Chapitre 1
Introduction au génie logiciel

- Problèmes de l'industrie logiciel
- La qualité du logiciel
- Fondements du génie logiciel
- Modélisation en génie logiciel

1. Problèmes du génie logiciel

Tout système informatique présente deux aspects :

- Un **aspect matériel** qui représente 20% du coût global.
- Un **aspect logiciel** représentant 80% du coût global.

Le matériel est de plus en plus fiable, performant et standard. Le logiciel a atteint un bon niveau de maturité. Cependant, il se heurte aux problèmes liés à la maîtrise de la complexité qui sont de deux types :

- **Complexité de taille** : liée à la manipulation des logiciels de grandes dimensions.
- **Complexité de structure** : liée à la représentation et la à manipulation des connaissances.

La complexité de taille pose des problèmes relatives à :

- La décomposition du logiciel lors de la phase de conception.
- La coordination lors de la phase de réalisation du logiciel.
- La maintenance logicielle qui représente plus de 70% du coût total d'un logiciel.
- La réutilisation du code : moins de 15% des codes produits chaque année sont originels.

La complexité de structure est liée essentiellement à :

- La modélisation de l'information complexe.
- L'utilisation des outils de manipulation de l'information.

Les logiciels actuels doivent manipuler des atomes qui ne sont pas simplement des nombres ou des symboles, mais des structures très complexes : les **objets**.

Ces problèmes sont les causes de ce qu'on appelle « **la crise du logiciel** ». D'après une étude menée par le ***Standish Group*** en 1995 sur 8380 projets de l'industrie du logiciel :

- **Succès** : **16%**.
- **Problématique** : **53%** (budget ou délais non respectés, défaut de fonctionnement).
- **Échecs** : **31%** (projets abandonnés).

Définition. Le **génie logiciel** (*software Engineering*) est une discipline de l'informatique qui étudie l'application des techniques de l'ingénieur pour produire des logiciels de qualité.

Les objectifs du génie logiciel :

- Comment faire des logiciels de qualité ?
- Définir les critères de qualité du logiciel.
- Élaborer des méthodes et des outils pour réaliser ces critères de qualité du logiciel.

2. La qualité du logiciel

Les facteurs de qualité d'un logiciel sont de deux types :

- **Facteurs internes** : perceptibles aux informaticiens professionnels qui ont accès au code source du logiciel.
- **Facteurs externes** : dont la présence dans un produit logiciel peut être détectée par ses utilisateurs (le client).

Quelques facteurs internes :

- **Modularité** : logiciel conçu et réalisé par modules.
- **Lisibilité** : le code source est facile à lire et bien commenté.

Les facteurs internes ne sont pas une fin en soi, mais un moyen d'atteindre les facteurs externes, qui, en fin de compte, seuls auront de l'importance.

Quelques facteurs externes du logiciel :

- **Correction** : capacité de mener à bien sa tâche, telle qu'elle a été définie par sa spécification.
- **Robustesse** : capacité à réagir de manière adéquate à la présence de conditions anormales.
- **Extensibilité** : facilité d'adaptation aux changements de spécifications.
- **Réutilisabilité** : capacité des éléments logiciels à servir à la construction de plusieurs applications différentes.

- **Compatibilité** : facilité avec laquelle des éléments logiciels peuvent être combinés à d'autres.
- **Efficacité** : capacité à utiliser le minimum de ressources matérielles, que ce soit le temps machine, l'espace occupé en mémoire interne ou externe, ou la bande passante des moyens de communication.
- **Portabilité** : facilité avec laquelle des produits logiciels peuvent être transférés d'un environnement logiciel ou matériel à un autre.
- **Facilité d'utilisation** : facilité avec laquelle des personnes présentant des formations et des compétences différentes peuvent apprendre à utiliser le logiciel et à s'en servir pour résoudre des problèmes. Elle recouvre également la facilité d'installation et de contrôle.
- **Fonctionnalité** : étendue des possibilités offertes par un système.
- **Ponctualité** : capacité d'un logiciel à être livré au moment désiré par ses utilisateurs, ou avant.
- **Intégrité** : capacité à protéger les divers composants contre les accès et modifications non autorisés.
- **Économie** : capacité d'un logiciel à être terminé dans les limites de son budget, ou en deçà.
- **Réparabilité** : capacité à faciliter la réparation des défauts.
- **Vérifiabilité** : facilité à préparer les procédures de recette, en particulier les jeux de tests, ainsi que les procédures permettent de détecter les erreurs.
- **Bonne documentation**.

Les facteurs de correction, de robustesse, d'extensibilité et de réutilisabilité sont les préoccupations essentielles du génie logiciel en termes de qualité.

