# GEN: TE1

## Problématique

| Petits logiciels | | Gros logiciels |
|---|---|---|
| Faciles à comprendre | ⟷ | Difficile |
| Coûts faibles | ⟷ | Coût énorme |
| Développé par une personne | ⟷ | Développement par équipe |
| Rarement documenté | ⟷ | Parfaitement documenté |

→ Définitions :
- Spécification : description du fonctionnement du système ⇒ QUOI ?
- Conception : modélisation du système ⇒ COMMENT on va l'implémenter ?
- Implémentation : codage du système.
- Tests.
- Maintenance.

→ Crise du logiciel des années 70 : (cycles de vie en cascade) évolutions trop rapides, communication user ⟷ dév. complexe, ...

Pt. de vue user :
- Les logiciels ne correspondent souvent pas aux besoins.
- Changements du client difficiles à intégrer.

Pt. de vue développ. :
- Maintenance complexe ⇒ coûteuse.
- Trop d'erreurs.
- Performances inacceptables.
- Rarement portables.

Pt. de vue gestion :
- Difficilement réutilisables.
- Coûts imprévisibles.
- Délais dépassés.

→ Promotion de la phase d'analyse des besoins (QUOI ?).

→ Naissance du langage ADA.

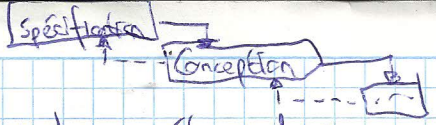
→ Génie logiciel ⇒ assure la qualité, des coûts contrôlés et des délais garantis.

→ Qualités d'un logiciel :

Qualités externes (pt. vue user) :
- Correction : le logiciel correspond à sa spécification (fonctions attendues).
- Fiabilité : degré de confiance, est-ce qu'il fonctionnera encore demain ?
- Robustesse : fonctionnement correct dans des situations inhabituelles (div. 0, ...).
- Performance : complexité des algorithmes, ...
- Ergonomie : facilité d'utilisation.

Qualités internes (pt. vue dév.) :
- Maintenabilité : corrective, adaptible (changements dans l'environnement), perfective (amélioration grâce à des fonctionnalités)
- Réparabilité : facilité des corrections.
- Evolutivité : facilité des changements de spécification.
- Réutilisation :
- Portabilité : indépendant de l'environnement.
- Interopérabilité : qualité à intéragir avec d'autres systèmes.

Qualités du processus de dévt :
- Productivité.
- Respect des délais.
- Echange d'informations.




Prise de risque : donner la priorité aux éléments possédant le + de risques.

## Cycles de vie

**Cascade (Waterfall)** : chaque étape du processus doit être terminée à une date précise avant que la suivante ne puisse débuter.

Avantages : développement très linéaire, planification aisée.

Désavantage : difficile de gérer les changements en cours de projet (retours en arrière), le client ne voit le système réalisé qu'à la fin, l'analyse des besoins est manquée.

**Cycle en V** : introduit les notions de découpage modulaire et de validation (quoi) / vérification (comment).

**Cycle en spirale** : implique fréquemment le client.

⚠ Systèmes connus ⇒ Waterfall.
Systèmes à risques ⇒ Spirale.

**Modèle incrémental itératif** : Planif. ⇒ Spécif. ⇒ Concept. ⇒ Implém. ⇒ Test ⇒ Evaluation

Chaque incrément donne lieu à un produit final.
⇒ Le client peut valider en tout temps ; erreurs identifiées rapidement.

→ **Méthode UP** : développement incrémental itératif + technologies objet.

Analyse OO : accent sur la recherche et description des objets du domaine du problème (style BDR).

Conception OO : accent sur les objets logiciels (UML).

Cas d'utilisation : décrit les fonctionnalités non-techniques du projet.

→ Les 4 phases de UP :

Initialisation : Compréhension de la finalité du projet ⇒ décider ou non de poursuivre le projet.
↓ → Etude de faisabilité, estimations globales, modélisation de domaine.

Elaboration : Analyse des besoins, planification, Gestion des risques,
↓ → Réalisation de l'architecture, rédaction des cas d'utilisation, planification des itérations.

Construction : Développement itératif.
↓ → Réalisation d'un module, génération d'un sous-ensemble exécutable, tests, analyse détaillée.
⇒ L'itération se termine quelque soit le résultat à la date planifiée (replanification de l'it. suiv.)

Transition : bêta tests et déploiement
→ Livraison finale.

