

Chapitre 1 : Introduction au Génie Logiciel Partie 2

Rappel

La séance passée...



Rappel

- Logiciel

- ✓ Des logiciels partout!!
- ✓ Notre vie dépend très fortement de la qualité des logiciels
- ✓ Impacts positifs des logiciels:
- ✓ Impacts négatifs des logiciels:



- Développement

- ✓ 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
- ✓ N'est pas une opération facile
- ✓ Il comprend un ensemble d'activités
- ✓ Cycle de vie

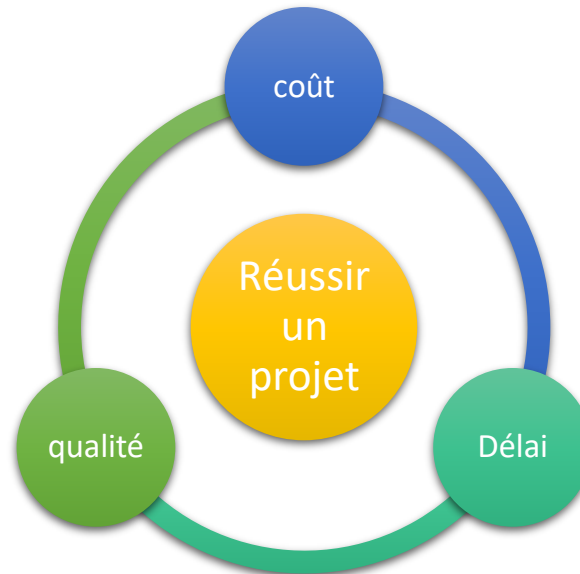
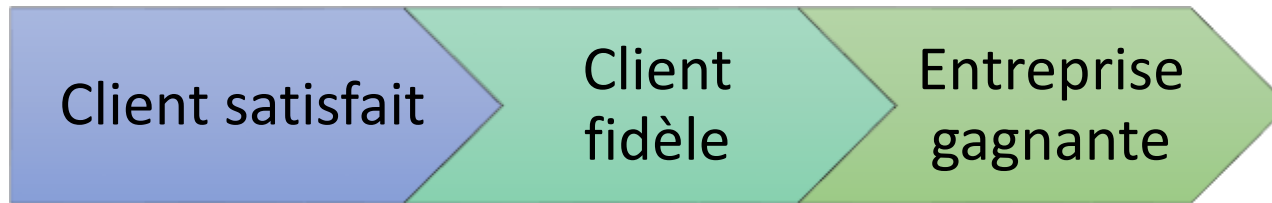
Rappel

- Projet de développement de logiciels
 - ✓ Equipe
 - ✓ Procédé
 - ✓ Outils
 - ✓ Communication
- Qu'est ce qu'un bon logiciel ?
 - ✓ client
 - ✓ fournisseur



Rappel

Penser **qualité**:



Plan

- La crise du Logiciel
- Le Génie Logiciel
- Critères de qualité d'un logiciel
- Cycle de vie de logiciel
- Modèles de cycle de vie de logiciel

La crise du Logiciel



La crise du logiciel

- Apparue à la fin des années 1960
- À cette époque, le développement logiciel relevait encore largement de l'**artisanat**
 - ✓ Reposant sur des pratiques **empiriques** et des méthodes peu **standardisées**, inadaptées à la **complexité croissante** des grands systèmes informatiques

De nombreuses difficultés :

- **Non-conformité aux besoins** : logiciels ne répondant pas aux attentes des utilisateurs
- **Difficultés de communication** : équipes de développement **débordées** et **coordination** insuffisante
- **Dépassements de coûts et de délais** : projets livrés en retard ou abandonnés
- **Maintenance coûteuse** : logiciels difficiles à comprendre, à corriger et à faire évoluer
- **Performances insuffisantes** : temps de réponse trop longs
- **Faible évolutivité** : difficulté à adapter les logiciels aux nouveaux besoins
- **Ergonomie insuffisante** : interfaces peu conviviales

Exemple : le projet *TAURUS* de la Bourse de Londres, abandonné en 1993 après plusieurs années de développement et des difficultés importantes