La **maintenance** du logiciel est un autre problème très difficile. Il y a deux types de maintenance :

- **Maintenance corrective** : elle vise à corriger la version courante du système.
- **Maintenance évolutive** : elle vise à améliorer la version courante du système.

3. Les fondements du génie logiciel

La naissance du génie logiciel était vers 1968, date du première conférence sur le génie logiciel. L'idée clé du génie logiciel est que « la production du logiciel doit être organisée ». Cette organisation passe par :

- Le contrôle des coûts des projets.
- L'organisation de l'équipe de production.
- Le contrôle des délais de réalisation.

- Le contrôle de la qualité.

La vie d'un logiciel est composée de différentes étapes. La succession de ces étapes forme le **cycle de vie** (***software lifecycle***) du logiciel. Le contrôle et la bonne gestion des différentes étapes du cycle de vie sont des garanties pour obtenir un logiciel de qualité.

3.1 Composants du cycle de vie du logiciel

Le cycle de vie du logiciel est constitue des étapes suivantes :

- Étude de faisabilité.
- Spécification des besoins.
- Organisation du projet.
- Conception.
- Implémentation.
- Tests.
- Livraison.
- Maintenance.

Étude de la faisabilité :

⇨**Objectifs** :

- Étudier la faisabilité économique du projet.
- Étudier les risques de développement du projet.
- Étude du marché : déterminer s'il existe un marché potentiel pour le futur produit.

⇨**Moyens** :

- Modèles probabilistes.
- Modèles statistiques.
- Modèles économiques.

Spécifications des besoins :

⇨**Objectif** : déterminer l'ensemble des fonctionnalités attendues du logiciel. C'est l'étude du « **quoi** ? ».

⇨**Moyens** :

- Collecte des exigences des clients pour le logiciel.

- Analyse du domaine de l'application : identifier les principaux travaux et les modéliser.

Organisation du projet :

⇨**Objectifs** :

- Estimer le coût global de production.
- Estimer les délais de réalisation du projet.
- Déterminer les ressources nécessaires au projet (humaines, matérielles, techniques).
- Établir un planning des travaux.
- Élaborer le processus qui pilote le contrôle d'assurance qualité.
- Ordonnancer les travaux du projet.

Conception du logiciel :

⇨**Objectif** : décrire la manière avec laquelle le logiciel sera fait. C'est l'étude du « **comment** ? ».

Il y a étapes de conception :

La conception générale :

- Conception de l'architecture du logiciel : structure globale du système (éléments, structures et contraintes).
- Conception des interfaces : comment les différents éléments interagissent entre eux ?

La conception détaillée :

- Définir en détail les composants et leurs entités.
- Établir les algorithmes.
- Définir les choix de réalisation concernant les langages de programmation, les bases de données, la plate-forme du réseau, ...etc.

Implémentation :

⇨**Objectif** : écrire le code source de l'application.

Remarques :

- La charpente du code est générée automatiquement depuis le modèle de conception.
- Le détail des opérations est ensuite réalisé manuellement.
- L'intégration du nouveau code avec le code existant est réalisée de façon graduelle.

Tests :

⇨**Objectif** : essayer le logiciel pour s'assurer de sa qualité.
Il y a plusieurs types de tests :

- **Tests unitaires** : tester indépendamment les différentes parties du logiciel.
- **Tests d'intégration** : tester globalement le logiciel pendant la phase d'intégration de ses composants.
- **Tests de validation** : tester pour acceptation par l'acheteur (le client).

Livraison :

Fournir au client le logiciel qui a été demandé. Cela inclus entre autre :

- **L'installation** : installer le logiciel sur les sites des clients.
- **La documentation** : fournir les documents d'installation, d'administration et d'utilisation du logiciel.
- **La formation et assistance** : enseigner le logiciel à ces utilisateurs et répondre à leurs questions.

Maintenance :

⇨**Objectif** :

- Effectuer les mises à jour du logiciel.
- Améliorer les performances du logiciel.
- Prévoir les changements.

