Génie Logiciel

Chapitre 1:

Introduction au Génie Logiciel

Niveau: 3ième année Licence informatique

Année: 2023/2024

Introduction

Logiciel=>

Introduction

- **Un logiciel** est une suite **d'instructions** (*programmes*) et des **données** (fichiers, images, BDD…) relatives à des traitements effectués automatiquement par un appareil informatique pour accomplir une tâche donnée.
- un certain nombre de ***documents*** se rapportant à ces programmes et sont nécessaires à leur installation, utilisation, développement et maintenance**:** spécifications, schémas conceptuels, jeux de tests, mode d'emploi, ...

Introduction

- Des logiciels partout!!
  - La vie quotidienne est inimaginable sans logiciel: applications de bureautique, loisirs, internet, gestions, administration, industrie …etc.

Introduction

- Des logiciels partout!!

Les systèmes informatiques:

- 80 % de logiciel (source des problèmes).
- 20 % de matériel(relativement fiable /standardisé).

Introduction

- Des logiciels partout!!

PAR CONSÉQUENT

- Notre vie dépend très fortement de la qualité des logiciels qui la gèrent.
- Les problèmes liés à l'informatique sont essentiellement des problèmes de logiciel.

Introduction

- Impacts positifs des logiciels:
  - Amélioration de traitement de données (texte, vidéo , bureautique..)
  - Accélération du stockage, restauration et traitement des informations (gestion des BDD).
  - Introduction des nouveaux loisirs (les jeux …).
  - Augmenter la productivité (production de voitures).
  - Combler les distances: applications réseaux et internet.
  - Un logiciel est inlassable : il exécute ses tâches de manière constante.
  - Remplacer l'être humain dans les taches:
    - Rapides et sensibles (exactes): procédés industriels.
    - Dangereuses: opérations militaires.
    - temps réels: chaine de production.
  - …. etc

Introduction

- Et si le logiciel est mal fait?

- Plusieurs incidents plus ou moins dangereux ont été marqués à cause des logiciels mal faits.

Introduction

- Impacts négatifs des logiciels:
  - Le bug 2000: Quand l'an 2000 fut sur le point d'arriver, personne ne pouvait vraiment prédire ce qui allait se passer car les dates sont codés sur 2 chiffres.

Est resté dans l'histoire comme un énorme chantier de correction plutôt qu'une catastrophe

Introduction

- Impacts négatifs des logiciels:
  - Le bug du Mariner-1 le 22 juillet 1962: Une fusée spatiale (pour explorer la planète venus) a dérouté de sa trajectoire à cause d'une formule mathématique qui a été mal transcrite en code source. l'officier de sécurité de champ de tir commanda sa destruction 294 s après son lancement.

Introduction

- Impacts négatifs des logiciels:

  - Therac-25 accélérateur médical (1985) : La machine était destinée à soigner des malades. À cause d'un bug sur le déclenchement des radiations, au moins cinq personnes ont trouvé la mort.

Introduction

- Impacts négatifs des logiciels:
  - 1991, pendant la guerre du golfe : Un missile américain tue 22 soldats américains au lieu d'intercepter un missile ennemi. Cause : une erreur de fonction d'arrondi.

Introduction

- Impacts négatifs des logiciels:
  - En 1996, la fusée Ariane 5 – Vol 501: s'est brisée et explosée en vol seulement 36 secondes après le décollage. L'incident, dû à un dépassement d'entier dans les registres mémoire des calculateurs électroniques utilisés par le pilote automatique.

Développement Logiciel

Rappel: développement logiciel

- Le développement est la transformation d'une idée ou d'un besoin en un logiciel fonctionnel.

- L'idée est produite par un client (utilisateur) et développée par un fournisseur

client

Idée ou besoin

Exploitation du logiciel

Fournisseur

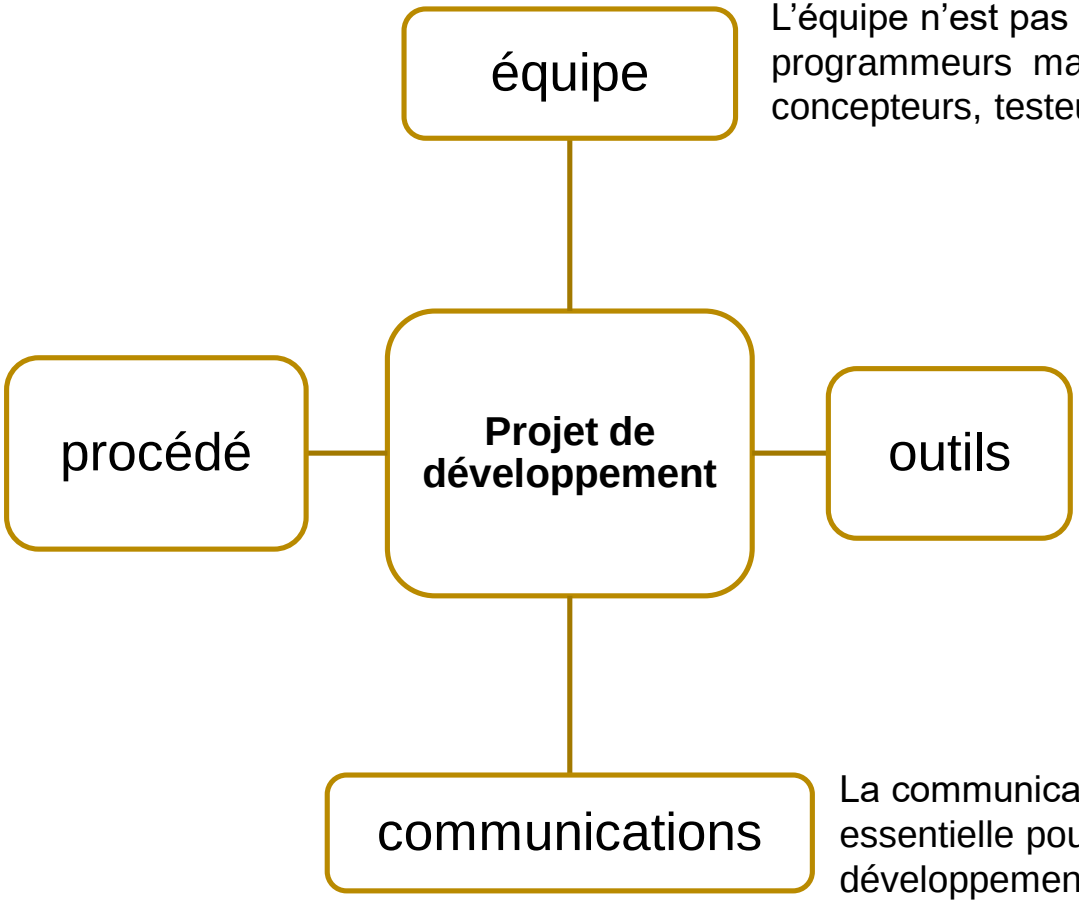
Développement

maintenance

Rappel: développement logiciel

- Le développement de logiciels n'est pas une opération facile. Il comprend un ensemble d'activités.
- La **programmation (le codage) n'est pas le développement** mais **une** des activités du développement.
- Il n'y a pas une seule façon de développer un logiciel donné mais plusieurs.
- Les projets de développement sont souvent longs et coûteux (50 % des coûts dans la maintenance).
- Les projets de développement font souvent intervenir plusieurs personnes de compétences différentes.

Rappel: développement logiciel



L'équipe n'est pas uniquement composée de programmeurs mais d'autres acteurs : chefs de projets, concepteurs, testeurs, …

compilateurs, environnements de tests, éditeurs, Framework…

La communication (dans l'équipe et avec le client) est essentielle pour faire réussir un projet de développement

Rappel: développement logiciel

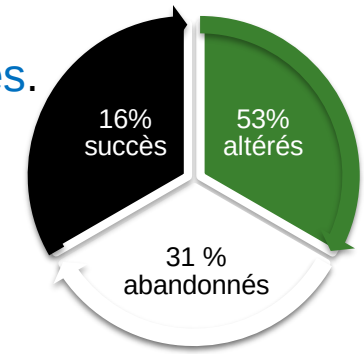
Plusieurs difficultés caractérisent le développement de logiciels :

- Les clients arrivent difficilement à décrire leurs besoins de façon assez claire pour les fournisseurs.
- Les besoins sont en constante évolution ainsi que l'environnement.
- Le logiciel est immatériel : on ne peut pas le toucher, seulement l'utiliser.
- Différence de langage entre les personnes techniques et non techniques.
- Difficulté de découvrir les erreurs avant la livraison du produit.
- Le piratage de logiciels cause un énorme préjudice pour les fournisseurs.

Rappel: développement logiciel

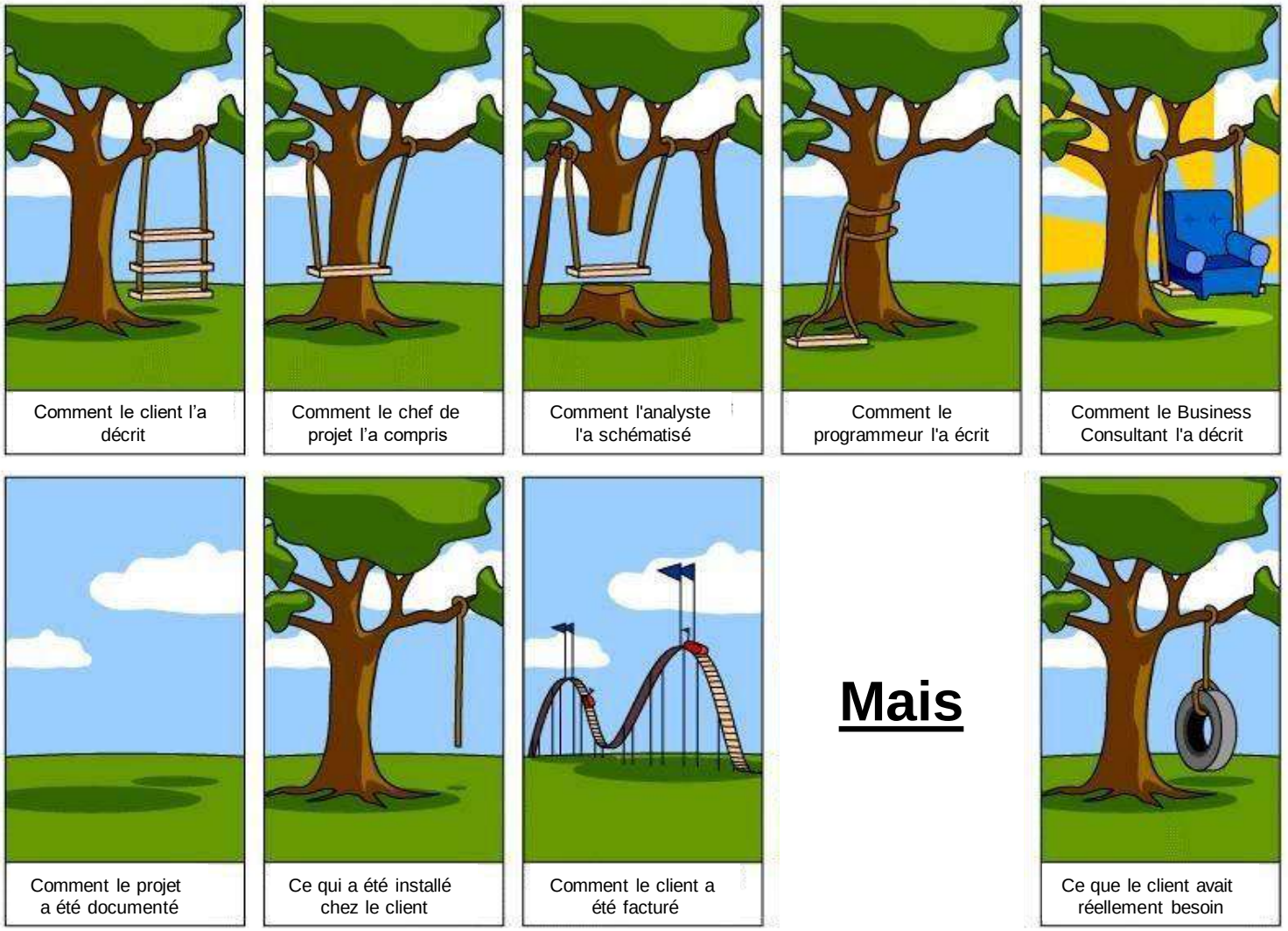
Une étude menée par **STANDISH GROUP*** sur ***8380 projets*** révèle que :

- **16%** des applications sont construites avec succès.
- **53%** des projets aboutissent mais posent des problèmes tels :
  - ✓ la diminution des fonctionnalités fixées de départ,
  - ✓ le non-respect des délais spécifiés ou
  - ✓ encore l'augmentation des coûts.
- **31%** des projets sont purement et simplement abandonnés.



