système au fur et à mesure de leur découverte.

Cycle de vie d'un logiciel

→ Définition

Le cycle de vie d'un logiciel (ou cycle de développement logiciel) est une séquence d'étapes et de phases par lesquelles un logiciel passe depuis sa conception initiale jusqu'à sa mise en service, sa maintenance, et finalement son retrait.

Modèle de cycle de vie :

- Un modèle de cycle de vie débute par une idée et se termine par un logiciel .
- Il permet de gérer l'exécution d'un processus de développement sous forme d'une seule itération ou par itérations successives dans le cadre d'un projet jusqu'à la satisfaction des besoins du client .
- Dans le cas de plusieurs itérations , chaque itération représente un processus ou mini-projet complet (**spécification** + **analyse** + **Conception** + **Implémentation** + **Test**) .

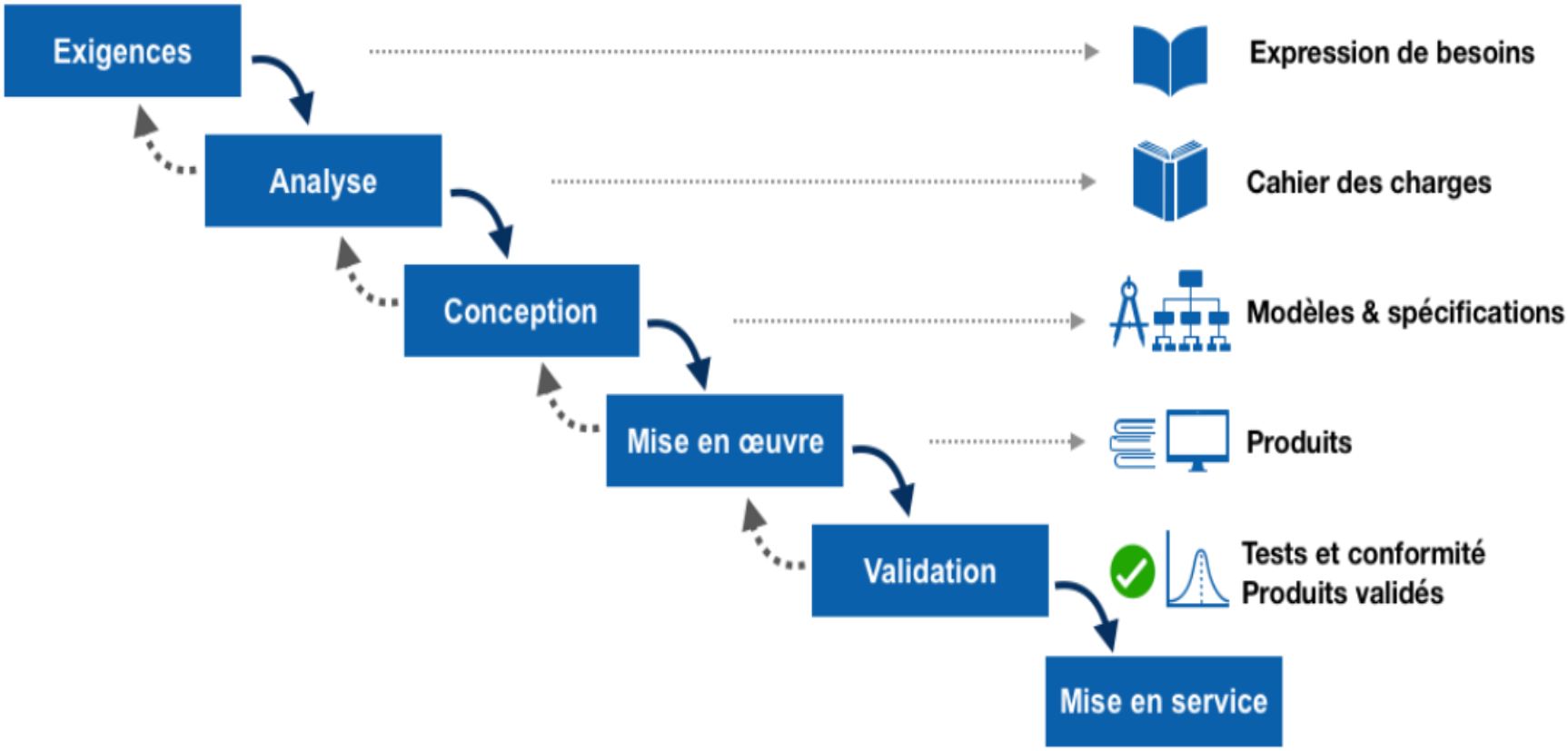
Modèles de cycle de vie traditionnels

1. Modèle en cascade (Waterfall)

- **Principe** : Chaque activité est relative à l'entièreté du cahier des charges (cycle de vie à une seule itération)
- **Inconvénients**:
  - Modèle à structure linéaire bien adapté à certaines disciplines (exemple : génie civil) mais pas au génie logiciel .
  - Modèle trop rigide: une activité ne peut commencer que si l'activité précédente est complètement terminée
  - Pas de possibilité de travail en parallèle et validation finale tardive