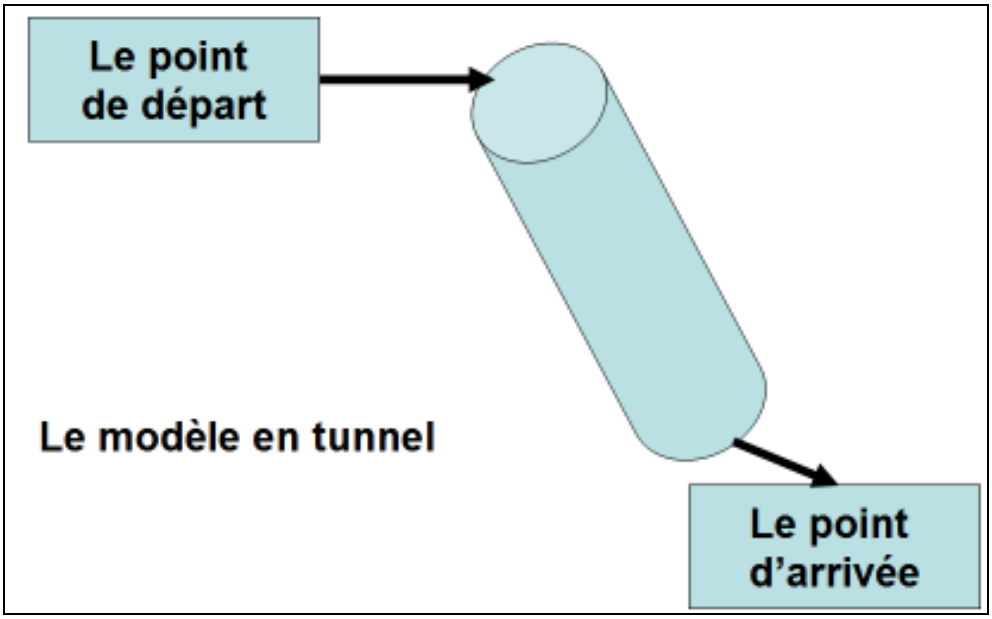
3.2 Les différents documents à produire

- **Cahier de charge** : description initiale des fonctionnalités généralement écrite par le client.
- **Cahier des spécifications** : décrit en détail les contraintes que doit respecter le logiciel.
- **Calendrier du projet** : contient une description des tâches de réalisation du projet ainsi que leur ordonnancement.
- **Plan de tests** : fournit une description des procédures de tests à appliquer.
- **Plan d'assurance qualité** : décrit les actions à effectuer pour assurer l'obtention d'un logiciel de qualité.
- **Manuel d'installation** : décrit comment installer le logiciel.
- **Manuel d'utilisation** : décrit comment et dans quelles conditions utiliser le logiciel.
- **Code source** : le code du logiciel en termes de programmes.

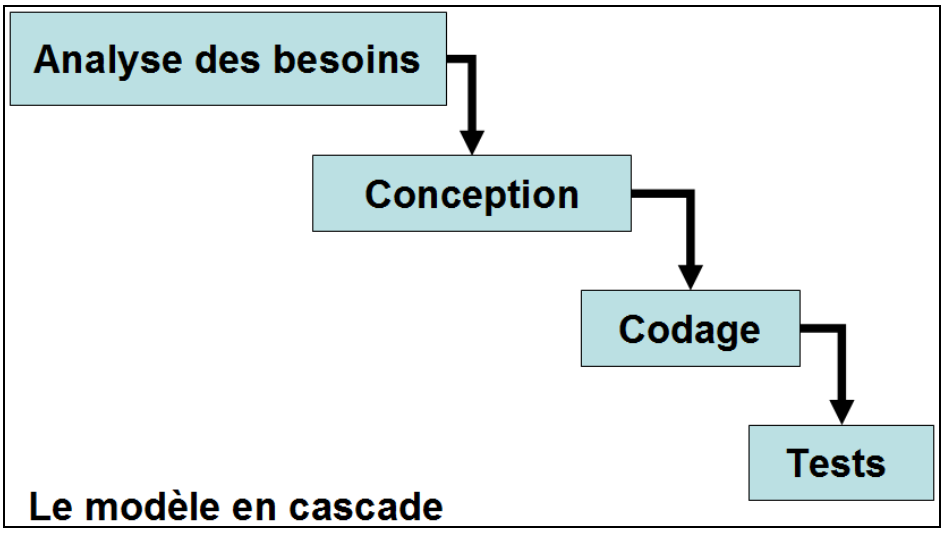
- **Rapport des tests** : décrit les tests effectués et leurs résultats (réactions du logiciel).
- **Rapport des défauts** : décrit les anomalies du logiciel (erreurs, bogues).