Société américaine de recherche et de conseil en management de projets informatiques

Rappel: développement logiciel



Rappel: développement logiciel

- Qu'est ce qu'un bon logiciel pour le client ?

Rappel: développement logiciel

- Qu'est ce qu'un bon logiciel pour le client ?

Fait ce qu'on lui demande de faire

Respecte les critères de qualité

Livré dans les délai

Peu couteux

Règle du CQFD : Coût Qualité Fonctionnalités Délai.

Rappel: développement logiciel

- Qu'est ce qu'un bon logiciel pour le fournisseur?

Rappel: développement logiciel

- Qu'est ce qu'un bon logiciel pour le fournisseur?

Fait ce qu'on lui demande de faire

Respecte les critères de qualité

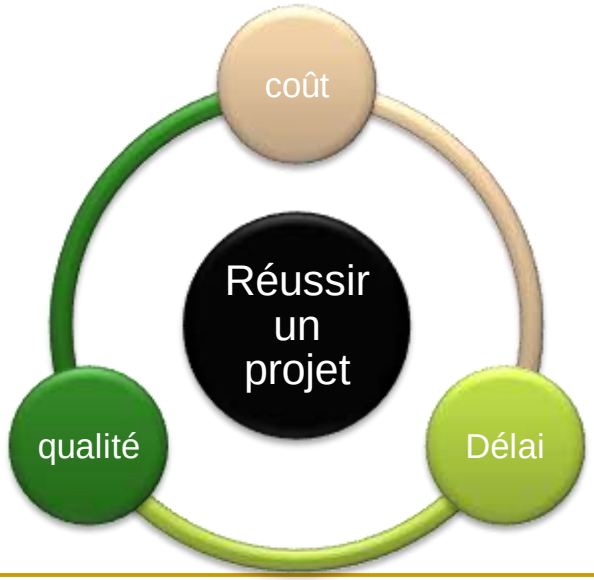
Minimiser les délai

Minimiser les couts

Maximiser les gains

Rappel: développement logiciel

Penser qualité:



Rappel:

La séance passée…

La crise du Logiciel

Rappel: la crise du logiciel:

A la fin des années 60, apparut la crise du logiciel. La construction de logiciels restait dans le domaine de **l'artisanat** et du **folklore**, où chacun y allait de sa petite recette qui ne correspond pas aux grands systèmes. Cela a conduit à :

- Des logiciels qui ne satisfont pas les attentes des clients.
- Les équipes de programmeurs sont débordées confrontées à des problèmes de communication.
- Non respect des délais et des coûts. certains projets ont été abandonnés (Le projet **TAURUS** de la **bourse de Londres**)
- Maintenance trop chère car trop difficile.
- Temps de réponse trop long!
- Applications non évolutives.
- Convivialité inexistante.
- … etc

Le Génie Logiciel

Deux conférences célèbres du comité scientifique de l'**O.T.A.N** (**NATO**).

- à Garmisch, en Allemagne, du 7 au 11 octobre 1968,
- et à Rome, en Italie, du 27 au 31 octobre 1969.

Rappel: le génie logiciel

- Le **génie logiciel** (ou ***Software engineering***) est un domaine de l'ingénierie qui permet la conception, la réalisation et la maintenance des systèmes logiciels de qualité.

- le génie logiciel est l'art de produire de bons logiciels. Il est basé sur des méthodologies, des techniques et des outils qui permettent de produire des logiciels de **grande ampleur**, c'est à dire, nécessitant des moyens humains importants et dont le déroulement est relativement long tout en satisfaisant le client et le fournisseur:
  - ✓ Dans des délais raisonnables
  - ✓ Avec des coûts acceptables
  - ✓ De qualité

Génie logiciel

Principes du Génie Logiciel

-Principes utilisés dans le Génie Logiciel

- **Généralisation** : regroupement d'un ensemble de fonctionnalités semblables en une fonctionnalité paramétrable (généricité, héritage)
- **Structuration** : façon de décomposer un logiciel (utilisation d'une méthode bottom-up ou top-down)
- **Abstraction** : mécanisme qui permet de présenter un contexte en exprimant les éléments pertinents et en omettant ceux qui ne le sont pas
- **Modularité** : décomposition d'un logiciel en composants discrets
- **Documentation** : gestion des documents incluant leur identification, acquisition, production, stockage et distribution
- **Vérification** : détermination du respect des spécifications établies sur la base des besoins identifiés dans la phase précédente du cycle de vie

Critères de qualité d'un logiciel

1-Fiabilité
2- Utilisabilité
3- Extensibilité
4- Portabilité
5- Performance
6- réutilisabilité
7- Interopérabilité
8- Mesurabilité
9- Facilité de maintenance

Critères de qualité d'un logiciel

1-la fiabilité:

1. **la validité (conformité ou utilité)**: logiciel conforme à ses spécifications, les résultats sont ceux attendus.
2. **La robustesse**: le logiciel fonctionne raisonnablement en toutes circonstances, même dans des conditions non prévues au départ

Mesures :

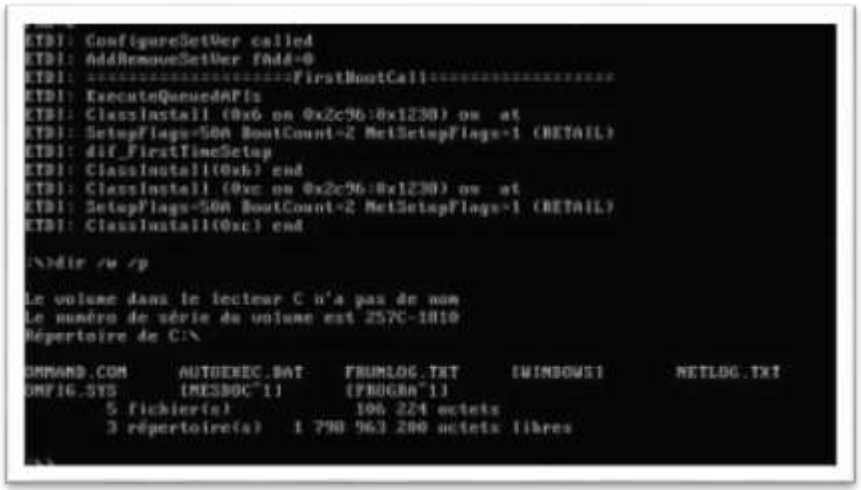
1. MTBF : Mean Time Between Failures
2. Disponibilité (pourcentage du temps pendant lequel le système est utilisable) et Taux d'erreur (nombre d'erreurs par KLOC)

Type	Indisponibilité (par an)	Pourcentage disponibilité	Classe
non géré	50 000 (34 jours, 17 heures et 20 min)	90 %	1
géré	5 000 (3 jours, 11 heures et 20 min)	99 %	2
bien géré	500 (8 heures 20 minutes)	99,9 %	3
tolérance fautive	50 (un peu moins d'une heure)	99,99 %	4
haute disponibilité	5 minutes	99,999 %	5
très haute disponibilité	0,5 (30 secondes)	99,9999 %	6
très grande haute disponibilité	0,05 (3 secondes)	99,99999 %	7

Critères de qualité d'un logiciel

2- utilisabilité :

- Aptitude à communiquer de façon efficace avec l'utilisateur
- L'utilisateur doit:
  - connaitre et comprendre toutes les fonctionnalités offertes par le logiciel (facilité d'apprentissage).
  - apprendre comment les utiliser pour des fins donnés (ergonomie et facilité d'utilisation).



Ergonomie logicielle

- Facilité d'utilisation, de compréhension et d'apprentissage
- Adapter le logiciel à l'utilisateur pour le rendre agréable, efficace et intuitif.

- Objectifs
  - ✓ Améliorer la productivité et la satisfaction des utilisateurs
  - ✓ Réduire les erreurs et le temps d'apprentissage
  - ✓ Favoriser une utilisation fluide et cohérente

- Principes essentiels
  - ✓ Simplicité : interface claire, menus limités
  - ✓ Cohérence : mêmes règles partout
  - ✓ Lisibilité : textes et icônes compréhensibles
  - ✓ Feedback : retour visible du système
  - ✓ Prévention des erreurs : messages d'avertissement, confirmations

Critères de qualité d'un logiciel

3- Extensibilité: Aptitude d'un logiciel à être enrichi et mis à jour sans rejeter le logiciel ou recommencer à 0.

4- Portabilité: Un même logiciel doit pouvoir fonctionner sur plusieurs machines (plates formes), c'est-à-dire Rendre le logiciel indépendant de son environnement d'exécution.

5- Performance: Les logiciels doivent satisfaire aux contraintes de temps d'exécution et d'occupation d'espace mémoire. (veillez à la complexité des logiciels)

Critères de qualité d'un logiciel

6- réutilisabilité :

Aptitude d'un logiciel (ou de certaines de ses parties) à être utilisé plusieurs fois dans des contextes différents.

On peut espérer des gains considérables car dans la plupart des logiciels :

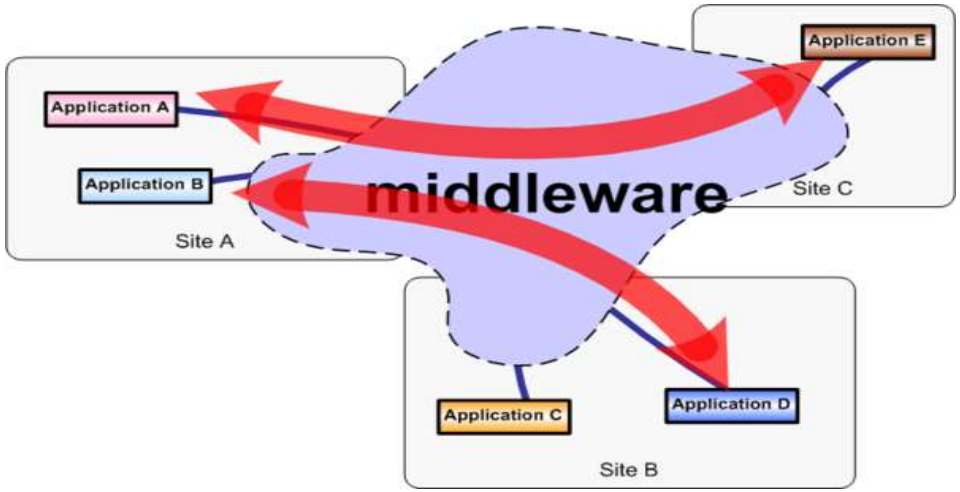
- 80 % du code est du « tout venant » qu'on retrouve à peu près partout.
- 20 % du code est spécifique.