Le Génie Logiciel



- Apparition du *Software Engineering*
- 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.
- **Contexte** : crise du logiciel et difficultés croissantes liées au **développement** de systèmes informatiques **complexes**
- **Objectif** : promouvoir des méthodes **rigoureuses** et **systématiques** pour le développement des logiciels
- **Idée clé** : passer d'un développement logiciel **artisanal** à une véritable démarche **d'ingénierie**

Le Génie Logiciel

- ***Software engineering***
- Un domaine de l'ingénierie qui permet la **conception**, la **réalisation** et la **maintenance** des systèmes logiciels de **qualité**
- 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**

Principes du 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. 8- Mesurabilité
9. Facilité de maintenance

Fiabilité

- **la validité (conformité ou utilité)**: logiciel conforme à ses spécifications, les résultats sont ceux attendus.
- **La robustesse**: le logiciel fonctionne raisonnablement en toutes circonstances, même dans des situations imprévues, des erreurs ou des données inattendues
- 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

Utilisabilité

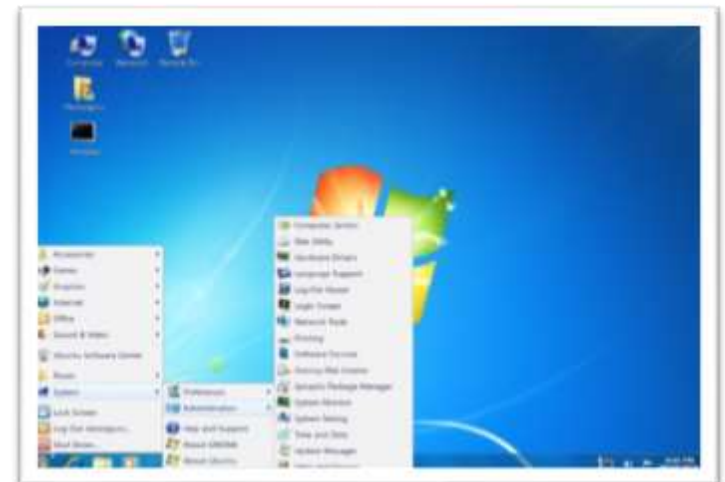
- Aptitude à communiquer de façon **efficace** avec l'utilisateur
- L'utilisateur doit:
 - ✓ **Connaître** et **comprendre** toutes les fonctionnalités offertes par le logiciel (facilité d'apprentissage).
 - ✓ Apprendre comment les utiliser pour des fins donnés (**ergonomie** et facilité d'utilisation).

```
ETD1: ConfigureSetup called
ETD1: AddRemoveSetup f6dd=0
ETD1: .....FirstBootCall.....
ETD1: ExecuteQueuedAPIs
ETD1: ClassInstall (0x6 on 0x2c96:0x1230) on at
ETD1: SetupFlags=506 BootCount=2 NetSetupFlags=1 (RETAIL)
ETD1: dif_FirstTimeSetup
ETD1: ClassInstall(0x6) end
ETD1: ClassInstall (0xc on 0x2c96:0x1230) on at
ETD1: SetupFlags=506 BootCount=2 NetSetupFlags=1 (RETAIL)
ETD1: ClassInstall(0xc) end

C:\>dir /w /p

Le volume dans le lecteur C n'a pas de nom
Le numéro de série du volume est 257C-1B10
Répertoire de C:\

DIRNAME.COM      AUTOEXEC.BAT      FRUNLOG.TXT      I\WINDOWS1      NETLOG.TXT
DIRFIG.SYS      (FROGGER"1)      (FROGGER"1)
5 fichier(s)      106 224 octets
3 répertoire(s)  1 798 963 266 octets libres
```



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

Extensibilité- Portabilité- Performance

- **Extensibilité:** Aptitude d'un logiciel à être **enrichi** et **mis à jour** sans rejeter le logiciel ou recommencer à 0.
- **Portabilité:** Un même logiciel doit pouvoir fonctionner sur **plusieurs machines** (plateformes), c'est-à-dire Rendre le logiciel indépendant de son environnement d'exécution.
- **Performance:** Les logiciels doivent satisfaire aux contraintes de **temps d'exécution** et d'occupation d'espace mémoire. (veillez à la **complexité** des logiciels)

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 :