3.3 Les modèles de cycle de vie du logiciel

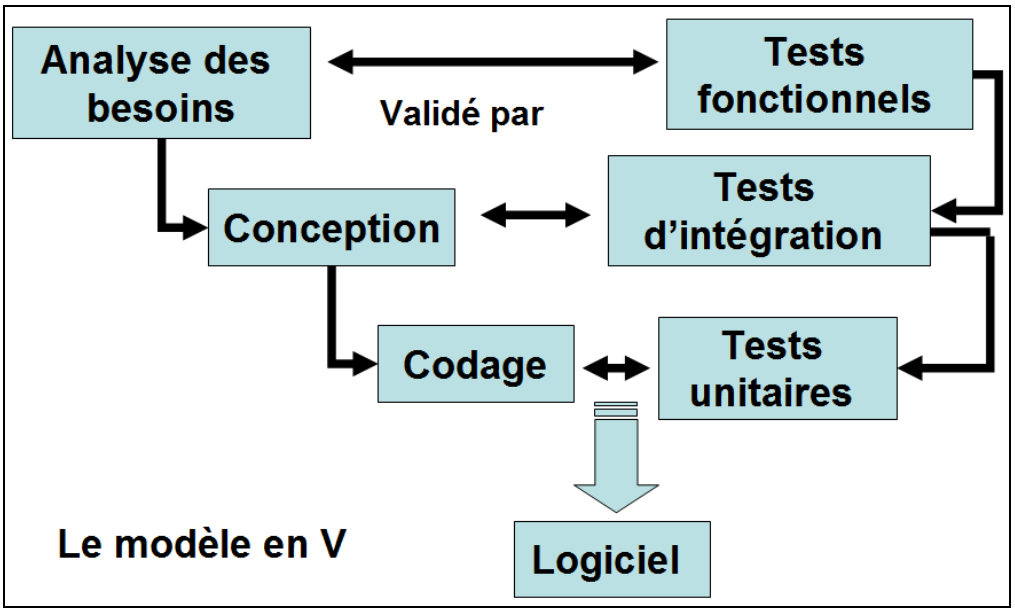
- **Le modèle en tunnel** : modèle de développement obscur (impossible de savoir ce qui se passe).



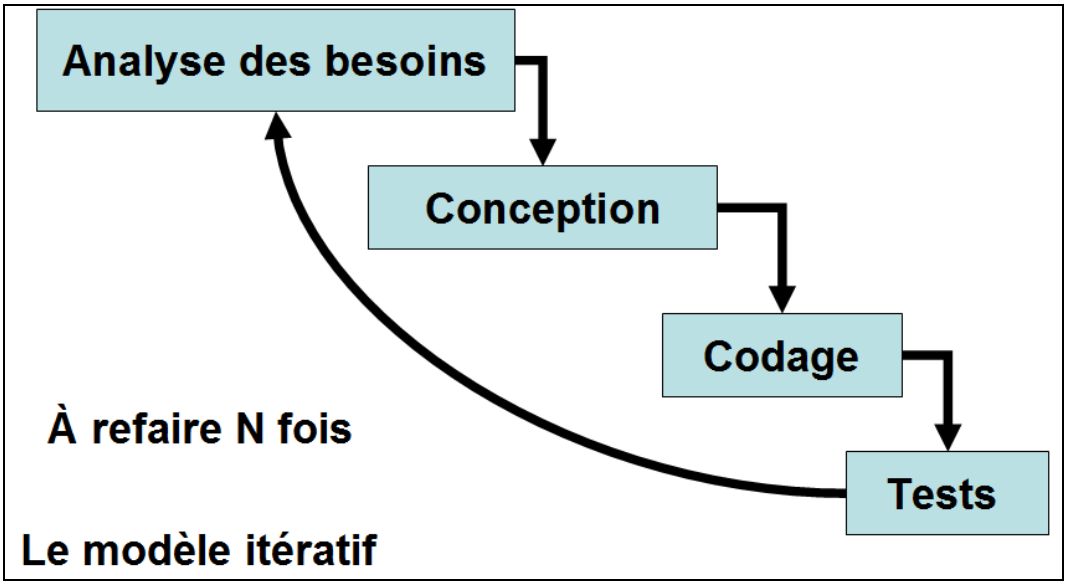
- **Le modèle en cascade** : développement de logiciel comme une suite de phases qui s'enchaînent dans un déroulement continu.



- **Le modèle en V** : développement des tests s'effectuant en parallèle avec le développement du logiciel.



- Le **cycle de vie itératif** : basé sur l'évolution de prototypes (versions d'essai) exécutables, mesurables et donc sur l'évaluation d'éléments concrets.



<u>Avantages du cycle de vie itératif</u> :

- Facilite la prise en compte des problèmes.
- Les changements sont incorporés dans les itérations futures.
- Les progrès se mesurent par programmes.
- L'utilisateur est placé devant des situations d'utilisation concrètes qui lui permettent de mieux structurer ses besoins et de les communiquer à l'équipe du projet.
- L'utilisateur devient partenaire du projet : il prend sa part de responsabilité et de fait accepte le nouveau système plus facilement.