Critères de qualité d'un logiciel

7- Interopérabilité :

Aptitude d'un programme à fonctionner correctement en collaboration avec d'autres logiciels.

- base de données.
- standardisation des formats d'échange de fichiers (XML...) et des protocoles de communication (CORBA...)
- Externaliser certaines fonctions en utilisant des Middleware avec une API (Application Program Interface) bien définie.



Critères de qualité d'un logiciel

8- Mesurabilité:

Aptitude d'un logiciel à être évalué:

- cout
- délai
- complexité
- ….

Critères de qualité d'un logiciel

9- Facilité de maintenance:

la maintenance absorbe une très grosse partie des efforts de développement

	Répartition effort dév.	Origine des erreurs	Coût de la maintenance
Définition des besoins	6%	56%	82%
Conception	5%	27%	13%
Codage	7%	7%	1%
Intégration Tests	15%	10%	4%
Maintenance	67%		

(Zeltovitz, De Marco)

Maintenance d'un logiciel

1. Corrective
2. Adaptative
3. Perfective

Maintenance corrective (obligatoire)

- Corriger les erreurs : de fiabilité et d'utilisabilité.
  - ✓ Identifier la défaillance, le fonctionnement.
  - ✓ Localiser la partie du code responsable.
  - ✓ estimer les impacts et faire les modifications.

- Attention:
  - ✓ La plupart des corrections introduisent de nouvelles erreurs.
  - ✓ Les coûts de correction augmentent exponentiellement avec le délai de détection.
  - ✓ La maintenance corrective donne lieu a de nouvelles livraisons (release)

Maintenance adaptative (mise à jour nécessaire)

- Ajuster le logiciel pour qu'il continue à remplir son rôle compte tenu de l'évolution des:
  - ✓ Environnements d'exécution.
  - ✓ Fonctions à satisfaire.
  - ✓ Conditions d'utilisation.

- Exemple : changement de SGBD, de machine, de taux de TVA, an 2000, euro...

Maintenance perfective (optionnelle)

- Permet d'accroitre/améliorer les possibilités du logiciel.

- Exemple
  - ✓ Les services offerts,
  - ✓ Interface utilisateur,
  - ✓ les performances … etc.

- Donne lieu à de nouvelles versions.

Facilité de la maintenance

- Objectif
  - ✓ Réduire la quantité de maintenance corrective (zéro défaut).
  - ✓ Rendre moins couteuses les autres maintenances.

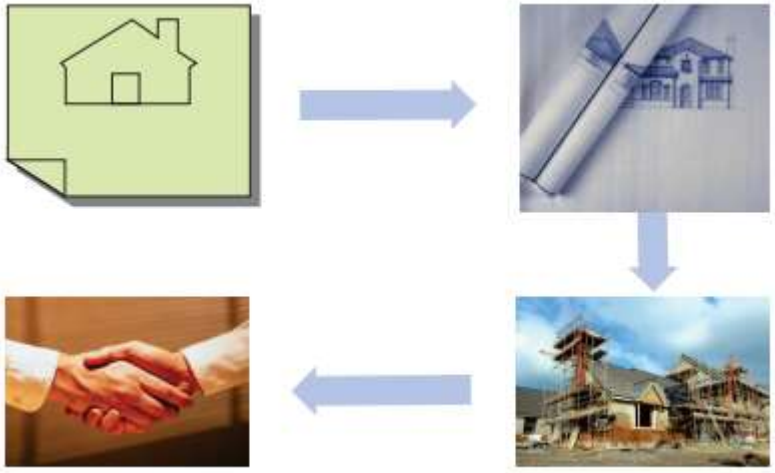
- Enjeux
  - ✓ Les coûts de maintenance se jouent très tôt dans le processus d'élaboration du logiciel.
  - ✓ Au fur et à mesure de la dégradation de la structure, la maintenance devient de plus en plus difficile.

- Solution
  - ✓ Réutilisabilité, modularité.
  - ✓ Renforcer les tests et la vérification.
  - ✓ Veiller à la complexité des algorithmes
  - ✓ Anticiper les changements à venir

Rappel

La séance passée…

Cycle de vie de logiciel



Cycle de vie de logiciel

La qualité du processus de fabrication est garante de la qualité du produit

- Pour obtenir un logiciel de qualité, il faut maitriser le processus d‘élaboration.
- Le développement d'un logiciel est décomposée en différentes étapes.
- La succession de ces étapes forme le **cycle de vie du logiciel**
- Il faut contrôler la succession de ces étapes.

Cycle de vie de logiciel

- Le cycle de vie d'un logiciel, désigne **toutes les étapes** du développement d'un logiciel, depuis la définition des besoins jusqu'à sa disparition.

- les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation (*10)

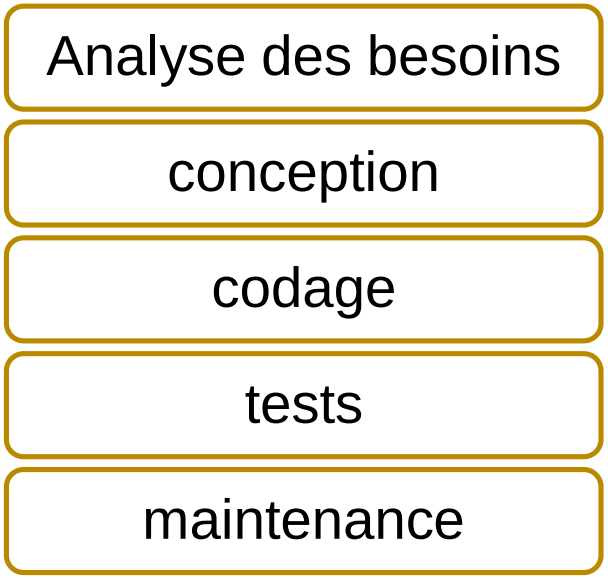
- Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la **qualité** du logiciel, les **délais** de sa réalisation et les **coûts** associés.

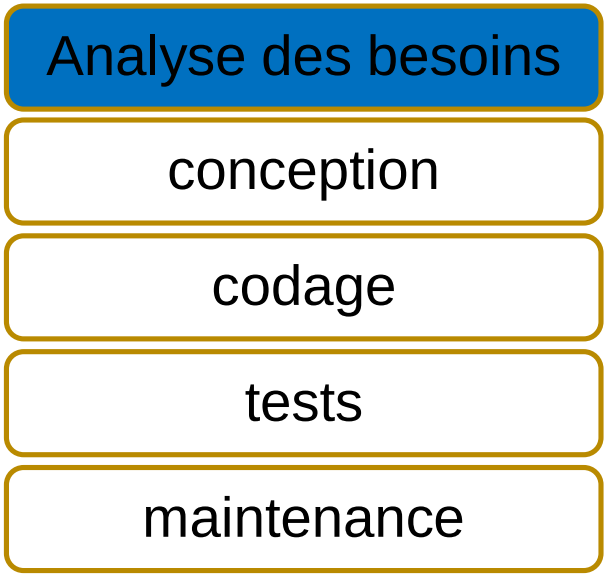
Activité, acteur et livrable

- Chaque projet de développement est composé de plusieurs activités.
- Chaque activité est conduite et réalisée par plusieurs acteurs.
- Une activité a des entrées et des sorties.
  - ✓ Les livrables font partie des sorties des activités.
- Les livrables sont des produits ou des documents produits par une activités et utilisé par les activités qui en dépendent.
  - ✓ Par exemple : document, planning, code source sont tous des livrables

Cycle de vie de logiciel

- Tous les projets de développement ont des activités communes:




Analyse des besoins
conception
codage
tests
maintenance

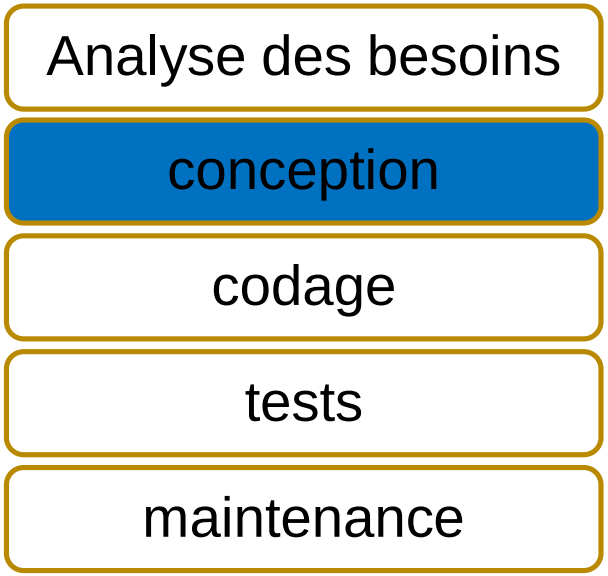
Analyse des besoins (Cahier de charge)

- l'expression, le recueil et la formalisation des besoins
  - ✓ pour un nouveau projet
  - ✓ ou projet altéré: évolution, mise à jour ou une adaptation
- Étudier les différentes contraintes (le travail nécessaire): techniques, économiques, temporelles, …
- Estimer la faisabilité du projet.
- Cette phase est essentielle et difficile car le client et les développeurs ne parlent pas le même langage.
- Le livrable de cette phase est un document de spécification (**cahier de charge** c-à-d fonctionnalités, analyse des couts, planification, répartitions des taches….)

Analyse des besoins (2)-Contraintes

- Contraintes techniques: Technologies imposées (langages, bases de données, serveurs…), Compatibilité avec un système existant, Limitations matérielles (mémoire, processeur, réseau, capteurs, etc.).
  - ✓ Exemple : le logiciel doit fonctionner sur Android et iOS.
- Contraintes organisationnelles: Structure et fonctionnement de l'entreprise, Disponibilité des utilisateurs pour les tests, Processus internes à respecter.
  - ✓ Exemple : le logiciel doit s'intégrer dans le système de gestion déjà en place.
- Contraintes économiques: Budget limité, ressources humaines ou matérielles restreintes.
  - ✓ Exemple : le projet doit être livré avec une équipe de 3 développeurs maximum.
- Contraintes temporelles: Délais de livraison.
  - ✓ Exemple : la première version doit être prête avant la fin de l'année.
- Contraintes réglementaires ou légales: Normes à respecter, protection des données, sécurité, accessibilité.
  - ✓ Exemple : conformité au RGPD* pour les données personnelles.
- Contraintes ergonomiques et humaines: Facilité d'utilisation, accessibilité, formation des utilisateurs.
  - ✓ Exemple : le logiciel doit être utilisable par du personnel non informaticien.