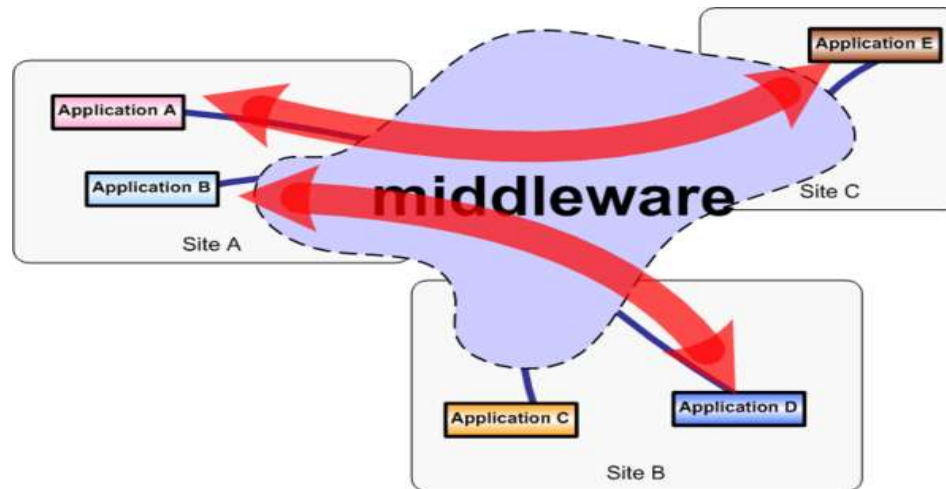
- 80 % du code est du « tout venant » qu'on retrouve à peu près partout.
- 20 % du code est **spécifique**.



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 bien définie.



Mesurabilité

Aptitude d'un logiciel à être évalué:

- cout
- délai
- complexité
-



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'accroître/améliorer les possibilités du logiciel.
- Exemple
 - ✓ Les services offerts,
 - ✓ Interface utilisateur,
 - ✓ les performances ... etc.
- Donne lieu à de nouvelles versions.

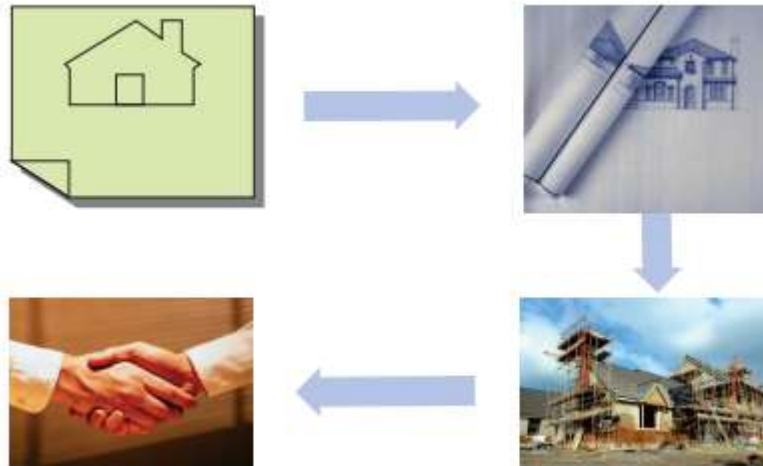


Facilité de la maintenance

- Objectif
 - ✓ Réduire la quantité de maintenance corrective (zéro défaut).
 - ✓ Rendre moins coûteuses les autres maintenances.
- Enjeux
 - ✓ Les coûts de maintenance se jouent très tôt dans le processus de développement.
 - ✓ 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



Cycle de vie de logiciel



Cycle de vie de logiciel

- La **qualité** du produit dépend en grande partie de la qualité du **processus** de développement.
- 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 (2)

- 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.
- Plus une erreur est détectée tardivement dans le processus de développement, plus son coût de correction est élevé.
- 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.