Modèles de cycle de vie traditionnels

1. Modèle en en cascade (Waterfall)

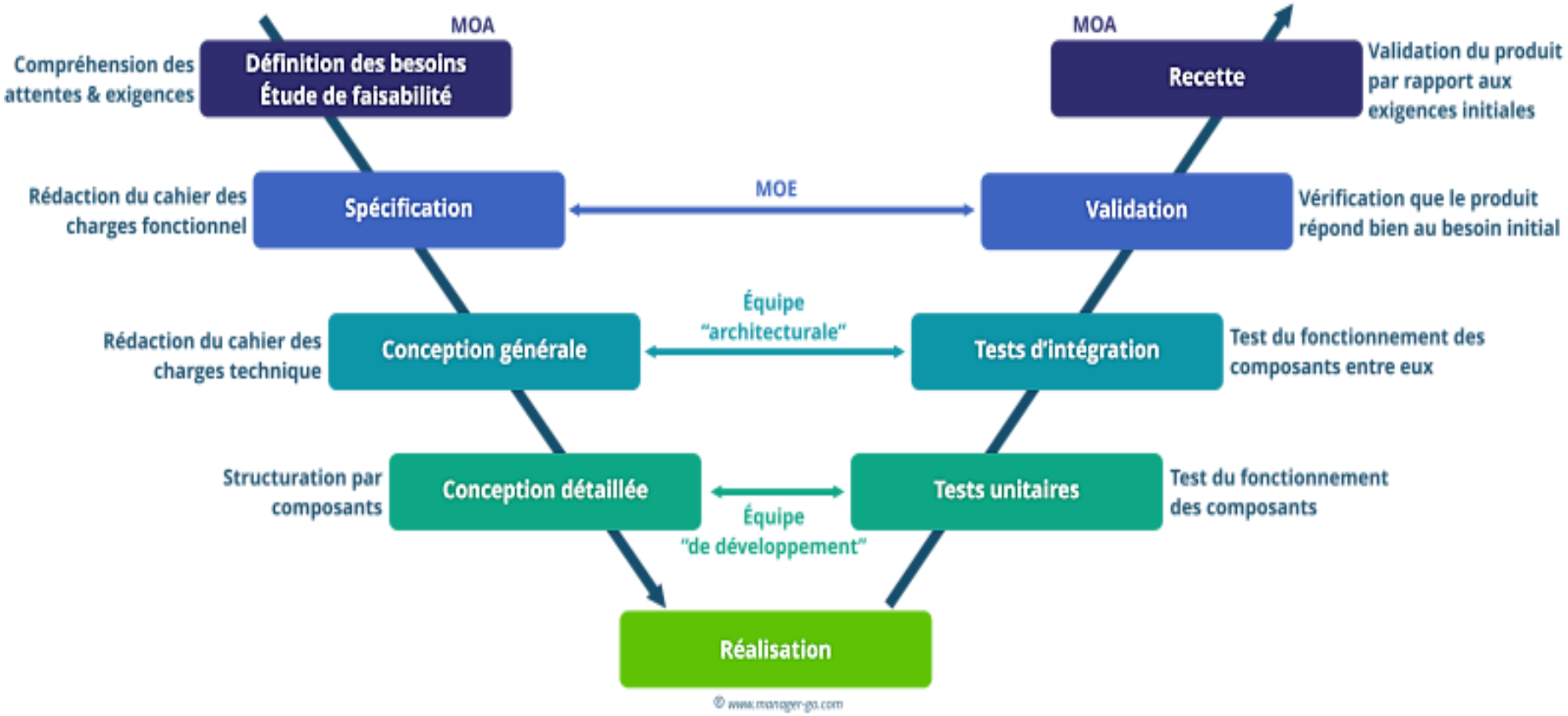


2. Modèle en V

- **Principe** : Similaire au modèle en chute d'eau mais avec une validation rétrograde et plus longue.
- **Inconvénients :**
  - Modèle à structure linéaire et validation tardive comme le modèle en cascade (waterfall)
  - Mise en œuvre et installation tardives entraînant des erreurs très coûteuses .

Modèles de cycle de vie traditionnels

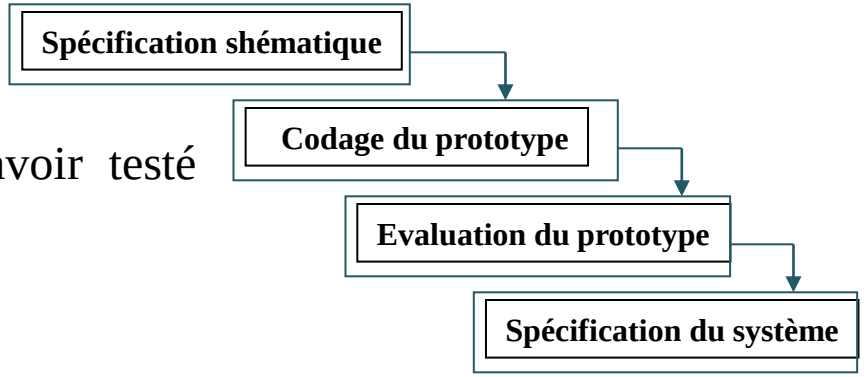
2. Modèle en V



→ Les principaux modèles de cycle de vie

3. Modèle par prototypage jetable

- **Principe** : On effectue une ou plusieurs itérations (prototypes) sous forme de processus distincts uniquement pour aboutir à une spécification exacte du système (concertation avec le client)



- **Avantages :**
  - On arrive à satisfaire le client après avoir testé différents prototypes du système
- **Inconvénients :**
  - Approche très coûteuse en terme de durée de développement (elle nécessite des ressources humaines importantes)