⚠ Activités : opérées au sein d'une itération (dessiner un diagramme, définir un cas d'utilisation, ...).
Disciplines : regroupement des activités (spécification, conception, ...).
Artefacts : produits liés à une discipline (diagramme, code graphique, plan de test, ...)

## Modèle des cas d'utilisation

→ Modèle des cas d'utilisation : identifier et enregistrer les besoins du système ; spécification.

Acteurs : personne physique ou appareil qui interagit avec le système.
Scénario : suite d'actions et d'interactions entre les acteurs et le système.
Pré-conditions : ce qui doit être vrai avant le début du scénario.
Post-conditions : ——— après la fin ———.

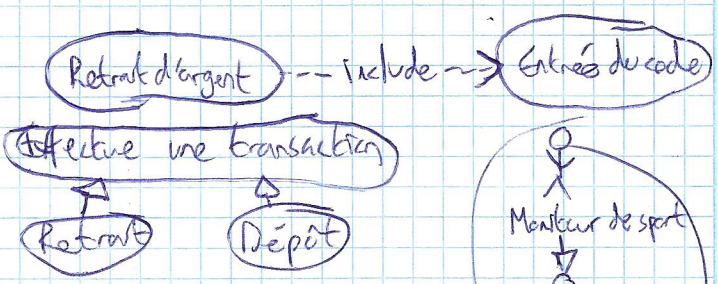
Scénario principal (succès) : 1. blabla.
2. blabla.

Extensions : scénarios d'échecs 2a. Code invalide. *a. A tout moment, si...
1. blabla
2. blabla
1. ...

Acteurs principaux : ceux qui déclenchent un cas d'utilisation.
Acteurs secondaires : sollicités pour accomplir tel ou tel cas d'utilisation.
Acteurs externes : concernés par l'accomplissement d'un cas d'utilisation, sans interagir avec
⇒ avides, ...

Relations :

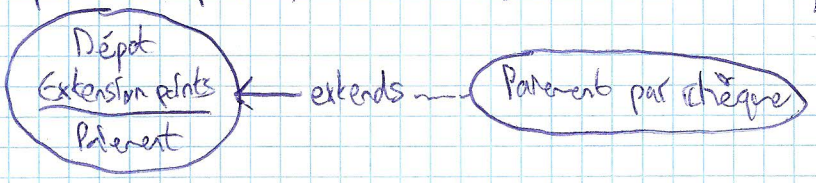
include ⇒ factorisation, décomposition.



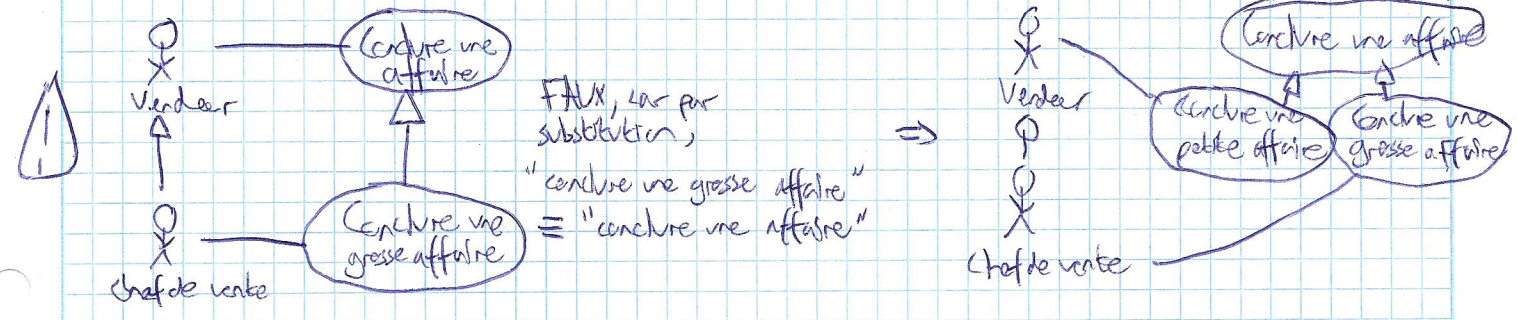

generalize ⇒ sorte... sorte...




extends ⇒ "modules", comportements facultatifs ou conditionnels.




le moniteur de sport "est un" moniteur de ski avec + de compétences.




MVC : modèle indépendant de l'application, contrairement à la vue et au contrôleur.
extends Observable ⇒ modèles. monModele.addObserver(maVue); setChanged(); notifyObservers();
implements Observer ⇒ vues. public void update(Observable o, Object arg);