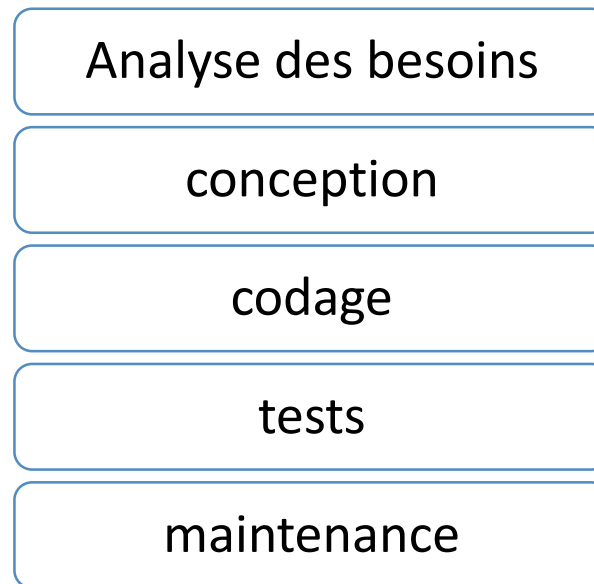
Activité, acteur et livrable

- 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.
- Exemple : document, **planning**, code **source** sont tous des livrables



Cycle de vie de logiciel - activités communes

- Tous les projets de développement ont des activités communes:



Requirements Analysis → Design → Implementation → Testing → Maintenance

Analyse des besoins

conception

codage

tests

maintenance

Analyse des besoins

- Cahier de charge
- L'expression, le recueil et la formalisation des besoins
 - ✓ Nouveau logiciel
 - ✓ **Évolution** d'un logiciel existant
 - ✓ Mise à jour ou 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 coûts, planification, **répartitions des tâches**....)



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 besoins) 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:
 - Définir précisément (des diagrammes et des algorithmes) de chaque **sous-ensemble** du logiciel.

Conception-Recommandation

- Maximiser la **cohésion** au sein des composants et minimiser le **couplage** entre eux
- Forte cohésion
 - ✓ Les éléments d'un même composant sont **fortement liés** et contribuent à une même **responsabilité**.
- Faible couplage
 - ✓ Les composants **dépendent** le moins possible les uns des autres.
- Forte cohésion + faible couplage = meilleure conception logicielle
- C'est une **bonne pratique** qui rend le logiciel
 - ✓ Plus modulaire,
 - ✓ Plus facile à comprendre,
 - ✓ Plus réutilisable,
 - ✓ 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.
 - ✓ Echantent 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'interface** des différents éléments (modules) du logiciel.
- **Tests d'acceptation:** vérification de la conformité du logiciel aux spécifications initiales (Validation par l'acheteur).



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'interface** des différents éléments (modules) du logiciel.
- **Tests d'acceptation:** vérification de la conformité du logiciel aux spécifications initiales (Validation par l'acheteur).



TDD

- TDD — Test-Driven Development
- Développement piloté par les tests.
- Le principe est généralement présenté avec le cycle Red → Green → Refactor:
 - ✓ Red : écrire un test qui échoue.
 - ✓ Green : écrire le minimum de code nécessaire pour faire réussir le test.
 - ✓ Refactor : améliorer le code sans modifier son comportement.

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

- But:
 - ✓ **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.

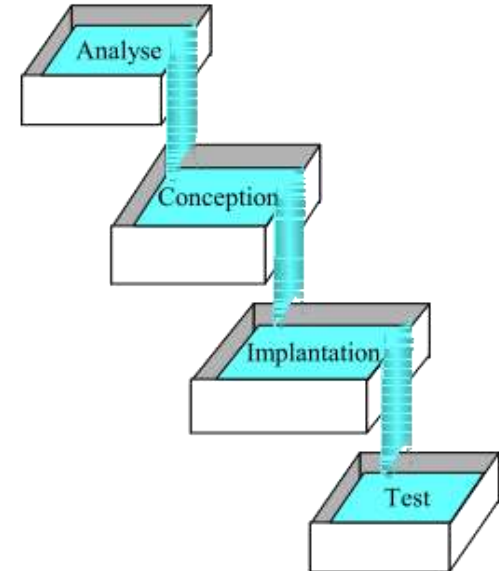
Modèle de cycle de vie/Procédé logiciel

- 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.