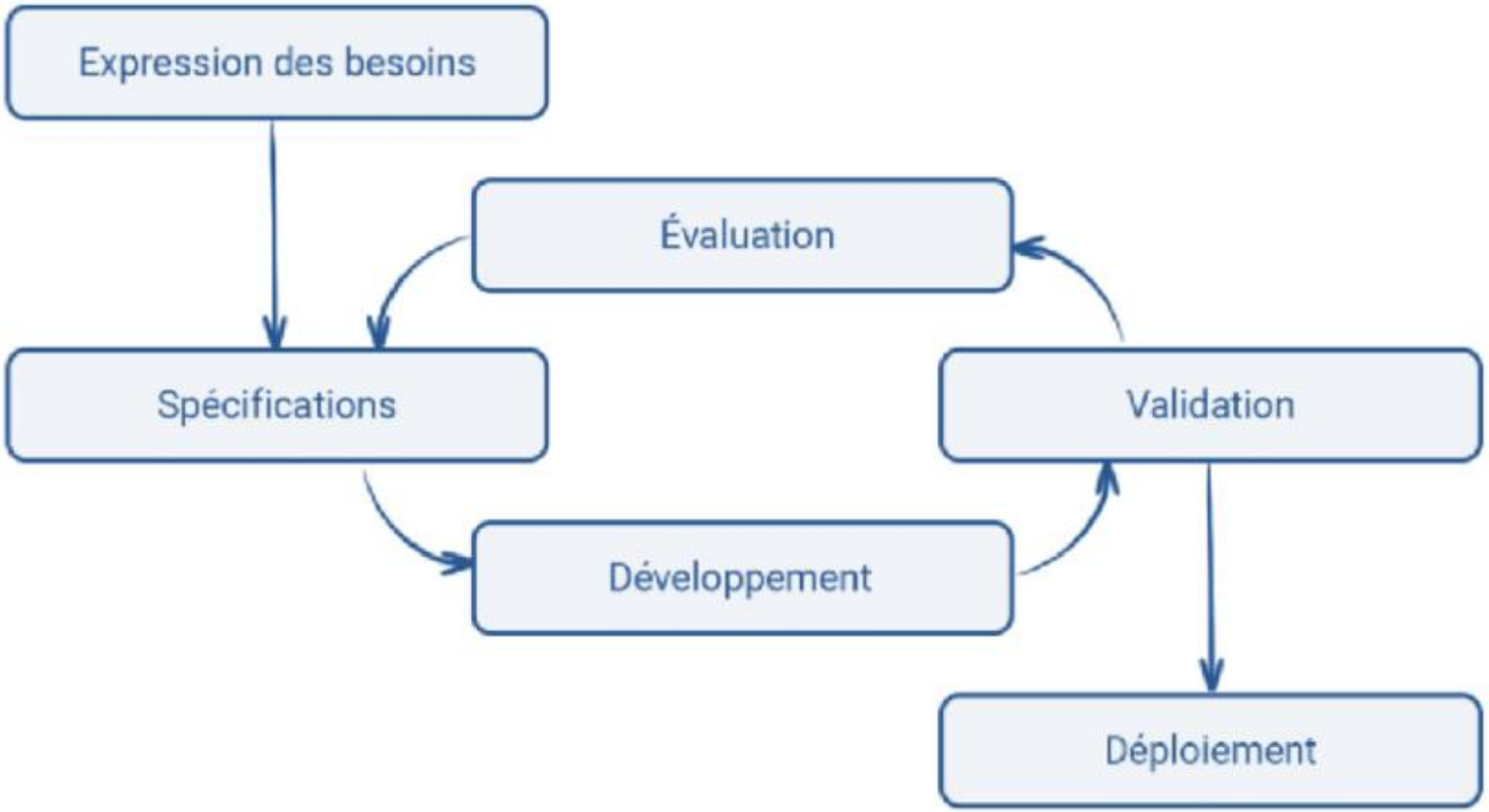
Modèles de cycle de vie itératif

4. Modèle incrémental ou itératif

- **Principe** :
  - Après l'analyse des besoins (spécification) , on effectue plusieurs cycles de développements successifs (itérations)
  - Chaque cycle conduit à une version utilisable du produit et traite un petit nombre de besoins
- **Avantages :**
  - A chaque itération , la complexité du système diminue.
  - Des implémentations de parties du système sont disponibles très rapidement . On peut disposer alors d'informations utiles sur le comportement du système (early feedback) et son adéquation aux besoins .
  - Ce modèle permet le travail en parallèle des équipes