* **Le RGPD signifie Règlement Général sur la Protection des Données**


Analyse des besoins
conception
codage
tests
maintenance

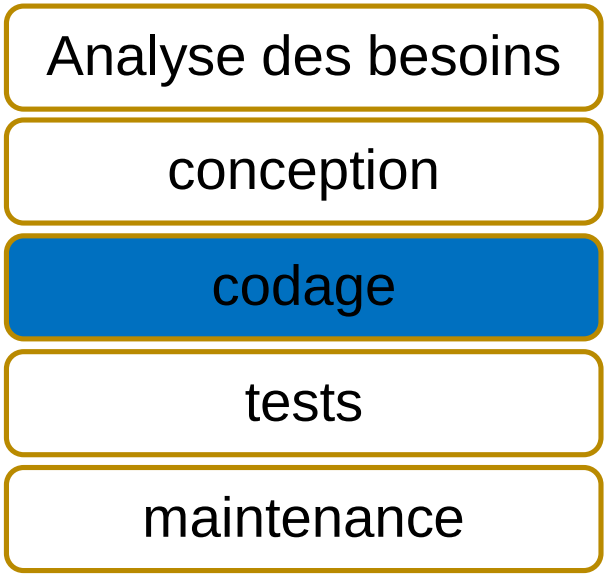
Conception

- Déterminer la façon dont le logiciel fournit les différentes fonctionnalités (les besoin) attendues.
- Conception architectural (générale):
  - ✓ déterminer la structure (l'architecture) générale du système (les blocs ou les modules et les interfaces entre eux).
  - ✓ Si le produit est centré sur l'utilisateur, la conception propose une ébauche de l'interface utilisateur.
- Conception détaillée:
  - ✓ Cette étape consiste à définir précisément (des diagrammes et des algorithmes) de chaque sous-ensemble du logiciel.
  - ✓ Recommandation: maximiser la cohésion à l'intérieur des composants et en minimiser le couplage entre eux,

Conception-Recommandation

- Maximiser la cohésion
  - ✓ Chaque composant fait une chose bien précise
- Minimiser le couplage
  - ✓ Les composants dépendent le moins possible les uns des autres

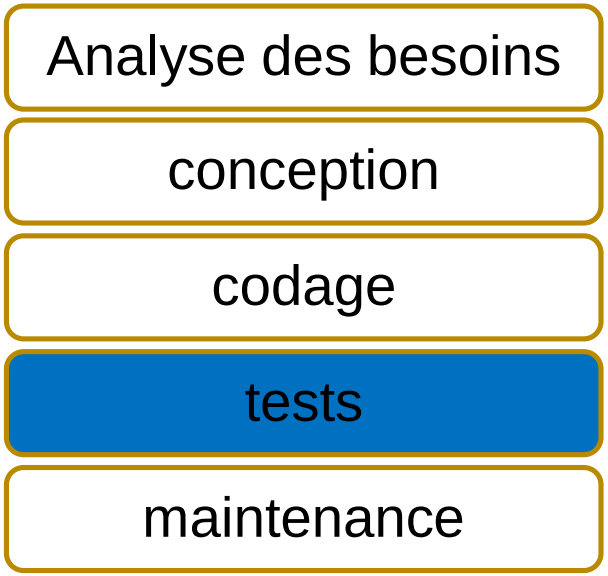
- C’est une bonne pratique qui rend le logiciel
  - ✓ plus modulaire,
  - ✓ plus facile à comprendre,
  - ✓ plus réutilisable,
  - ✓ et plus simple à maintenir.


Analyse des besoins
conception
codage
tests
maintenance

Codage (Implémentation)

- C’est la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.
- Les techniques de codage dépendent intrinsèquement du langage de programmation et du paradigme utilisé.

- Le **Pair Programming**, lecture croisée!!
  - ✓ pratique issue des méthodes Agile / eXtreme Programming (XP)
  - ✓ Driver: écrit le code.
  - ✓ Navigator: il relit, réfléchit à la logique, détecte les erreurs et propose des améliorations.
  - ✓ Echangent régulièrement leurs rôles.


Analyse des besoins
conception
codage
tests
maintenance

Tests

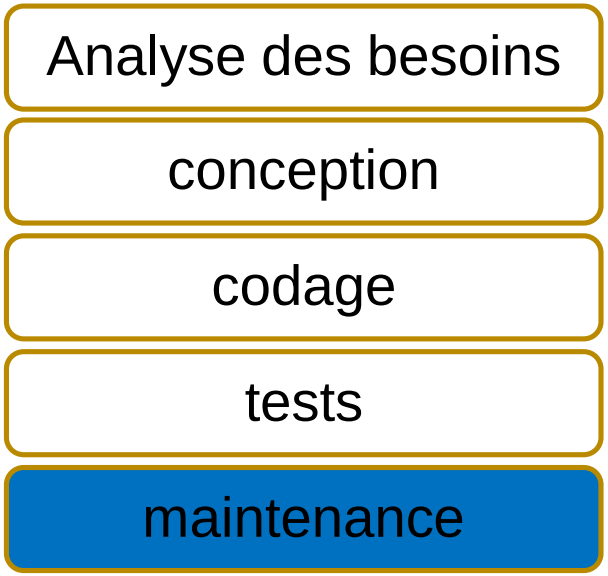
- Tester le logiciel sur des données d'exemple pour s'assurer qu'il fonctionne correctement.

  - ✓ **Tests unitaires**: Ils permettent de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.
  - ✓ **Tests d'intégration**: L'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel.
  - ✓ **Tests d'acceptation:** vérification de la conformité du
  - ✓ logiciel aux spécifications initiales (Validation par l'acheteur).

Livraison du produit

- Fournir au client une solution logicielle qui fonctionne correctement:

- **Installation** : rendre le logiciel opérationnel sur le site du client.
- **Formation** : enseigner aux utilisateurs à se servir du logiciel.
- **Assistance** : répondre aux questions des utilisateurs.


Analyse des besoins
conception
codage
tests
maintenance

Maintenance

Son but est:

- **Correctif**: corriger les éventuelles erreurs.
- **Évolutif**: Mettre à jour et améliorer le logiciel pour assurer sa pérennité.

Pour limiter le temps et les coûts de maintenance, il faut porter ses efforts sur les étapes antérieures.

Cycle de vie de logiciel

- **La séquence et la présence de chacune de ces activités dans le cycle de vie dépend du choix d'un modèle de cycle de vie entre le client et l'équipe de développement.**
- Un **procédé logiciel** (software **process**) est un ensemble d'activités conduisant à la production d'un logiciel.

Méthodologies

Les méthodologies classiques et les méthodes agiles

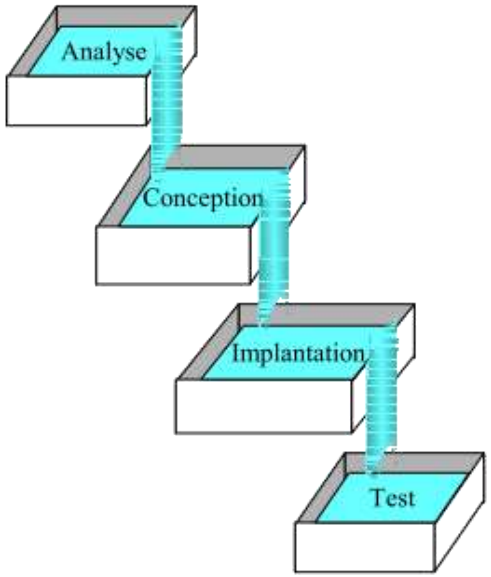
- **Méthodologies classiques**
  - ✓ Modèles stricts
  - ✓ Etapes très clairement définies
  - ✓ Documentation très fournie
  - ✓ Fonctionne bien avec les gros projets et les projets gouvernementaux
  - ✓ Exemple: Cascade, V, Incrémental, Spiral ..etc

- **Méthodes agiles**
  - ✓ Modèles incrémentaux et itératifs
  - ✓ Petites et fréquentes livraisons
  - ✓ Accent sur le code et moins sur la documentation
  - ✓ Convient aux projets de petite et moyenne taille
  - ✓ Exemple: Scrum, Extreme Programming (XP), Rapid Application Development (RAD)..etc

Méthodologies classiques

Modèle en cascade (chute d'eau)

- L'un des premiers modèles proposés (1970)
  - ✓ inspiré de l'industrie des bâtiments (pas de retour en arrière).
- chaque phase se termine à une date précise par la production de certains documents ou logiciels.
- Une phase ne peut démarrer que si la précédente est finie. Les résultats sont soumis à une revue approfondie et on ne passe à la phase suivante que s'ils sont jugés satisfaisants.
- il n'y a pas (ou peu) de retour en arrière.
- Adapté pour des projets de petite taille, et dont le domaine est bien maitrisé (des besoins stable, nouvelle versions d'un projet..).



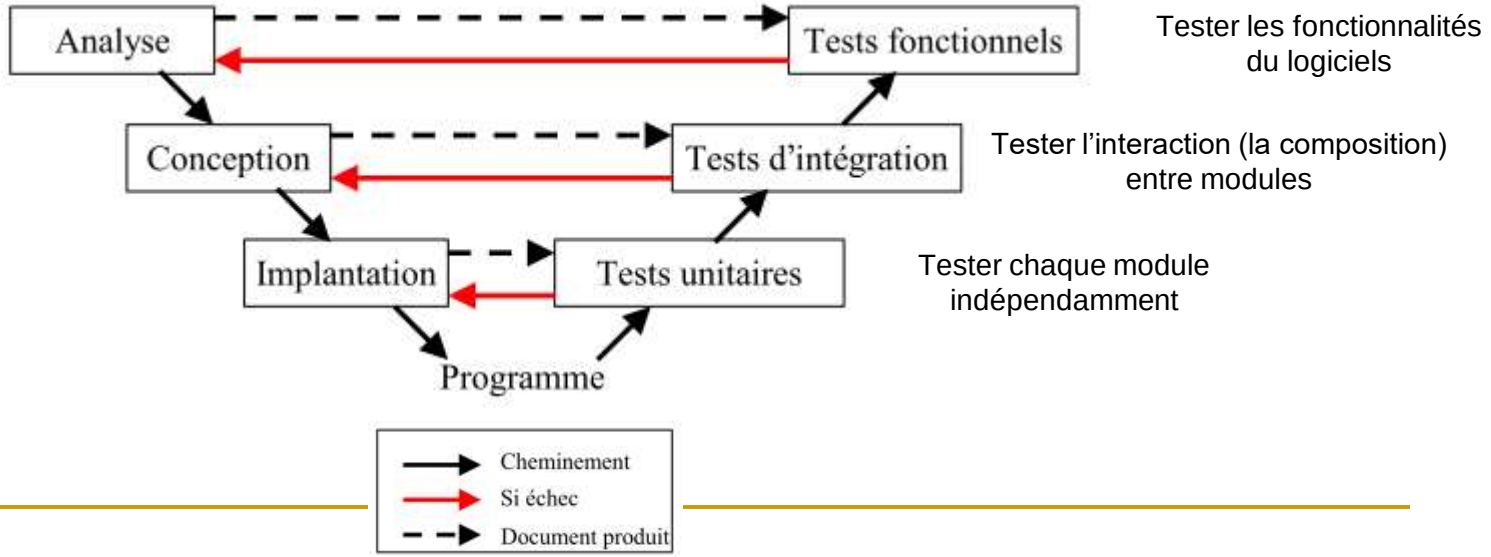
Remarque:
La phase de test intervient tard (découvrir des erreurs, ou changement des spécifications nécessite refaire tout le travail)

Modèle en V

- le plus connu et certainement le plus utilisé.
- Le développement des tests et du logiciel sont effectués de manière synchrone
  - ✓ Le test du produit se fait en parallèle par rapport aux autres activités.
- Chaque livrable (de chaque étape) doit être testable
  - ✓ (toute description d'un composant est accompagnée de tests qui permettront de s'assurer qu'il correspond à sa description).

Utiliser:

- Quand le produit à développer à de très hautes exigences de qualité.
- Quand les besoins sont connus à l'avance.