Modèles de Cycle de vie de logiciel

Modèle en cascade (chute d'eau)

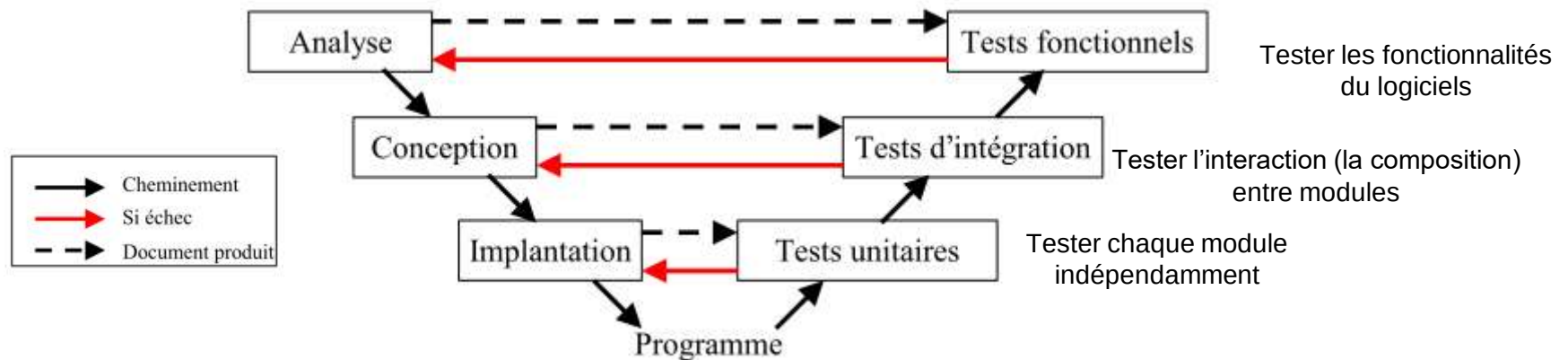
- *Waterfall Model*
- L'un des premiers modèles proposés (1970)
- Inspiré de l'industrie des bâtiments (pas de retour en arrière).
- Chaque phase se termine à une **date précise** par la production de certains documents ou logiciels.
- Une phase ne peut démarrer que si la précédente est finie. Les résultats 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 maîtrisé (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

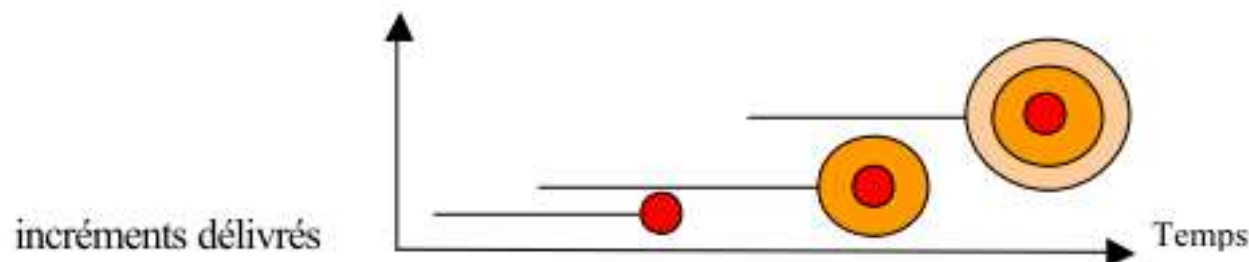
- Le plus **connu** et certainement le plus **utilisé**.
- Le développement des **tests** et du logiciel sont effectués de manière **synchrone**
 - ✓ Le test du produit se fait en **parallèle** par rapport aux autres activités.
- Chaque livrable (de chaque étape) **doit être testable**
 - ✓ (toute description 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**.

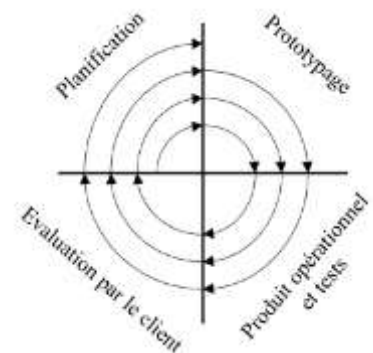
Modèle incrémental

- 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.



Modèle en spirale (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 en raison du développement cyclique, qui augmente le risque de devoir modifier ou refaire certaines réalisations précédentes.
- Nécessite une très grande expérience.
- Utilisation:
 - ✓ Des grands projets complexes (aéronautique, médical, défense, etc.)
 - ✓ Les projets à fort risque ou incertitude technique.



Questions