Modèles de cycle de vie itératif

4. Modèle incrémental ou itératif



5. Modèle en spirale

- **Principe :**

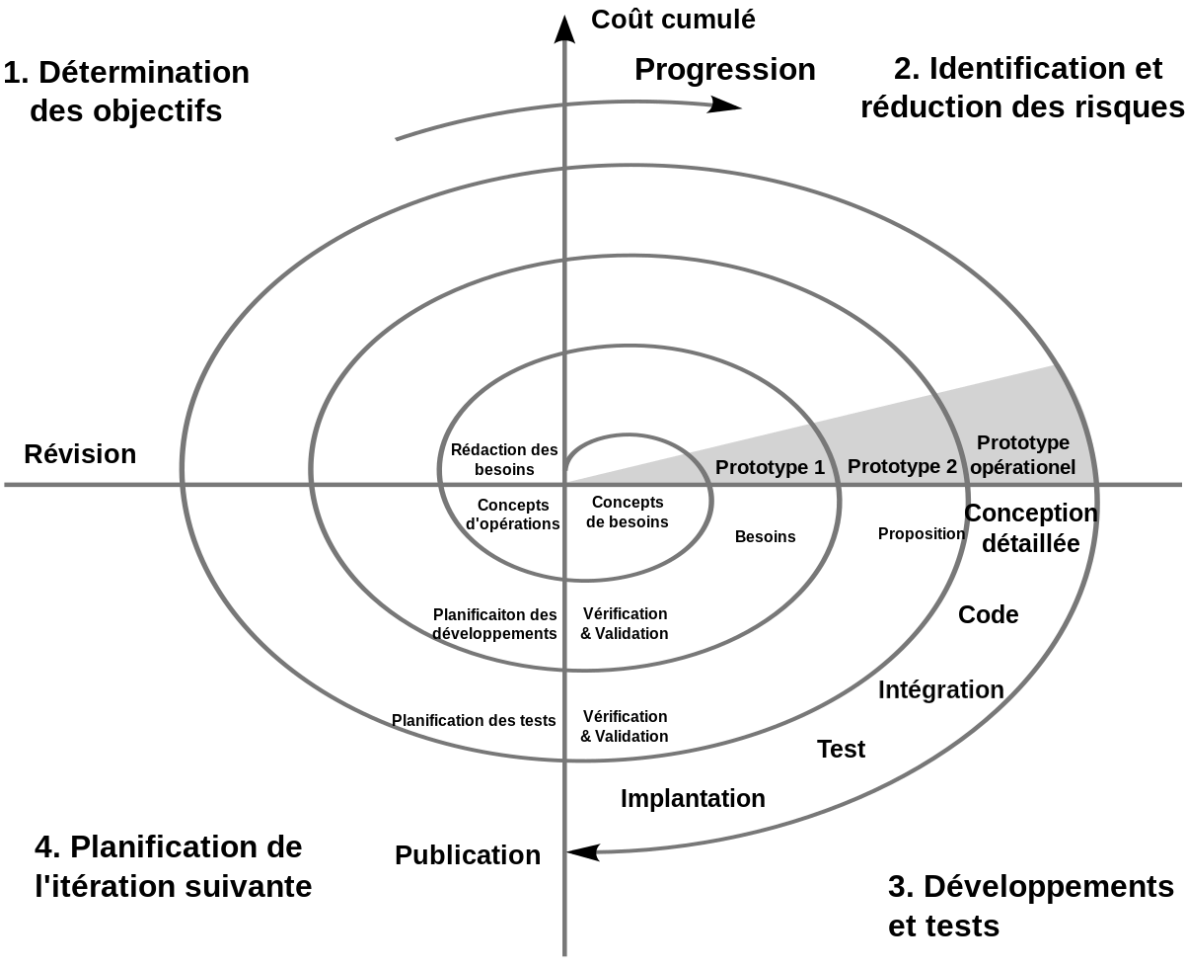
- On effectue plusieurs cycles de développement de manière itérative incluant chacun une spécification (analyse ou réanalyse des besoins)
- Chaque cycle conduit à une version utilisable du produit et peut faire appel à un modèle de développement différent : modèle en cascade, modèle de prototypage jetable , etc…

- **Avantages :**

- Ce modèle est plus flexible que le modèle incrémental avec la possibilité de changer de stratégie à chaque itération (sous-modèle)
- La spécification est constamment remise en cause (réexaminée)
- Ce modèle permet le travail en parallèle des équipes comme dans le cas du modèle incrémental .

Modèles de cycle de vie itératif

5. Modèle en spirale



L'approche agile de développement

Méthodologie de gestion de projet centrée sur l'adaptation rapide au changement, la collaboration et la livraison continue de valeur.