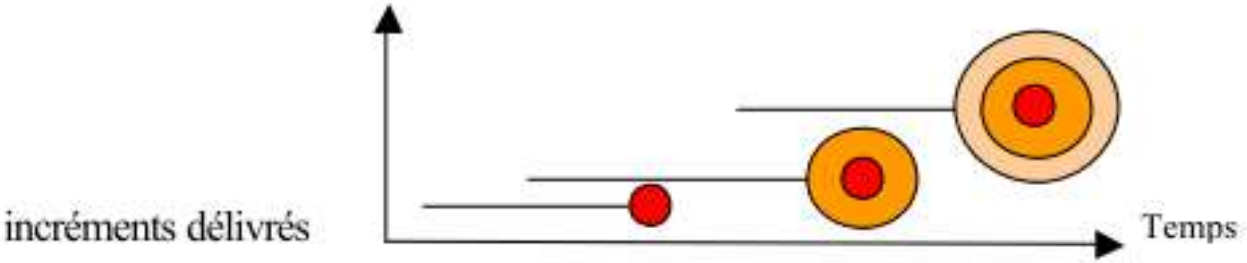


Modèle incrémental

- Concevoir et livrer au client un sous-ensemble minimal et fonctionnel du système
- On commence par développer les parties les plus stables, les plus importantes…
- Procéder par ajouts d'incréments minimaux jusqu'à la fin du processus de développement

Avantages:

- meilleure intégration du client dans le processus de développement.
- produit conforme à ses attentes.



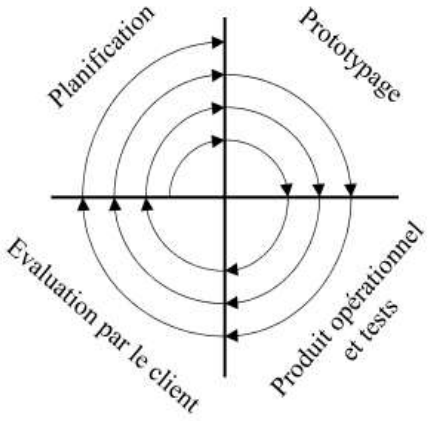
Le modèle en spiral (itératif):

- Le développement reprend les différentes étapes du cycle en V (évolutif). Par l'implémentation de versions successives, le cycle recommence en proposant un produit de plus en plus complet et robuste.
- Met cependant plus l'accent sur la gestion des risques:
  - ✓ Risques techniques (technologie non maîtrisée)
  - ✓ Risques de planning (retard possible)
  - ✓ Risques de coûts (budget dépassé)
  - ✓ Risques de performance ou de sécurité

- Avant de commencer une nouvelle itération, on réalise une analyse des risques.
- Nécessaire par le fait que, lors d'un développement cyclique, il y a plus de risques de défaire:

- Nécessite une très grande expérience.

- Utilisation:
  - ✓ Des grands projets complexes (aéronautique, médical, défense, etc.),
  - ✓ Les projets à fort risque ou incertitude technique.



Méthodologies Agiles

Les douze (12) principes agiles

1. Toujours satisfaire le client à travers des livraisons rapides et continues

2. Bien accueillir tous les changements même les tardifs

3. Livrer fréquemment un système fonctionnel

4. Les clients et les développeurs doivent collaborer

5. Conduire le projet autour d'équipes motivées

6. La meilleure méthode de faire circuler l'information c'est le contact direct entre collaborateurs

7. La première mesure d'avancement c'est un logiciel fonctionnel

8. Le développement doit être durable et à un rythme constant

9. La bonne conception et l'excellence technique augmentent l'agilité

10. Simplifier au maximum

11. Les meilleurs architectures, besoins et conceptions proviennent d'équipes qui s'organisent d'elles-mêmes

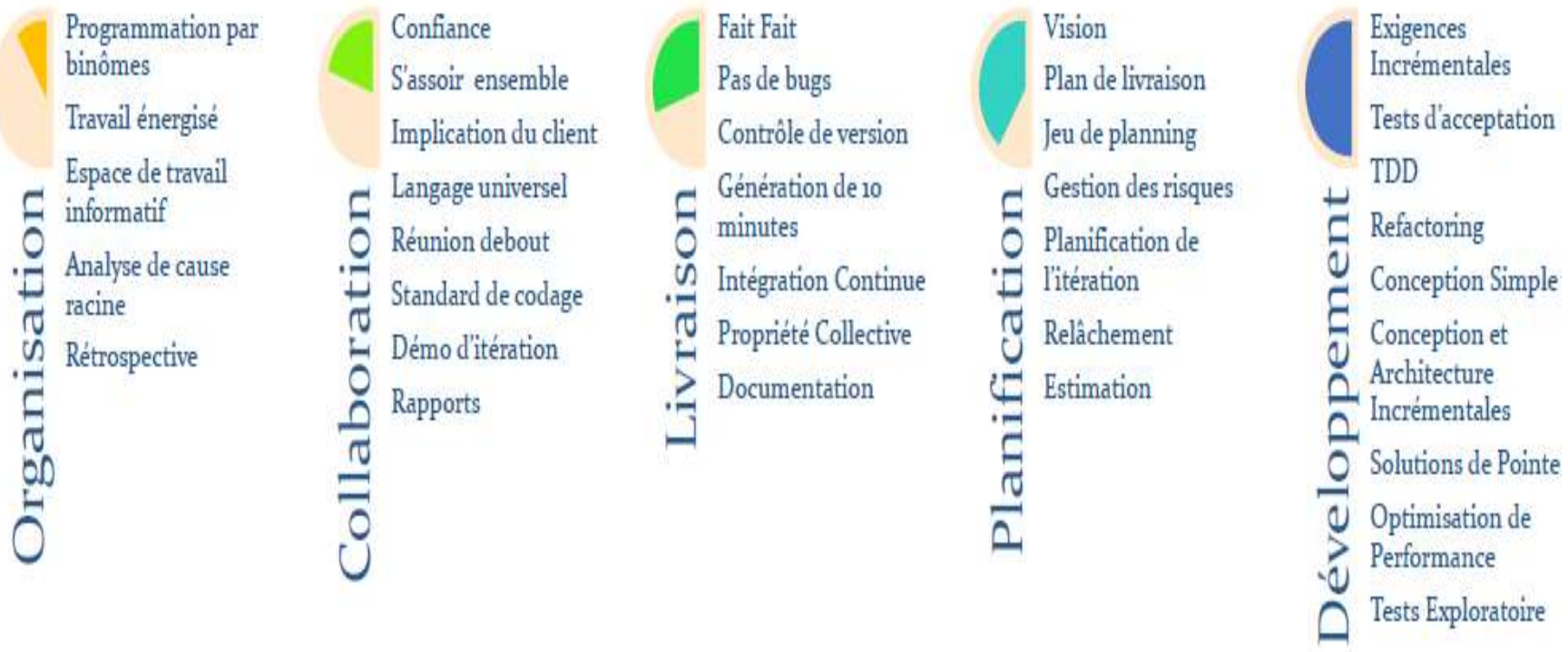
12. L'équipe s'améliore d'une manière autonome et régulière

Méthodologie XP(eXtreme Programming)

- Moyen léger, efficace, à bas risques, flexible, scientifique et amusant de développer des logiciels
- Destinée à des équipes de moyenne taille
- Avec des spécifications incomplètes et / ou vagues
- Le codage est le ***noyau*** de XP
- XP – Fondamentaux:
  - ✓ Implication massive du client
  - ✓ Test unitaire continu (TDD: Test-Driven Development)
  - ✓ Programmation par paires
  - ✓ Itérations courtes et livraisons fréquentes
- Méthodologie XP – Principales activités
  - ✓ Codage (noyau de XP)
  - ✓ Tests (pendant le codage)
  - ✓ Ecoute (le client ou le partenaire)
  - ✓ Conception (encore basée sur le codage)

Méthodologie XP(eXtreme Programming)

- Pratiques XP



Méthodologie XP(eXtreme Programming)

- Méthodologie XP – Inconvénients
  - ✓ Demande une certaine maturité des développeurs
  - ✓ La programmation par paires n'est toujours pas applicable
  - ✓ Difficulté de planifier et de budgétiser un projet
  - ✓ Stress dû aux devoir de l'intégration continue et des livraisons fréquentes
  - ✓ La faible documentation peut nuire en cas de départ des développeurs

Méthodologie Scrum

- Méthode agile de gestion de projet qui permet de développer un produit de manière itérative et incrémentale.
- Objectif: Livrer rapidement de la valeur au client en travaillant par courtes itérations appelées “sprints”.

- Les principes clés de Scrum
  - ✓ Itératif et incrémental : Le projet avance par petits pas (sprints) et le produit s’améliore à chaque cycle.
  - ✓ Autonomie de l’équipe : L’équipe s’auto-organise pour atteindre ses objectifs.
  - ✓ Transparence et communication :Tout le monde a accès aux informations sur l’avancement du projet.
  - ✓ Amélioration continue : L’équipe s’adapte à chaque sprint à travers des retours et rétrospectives.

Méthodologie Scrum

- **Simple**
  - ✓ Peut être combiné avec d'autres méthodes
  - ✓ Compatible avec les bonnes pratiques
- **Empirique**
  - ✓ Itérations courtes (**sprints**)
  - ✓ Feedback continu
- **Techniques simples**
  - ✓ Sprints de 2 à 4 semaines
  - ✓ Besoins capturés en tant que **user stories**
- **Equipes**
  - ✓ Auto-organisation
  - ✓ Multi-compétences
- **Optimisation**
  - ✓ Détection rapide des anomalies
  - ✓ Organisation simple
  - ✓ Requiert l'ouverture et la visibilité

mêlée

User Story

- Décrit ce que l'utilisateur veut faire et pourquoi
- Sert à capturer les besoins fonctionnels de manière simple, sans détails techniques.

- Forme typique :
  - ✓ En tant que [utilisateur], je veux [objectif] afin de [bénéfice].

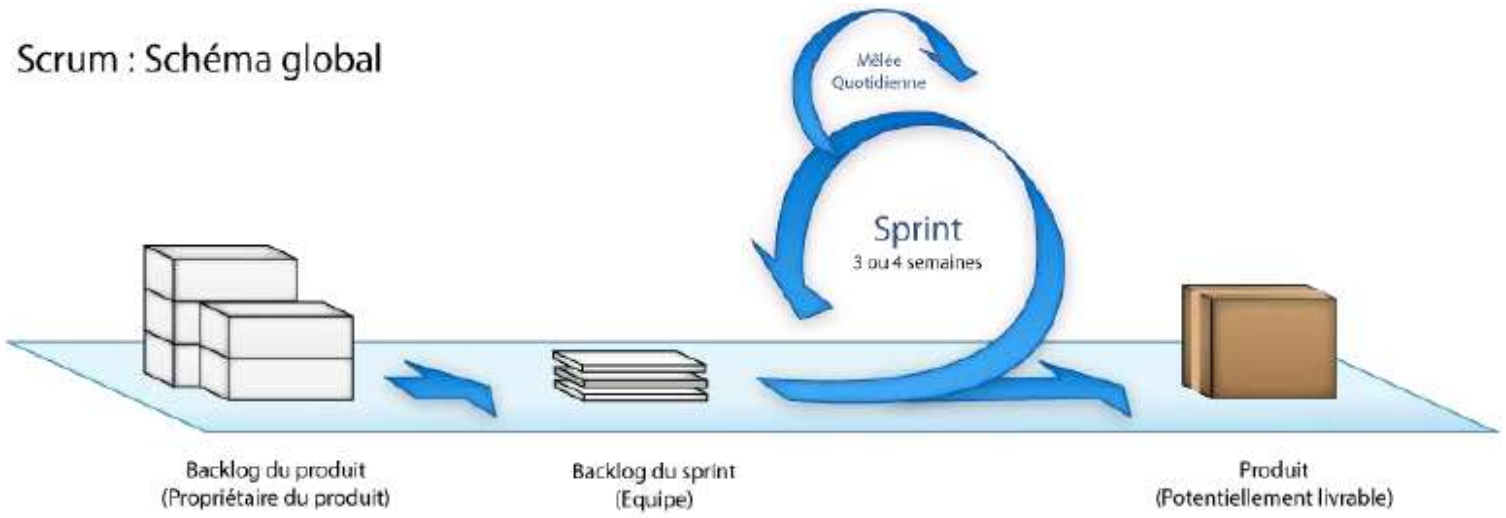
- Accompagnée de critères d'acceptation précisant quand elle sera considérée comme terminée.