❑ Principes clés (Manifeste Agile)

- Individus et interactions > Processus et outils
- Logiciel fonctionnel > Documentation exhaustive
- Collaboration client > Négociation contractuelle
- Réponse au changement > Suivi d'un plan

❑ Méthodes Agiles

- **Scrum :** rôles, sprints, backlog, mêlées quotidiennes
- **XP (Extreme Programming) :** intégration continue, développement piloté par les tests
- **Kanban :** visualisation du flux, amélioration continue

➔ Il convient également de noter l'existence d'autres méthodes agiles, telles que :

- La méthode **RAD** (Rapide Application Development)
- **IAD** (Iterative Application Development)
- Extreme programming (**XP**)
- Dynamic Systems Development Method (**DSDM**)
- Feature Driven Development (**FDD**)

L’approche agile de développement

❑ Les avantages

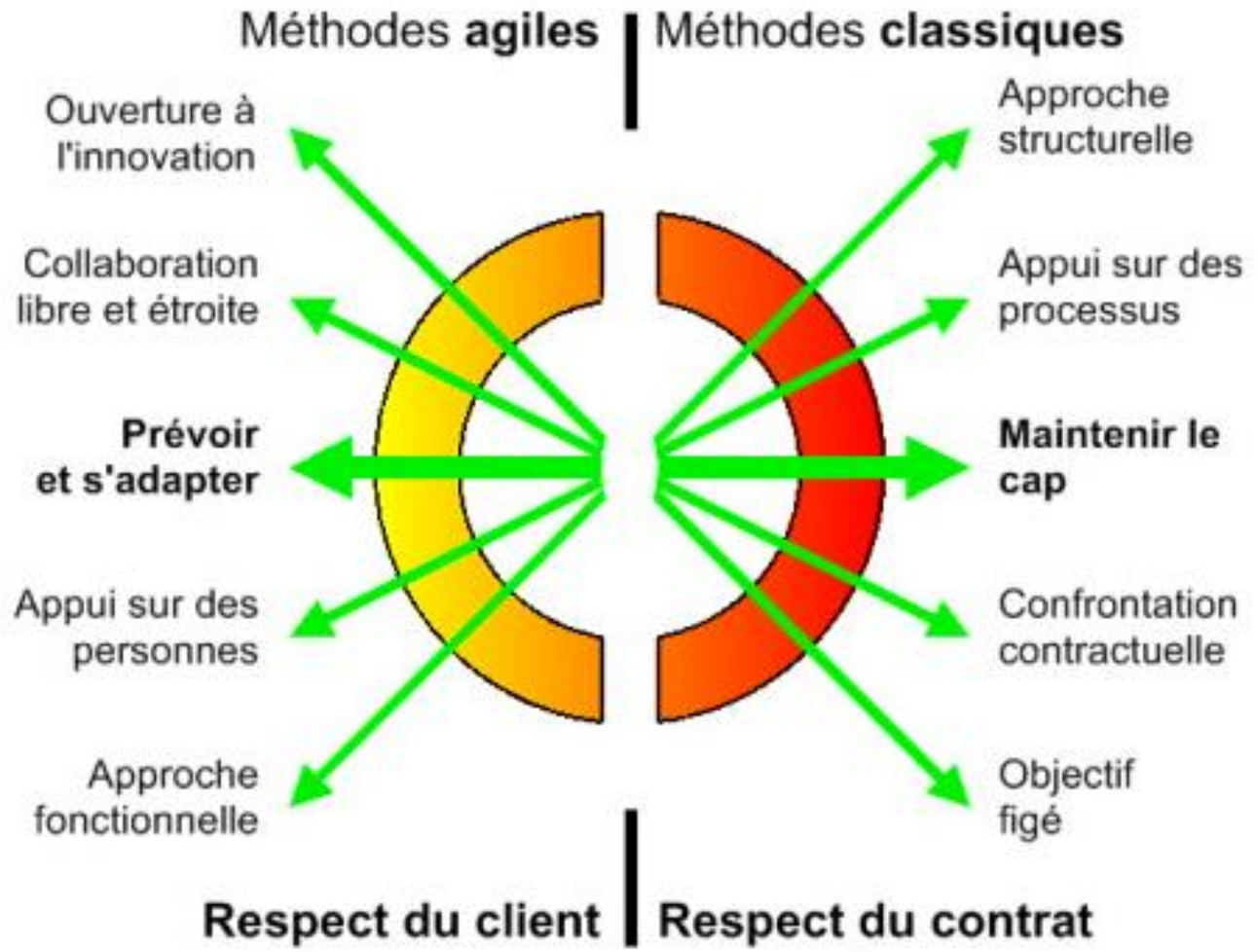
- Flexibilité
- Visibilité
- engagement des parties prenantes,
- meilleure qualité du produit

❑ Les limites

- Moins adapté aux projets à exigences fixes ou réglementés
- nécessite forte implication

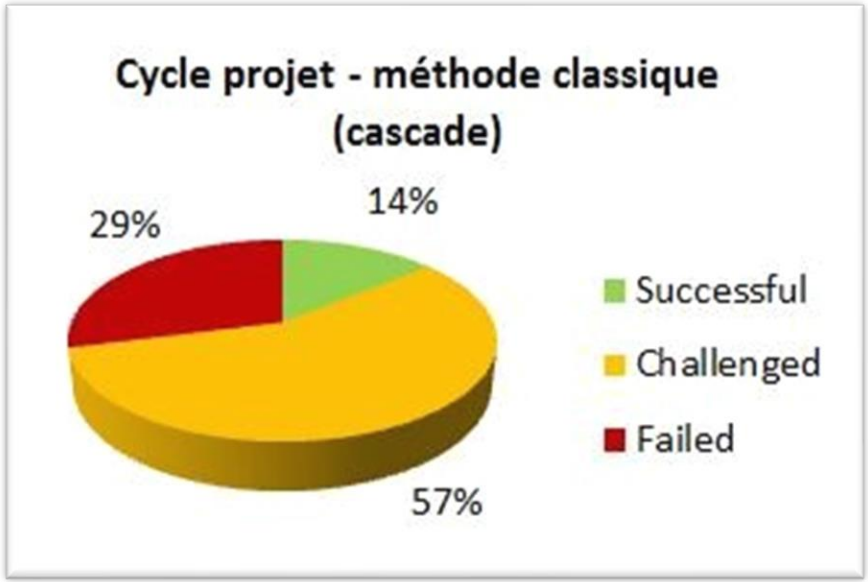
Les méthodes agiles de développement

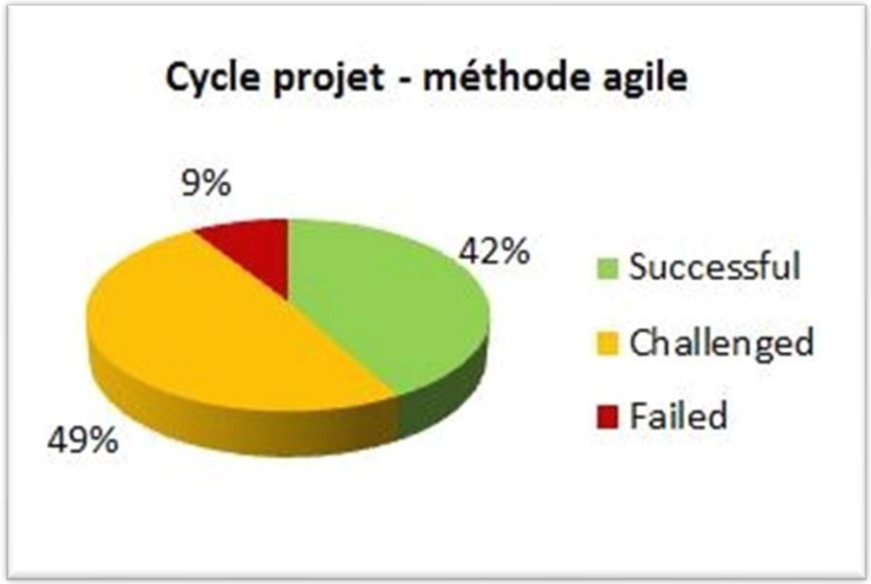
→ **Méthodes Agiles vs. méthodes Classiques**



Les méthodes agiles de développement

→ **Cycle de projet: méthodes agiles vs. les méthodes classiques**





QUALITÉ DU LOGICIEL

Qualité du logiciel

→ Définitions

La qualité est définie de manière générale par « l'aptitude d'un produit ou d'un service à satisfaire les besoins des utilisateurs ». Cette définition s'applique aux systèmes informatiques.