- Elle aide l'équipe à comprendre les besoins réels et planifier les tâches dans les sprints,

- Exemple :
  - ✓ En tant qu'étudiant,
  - ✓ je veux pouvoir télécharger mes supports de cours,
  - ✓ afin de pouvoir les consulter hors ligne.

- Critères d'acceptation :
  - ✓ L'étudiant peut voir la liste des fichiers.
  - ✓ Le bouton "Télécharger" fonctionne pour chaque fichier.
  - ✓ Un message d'erreur apparaît si un fichier est manquant.

Méthodologie Scrum



Principes et Concepts

- Processus
  - ✓ Sprint
- Backlog
  - ✓ Product Backlog
  - ✓ Sprint Backlog
- Equipe
  - ✓ Scrum Master
  - ✓ Product Owner
  - ✓ Team
  - ✓ Users and stakeholders
- Réunions
  - ✓ Scrum quotidien
  - ✓ Planning
  - ✓ Revue
  - ✓ Rétrospective
- Suivi
  - ✓ Rapports
  - ✓ Vélocité

Méthodologie Scrum

Principes et Concepts

- Processus
  - ✓ Sprint : Un projet Scrum se compose de plusieurs sprints, chacun durant généralement 2 à 4 semaines.

- Backlog
  - ✓ Product Backlog: Liste globale et évolutive de toutes les fonctionnalités, améliorations et corrections à réaliser pour le produit.
  - ✓ Sprint Backlog: Sous-ensemble du Product Backlog, contenant les éléments choisis pour un sprint donné, avec leurs tâches détaillées.

- Equipe
  - ✓ Scrum Master: Garant du respect de Scrum, aide l'équipe à s'améliorer, supprime les obstacles.
  - ✓ Product Owner: Définit les besoins, fixe les priorités, représente le client.
  - ✓ Team: Groupe pluridisciplinaire qui réalise le travail (dev, test, design…).

Principes et Concepts

- Réunions
  - ✓ Scrum quotidien: 15 min chaque matin pour synchroniser l'équipe.
  - ✓ Planning: définir ce qui sera fait.
  - ✓ Revue: démonstration du produit terminé au client.
  - ✓ Rétrospective: réflexion interne pour s'améliorer.

- Suivi
  - ✓ Rapports: permettent de suivre le projet
  - ✓ Vélocité: mesure la quantité de travail accomplie par l'équipe pendant un sprint. aide à planifier les prochains sprints avec réalisme.

Méthodologie Scrum

- **Avantages**:
  - Méthode très simple et très efficace
  - Adoptée par les géants du marché : **Microsoft**, **Google**, **Nokia**..
  - Orientée projet contrairement à XP qui est orientée développement
  - Peut inclure d'autres activité venant d'autres méthodologies (surtout XP)

- **Inconvénients**
  - N'est pas 100% spécifique au GL
  - Difficulté de budgétiser un projet

Autres procédés

- Nombreuses autres processus:
  - **Processus Unifié (UP) :**
    - **Rational Unified Process (RUP)**
    - **Agile Unified Process (AUP)**
    - **Basic Unified Process (BUP)**
    - **Enterprise Unified Process (EUP)**
    - **Essential Unified Process (EssUP)**
    - **Open Unified Process (OpenUP)**
    - **Oracle Unified Method (OUM)**
  - **Ambler**
  - **Merise**
  - **SADT**
  - **HERMES. . .**
- Un autre classement: trois grandes familles :
  - **ascendante**
  - **descendante**
  - **Agile**

- **Noter bien**: Parfois:**Méthodologie = Langage + Processus**
  - **Exemple: UML & Y**

Quand utiliser un procédé X ?

- Nature du Projet
  - ✓
  - ✓
- Taille du Projet
  - ✓
  - ✓
- Nature du client
  - ✓
  - ✓
- Exigences du contrat
  - ✓
  - ✓
- Compétences de l'équipe
  - ✓
  - ✓

Quand utiliser un procédé X ?

- Nature du Projet
  - ✓ Projets critiques (ex. : aéronautique, médical) → méthodes structurées (Cycle en V, cascade).
  - ✓ Projets évolutifs ou incertains → méthodes agiles (Scrum, XP).
- Taille du Projet
  - ✓ Petite équipe / projet court → XP ou Scrum.
  - ✓ Grande équipe / projet complexe → méthodes plus formelles (Cycle en V, Rational Unified Process).
- Nature du client
  - ✓ Client disponible et impliqué → méthodes agiles.
  - ✓ Client peu disponible ou très institutionnel → méthodes classiques (spécifications figées).
- Exigences du contrat
  - ✓ Contrats fixes (budget, délais) → méthodes prédictives (cascade, V).
  - ✓ Contrats souples ou évolutifs → méthodes agiles.
- Compétences de l'équipe
  - ✓ Équipe expérimentée, autonome → agile (XP, Scrum).
  - ✓ Équipe moins expérimentée → approche structurée et documentée.

Maturité du processus de développement logiciel

Capability Maturity Model (CMM)

- Est un système défini par le **Software Engineering Institute (SEI)** de **l'Université Carnegie Mellon de Pittsburgh (Pennsylvanie)** pour le **ministère de la Défense américain**.

- Vise à améliorer le processus de développement logiciel.

- Permet à une organisation de mesurer son niveau de maturité et de faire évoluer sa capacité de développement logiciel.

Objectif

Évaluer et évoluer le processus de développement logiciel.

Capability Maturity Model (CMM)

- Comporte **cinq niveaux de maturité** constituent autant d'étapes sur le chemin menant à des processus matures,
  - ✓ c'est-à-dire conformes à un ensemble de bonnes pratiques observées à travers le monde dans des entreprises réputées pour bien gérer leurs processus.

- La conformité au modèle CMM est notamment requise pour contracter avec le département américain de la défense.

Capability Maturity Model (CMM)

- Ce modèle offre à une organisation cherchant à améliorer ses capacités de développement logiciel :

  - ✓ une structure d'évaluation de son niveau de maturité (évaluation).
  - ✓ un ensemble de procédures documentées pour améliorer son niveau de maturité.
  - ✓ un ensemble de processus de contrôle pour valider les étapes de cette progression.

- L'amélioration de la maturité d'un organisme signifie :
  - ✓ la réduction des délais de développement.
  - ✓ la réduction des coûts.
  - ✓ l'utilisation plus efficace des ressources.

Capability Maturity Model (CMM)

- décrit 5 niveaux de maturité des processus: **initial**, **reproductible**, **défini**, **maîtrisé** et **optimisé**.
- Le niveau 1 (initial/chaotique) est le niveau le plus bas (aucune amélioration n'a encore commencé).
- Le niveau 5 (optimisé) est le plus élevé.
- La majorité des organismes est actuellement au niveau 1.

Capability Maturity Model (CMM)

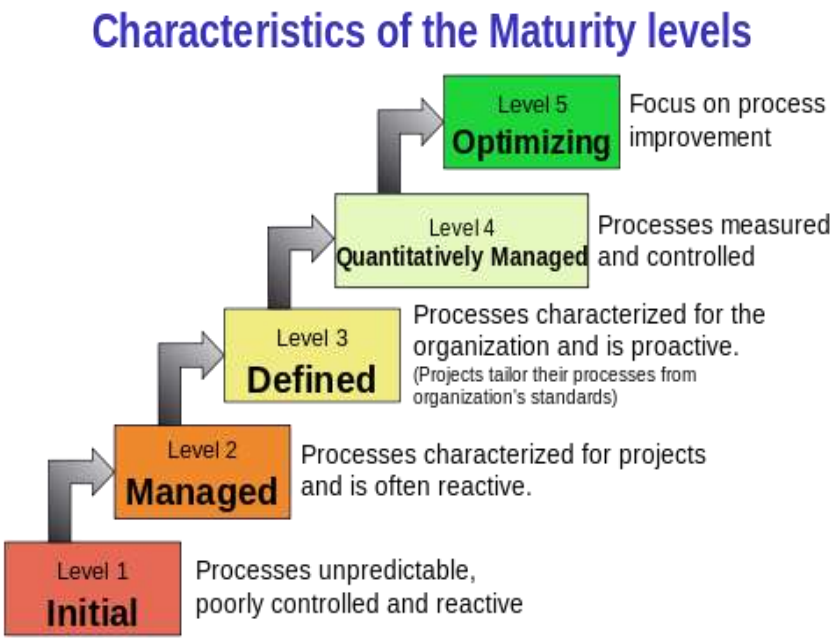
- Niveau 1 – Les processus ne sont pas organisés : chacun travaille à sa manière.
- Résultat : imprévisible, réactif et dépendant des individus.

- Niveau 2 – Des pratiques de base de gestion de projet existent.
- On peut reproduire certains succès, mais on reste surtout réactif.

- Niveau 3 – Les processus sont standardisés pour toute l'organisation.
- On travaille de façon cohérente et plus proactive.

- Niveau 4 – Les processus sont mesurés et contrôlés.
- Les décisions se fondent sur des indicateurs précis.

- Niveau 5 – L'organisation améliore continuellement ses processus.
- Objectif : innover et augmenter la performance.



Capability Maturity Model (CMM)

- une progression vers un niveau supérieur
  - ✓ améliore la qualité.
  - ✓ Réduit le cout de développement et de maintenance.
  - ✓ Améliore le délai.

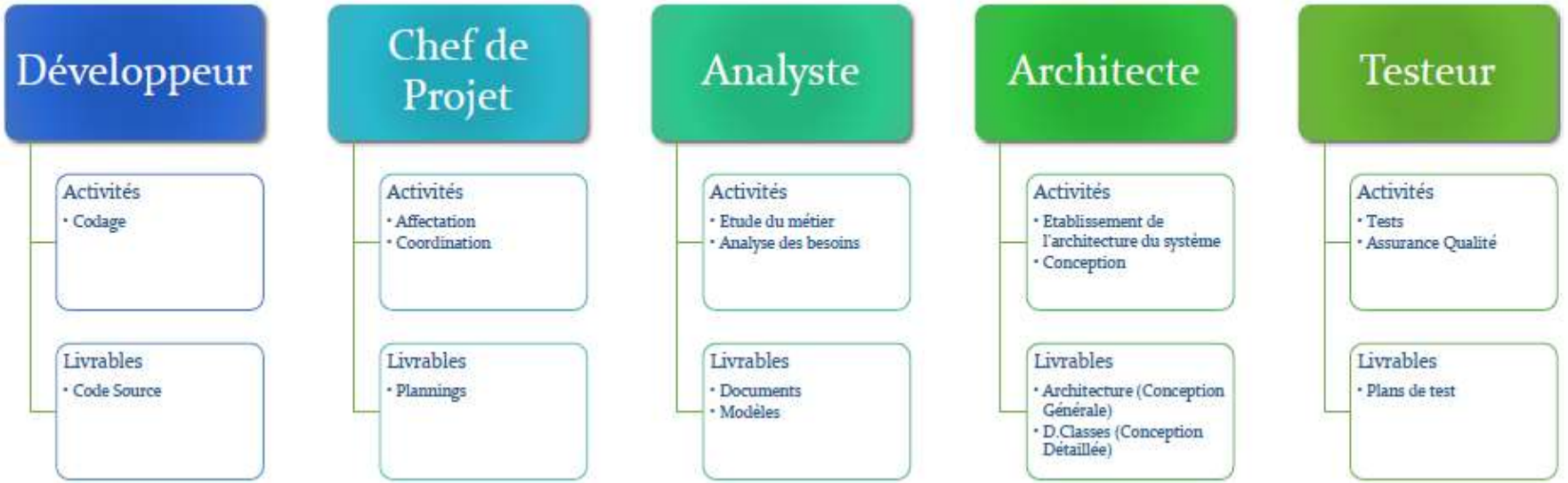
Niveau de maturité	Délai en mois	Défauts trouvés	Défauts livrés	Coût total
1	29,8	1348	61	$5 440 000
2	18,5	328	12	$1 311 000
3	15,2	182	7	$728 000
4	12,5	97	5	$392 000
5	9,0	37	1	$146 000