- La difficulté principale réside dans la capacité à exprimer ces besoins. On constate trop souvent que l'utilisateur n'est pas satisfait a posteriori ! Certes le cycle de vie en spirale, ou des approches par maquettage ou agiles réduisent ces risques, mais, il est fondamental de permettre à l'utilisateur (et/ou au client) d'exprimer ces exigences; ce sur quoi il va juger la qualité du système.

- La qualité est un processus qui dépasse la notion de logiciel. On peut l'appliquer à tout processus industriel pour peu qu'on souhaite l'améliorer => **Améliorer quoi ?** → le **produit**, le **service**, la **façon de développer le produit**, les **délais**, les **coûts**, … etc. Peu importe, ce qui compte c'est la volonté de mesurer, observer, contrôler, apprendre pour améliorer quelque chose.

Qualité du logiciel

→ Les mesures de la qualité d'un logiciel

Les premières études systématique de la qualité des logiciels datent de **1977**.

Depuis des modèles de qualités se sont développés, mais on retrouve toujours 3 niveaux :

- Les **facteurs** qualité: expression des exigences (point de vue externe, client)
- Les **critères** qualité: caractéristiques du produit (point de vue interne, technique)
- Les **métriques**: ce qui permet de mesurer un critère

Qualité du logiciel

❑ Les facteurs de qualité du logiciel

Conformité	Satisfaire aux spécifications - il faut qu'elles existent
Robustesse	Capacité à éviter les interruptions de fonctionnement (résistance aux pannes, capacité de tolérance aux pannes, correction d'erreurs, etc.).
Efficacité	Optimisation des ressources (cpu, E/S, mémoire, etc.)
Sécurité	Surveiller, contrôler, interdire les accès
Maniabilité	Minimiser l'effort d'apprentissage de l'utilisation du système
Maintenabilité	Minimiser l'effort pour localiser et corriger les fautes
Testabilité	Minimiser, automatiser l'effort de test
Adaptabilité	Minimiser l'effort d'évolution du système
Portabilité	Minimiser l'effort pour changer de plate-forme
Réutilisabilité	Optimiser la conception pour faciliter la réutilisation de parties du système
Interopérabilité	Garantir l'ouverture du système à d'autres systèmes ils doivent être appréciés par le client pour définir les points qu'il juge capitaux ou secondaires.

❑ Les principes de qualité du logiciel

Les approches mises en place visent à répondre aux critères de qualité, en intégrant les **principes** fondamentaux de **programmation** tels que:

- ❑ La modularité
- ❑ L'auto-descriptivité (les commentaires)
- ❑ L'indépendance matérielle logiciel
- ❑ La concision, etc.

Mais aussi des « **bons** » principes **d'organisation** tels que:

- ❑ L'utilisation de standards
- ❑ La **traçabilité:** Cette dernière est la capacité à suivre dans le temps l'évolution d'un produit et de ses plans et de pouvoir associer les éléments de la solutions aux éléments du problème.

❑ Les métriques de qualité logicielle

Pour mesurer la qualité du logiciel, des **métriques** sont associés aux critères eux même rattachés aux facteurs. Ces métriques peuvent caractériser les qualités :

- ❑ Du produit
- ❑ Du processus de développement
- ❑ Du service rendu

Elles peuvent se faire par des mesures objectives (**comptages**) ou par des enquêtes d'opinion (« pensez-vous que les résultats soient présentés clairement ? - de **0 à 5** »).

Les exemples des métriques concernant le logiciel sont :

- ❑ Nombre de lignes de code, de fonctions, d'opérateur par fonction,… (concision)
- ❑ Nombre de lignes de commentaires, … (auto-descriptivité)
- ❑ Nombre des modules, nombre de liens avec d'autres modules - couplage (modularité)
- ❑ Nombre de chemins possibles dans une fonction (simplicité)
- ❑ Nombre de données en entrées, en sortie, fréquence...(Simplicité)