Impact estimé sur un projet de développement de logiciel de 200 000 lignes de code

Outils et Métiers du Développement Logiciel

Outils et Métiers

- Principaux Métiers de Développement



Outils et Métiers

- Métiers et Activités

	Expression de besoins	Analyse	Conception	Implémentation	Tests
Développeur					
Analyste					
Architecte					
Testeur					
Chef de Projet					

Outils et Métiers

- **Outils CASE** (**Computer-aided software engineering**)
  - CASE est un nom donné aux ***logiciels*** utilisés dans les différentes activités de GL (besoins, conception,…)
  - Exemples : compilateurs, éditeurs, débogueurs, …etc.
  - Le but des outils CASE est d'automatiser les tâches et/ou gérer le projet de développement

Outils et Métiers

- **Classification des CASE:** Les outils CASE peuvent être classés :
  - D'un point de **vue fonctionnel** : selon la fonction de l'outil.
  - D'un point de **vue activité** : selon les activités dans lesquelles intervient l'outil

Outils et Métiers

- Classification fonctionnelle:

Outil	Exemples	Références	Métier
Outils de planification	Outils PERT, outils d'estimation, tableurs	Microsoft Project, Excel, GanttProject, DotProject	CDP
Editeurs	Editeurs de texte, éditeurs d'image, éditeurs de diagrammes	vi, bloc notes, GIMP, Photoshop, Visio	Tous
Gestion de configuration	Gestion de versions, gestion de builds	SVN, CVS, Team Foundation Server, ClearCase	CDP, Développeur, Architecte
Outils de support de procédé	Générateurs de code, outils d'assistance, IDE	Team Foundation Server, Accurev, Enterprise Architect	Tous
Outils de traitement de langage	Compilateurs, interpréteurs, débogueurs		Développeur, Architecte

Outils et Métiers

- Classification fonctionnelle – Suite:

Outil	Exemples	Références	Métier
Outils de test	Environnements de tests, outils de tests unitaires	Junit, Nunit, TestWorks, Bugzilla	Testeur, Développeur
Outils de documentation	Documents de projet, documents de code	Word, Open Office, Sandcastle, Doxygen, javadoc	Développeur

Les approches de modélisation de logiciels

Les paradigmes de modélisation de logiciels

- Deux célèbres paradigmes:
  - La modélisation orientée fonction
  - La modélisation orientée objet

La modélisation orientée fonction

- Le logiciel est conçu à partir d'un point de vue **fonctionnel**
- Approche **Top-down** (Décompostion)
- **DFD** est utilisé

Diagramme de flot de données (DFD):

- C'est le dossier de spécification le plus important.
- Représentation semi formelle qui décrit la circulation des données d'entrées à travers un ensemble de transformations fonctionnelles (des **taches** ou des **composants logiciel**) pour donner des résultats de sortie.
- Ils constituent un moyen intuitif et utile de décrire un système et sont compréhensibles sans formation particulière.
- Un DFD peut être raffiné à plusieurs niveaux (le premiers s'appelle **diagramme de contexte**).

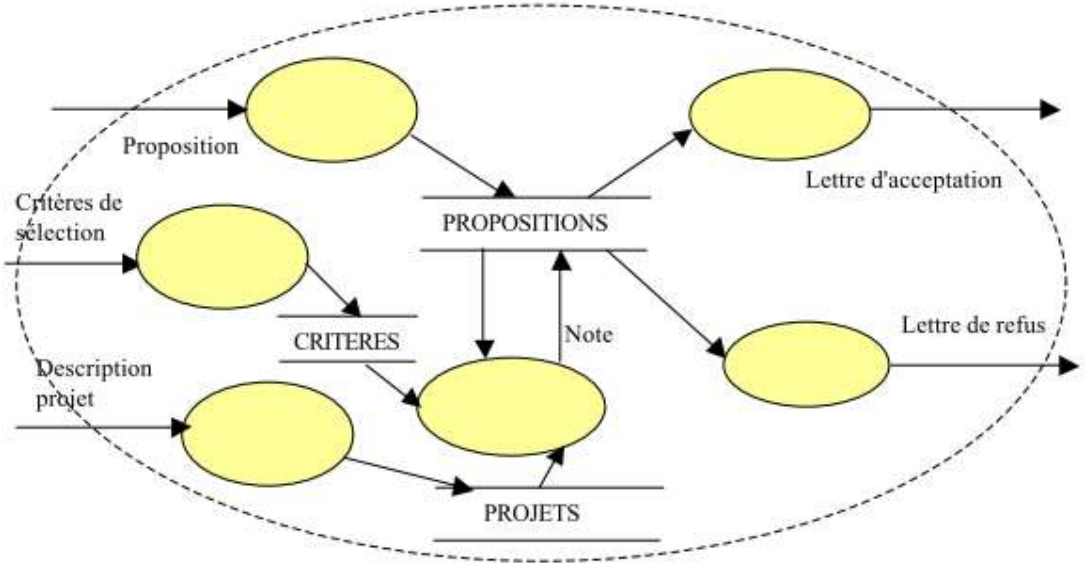


Diagramme de flot de données (DFD):

- 2 notations connues:

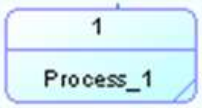
Concept	Outil	Gane & Sarson	Yourdon
Process (processus)			
Flow (flux)			
Data store (magasin de données)			
External entity (entité externe)			

Diagramme de flot de données (DFD):

1/Les processus (fonctions):

- Transforme le flux d'entrée en flux de sortie.
- Appeler aussi transformateur.
- Chacun possède un nom et numéro unique qui doit être un verbe significatif (éviter les noms vagues: comme traitement données…).

Exemple:

- Exécuter un calcul.
- Prendre des décisions (i.e., Déterminer la disponibilité d'un produit).
- Trier, filtrer et autre opérations de groupe sur les données.
- …etc.

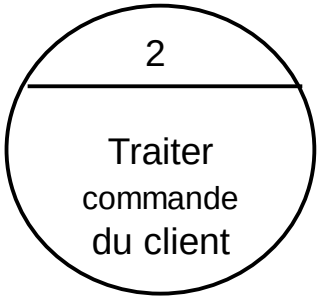


Diagramme de flot de données (DFD):

2/Les flots de données:

- Représentation des données qui circulent dans le système.
- La donnée peut être physique (produit) ou abstraite (une décision).
- Chaque flux de donnée dispose d'un nom ex: commande du client. Le nom doit être significatif et non ambigu.
- Un flot ne peut pas partir d'un élément et y arriver.
- 2 flèches sortantes signifie: simultanées ou exclusives.

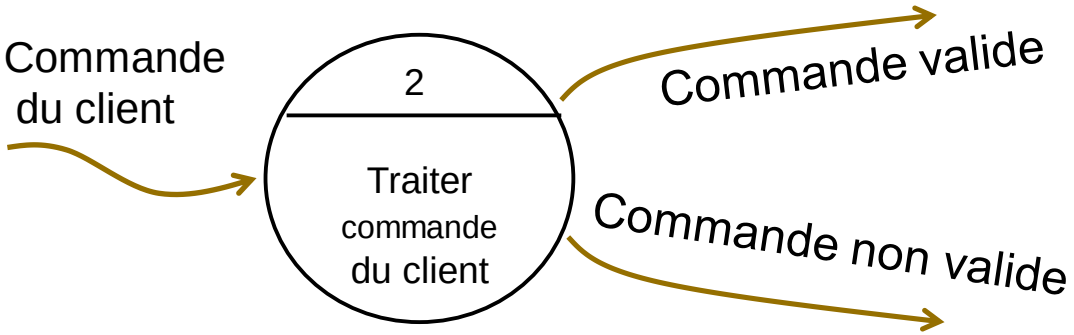


Diagramme de flot de données (DFD):

3/ Eléments externes:

- les entités situées en dehors des frontières de l'application étudiée (fournissent ou récupèrent des données, déclenche une action…).
- Disposent d'un nom: directeur, client, autre système ou département….
- Ils peuvent être des sources ou des destinations (ou les deux).

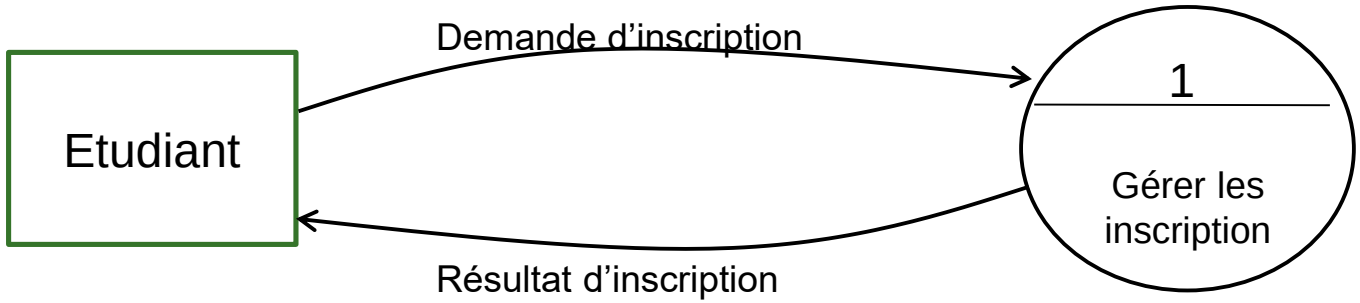


Diagramme de flot de données (DFD):

4/Le stockage de données (data store):

- Ensemble de données (numériques ou physiques) en repos (lieu de stockage de données).
- Des données nécessaires pour le fonctionnement du système.
- Leur contenu n'est accessible que par l'intermédiaire d'un processus (fichiers, base de données, liste,…).

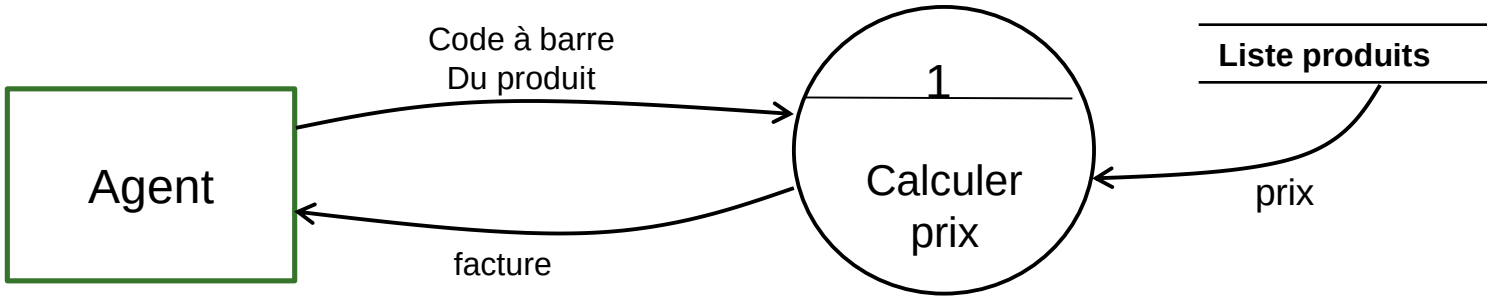


Diagramme de flot de données (DFD):

Flots de données correct:

- Entre processus.
- Depuis un data store vers un processus (utilisation données).
- Depuis un processus vers un data store (mise à jour).
- Depuis une source (élément externe) vers un processus.
- Depuis un processus vers une destination.

Diagramme de flot de données (DFD):

Exemple d'erreurs:

- Flôts Entre agents externes.
- Entre stockage de données.
- Entre un agent externe et un data store.
- Un processus (ou data store) qui a uniquement des entrées (black hole).
- Un processus qui a uniquement des sorties (miraculous process).
- Flux de données entre processus avec 2 directions.
- Les entrées et les sortites sont identiques.
- Processus nomé avec une phrase nominale.
- Flot de données nomé avec un verbe.
- Un DFD ne devrait pas avoir plus de 7 processus (surchargé).

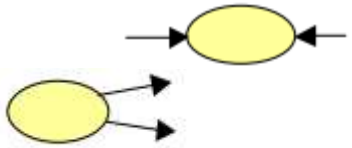
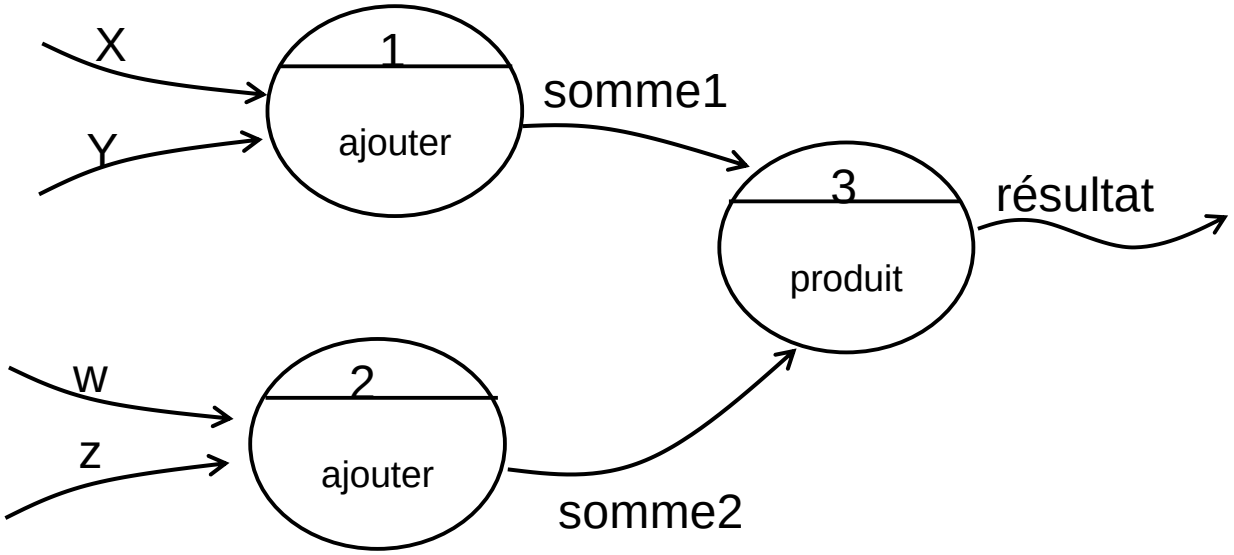


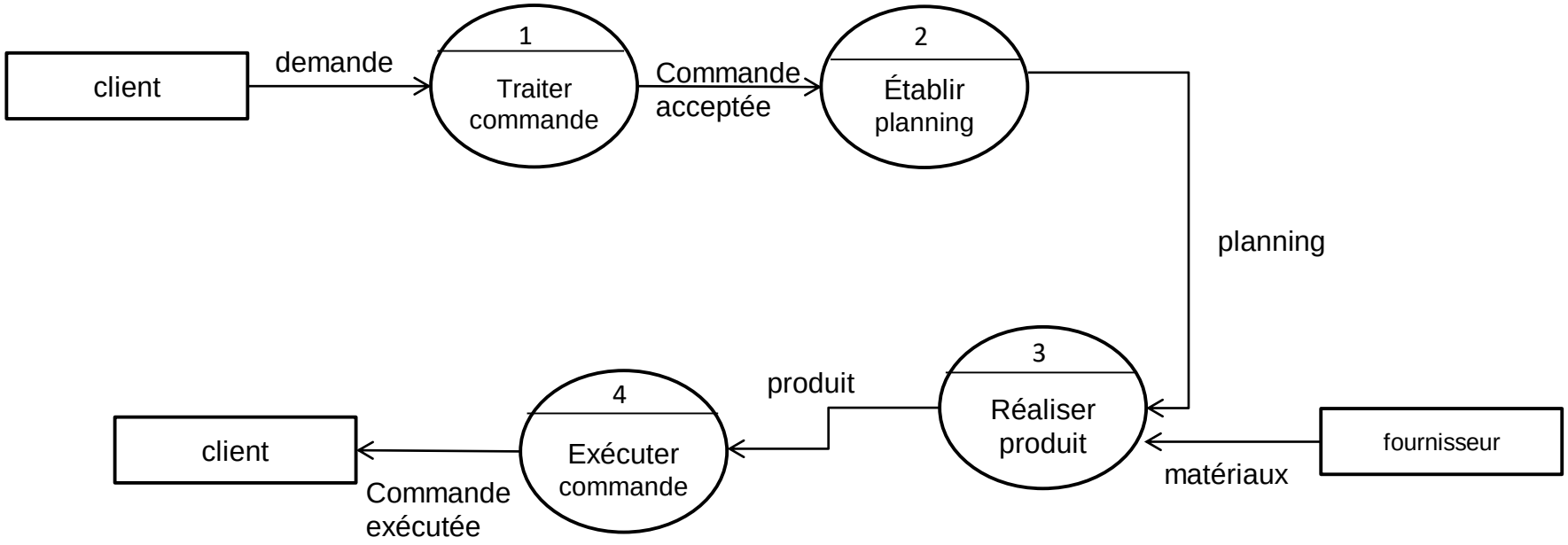
Diagramme de flot de données (DFD):

- Exemple1: représenter la formule (x+y)*(w+z):



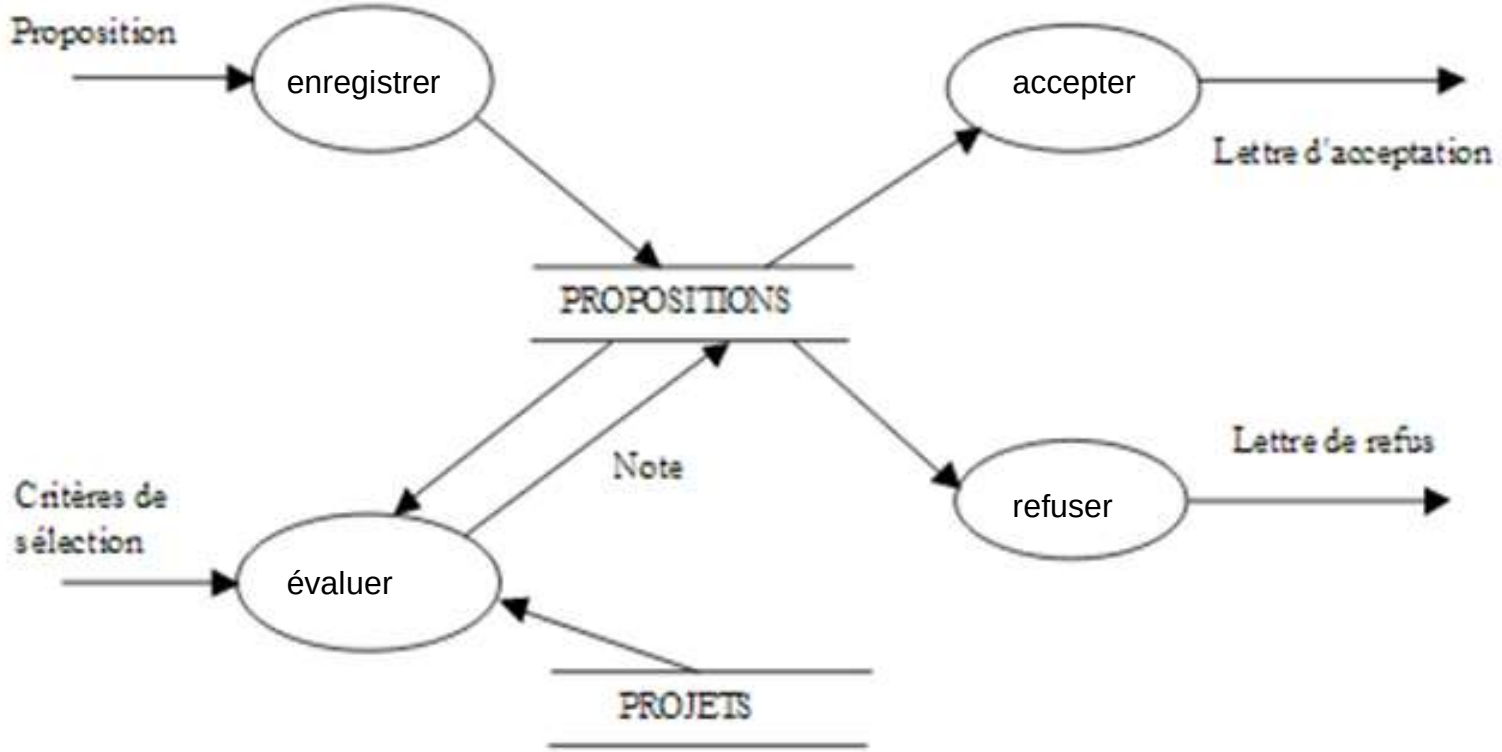
Exemple :

La figure donne un exemple de DFD concernant le traitement de la commande d'un client pour la fabrication d'un produit donné.



Exemple 2:

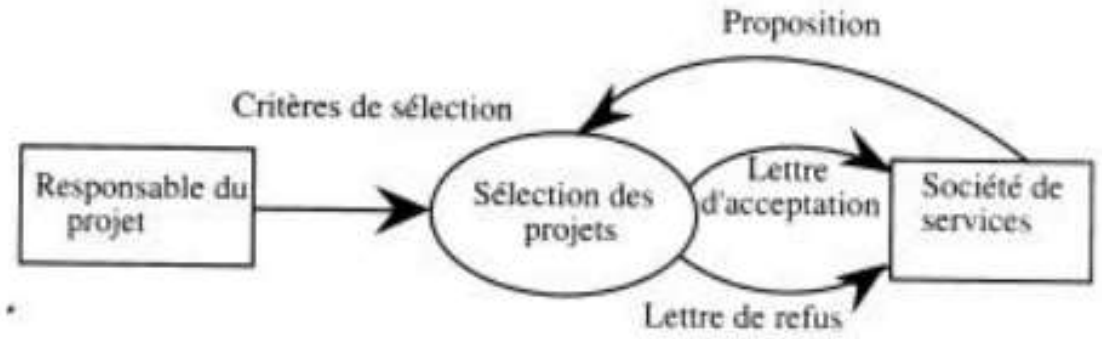
Le DFD décrivant le système d'évaluation des offres proposées pour un projet donné.



Exemple 2:

Le DFD décrivant le système d'évaluation des offres proposées pour un projet donné.

Diagramme de contexte:



La modélisation orientée objet

- Le logiciel est vu comme une collection d'**objets** (c.-à-d. des entités)
- Approche **Bottom-up** (composition)
- **UML** est utilisé

FIN