- L'équipe de développement est plus motivée du fait de la proximité de la cible.
- L'intégration des différents composants du logiciel est réalisée de manière progressive.

Inconvénients du cycle de vie itératif :

- Demande plus d'attention et d'implication de l'ensemble des acteurs du projet.
- La démarche doit être présente et comprise par tous : les clients, les utilisateurs, les concepteurs, les programmeurs, le chef de projet, les testeurs et les documentalistes.

4. La modélisation en génie logiciel

Définition. En informatique, la modélisation d'un système est le processus par lequel on aboutit à un modèle de ce système. Elle est toujours orientée vers un objectif précis et prédéterminé.

Minsky définit un modèle comme suit : « Pour un observateur A, M un modèle d'un système S, si A peut, à partir de M, apprendre quelque chose d'utile sur le fonctionnement de S ».

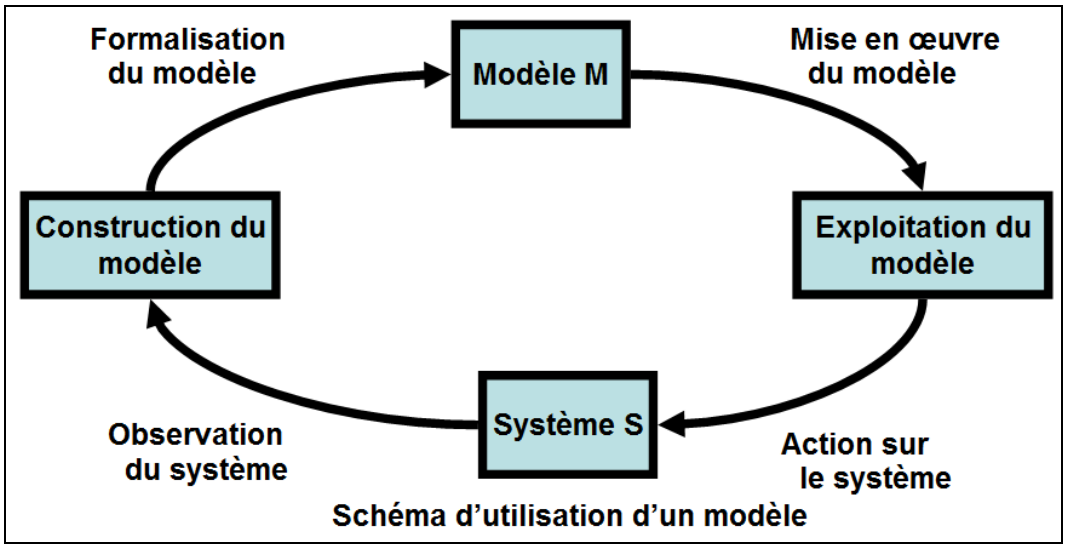


Schéma d'utilisation d'un modèle

Popper suggère trois concepts relatifs à la notion de modèle :

- Un modèle doit avoir un caractère de ressemblance avec le système réel.
- Un modèle doit constituer une simplification du système réel.
- Un modèle est une idéalisation du système réel.

En pratique, la construction d'un modèle doit s'effectue à partir d'observations du système réel à modéliser et doit tenir compte des objectifs que le modèle doit permettre d'atteindre.

Il y a deux types de modélisation :

- **La modélisation a priori** : le système à modéliser n'existe pas encore et on veut déterminer ses principales caractéristiques.
- **La modélisation a posteriori** : le système à modéliser existe et on veut l'améliorer.

La modélisation a priori permet de :

- Comprendre le fonctionnement du système.
- Maîtriser sa complexité.
- Assurer sa cohérence.
- Pouvoir communiquer au sein de l'équipe de projet.

La modélisation a posteriori permet d' :

- Aider à la reconfiguration du système existant.
- Aider à l'amélioration de son fonctionnement.

Méthode d'élaboration de logiciels

Définition. Une **méthode d'élaboration de logiciels** définit une démarche qui décrit comment modéliser et construire des systèmes logiciels de qualité.

Une méthode d'élaboration de logiciel définit également une **représentation** (**textuelle** et/ou **graphique**) qui permet de :

- Manipuler facilement les modèles.
- Communiquer et d'échanger l'information entre les différents intervenants du projet.

Une méthode d'élaboration de logiciel définit des **règles de mise en œuvre** qui décrivent :

- L'articulation des différents points de vue.
- L'enchaînement des actions.
- L'ordonnancement des tâches.
- La répartition des responsabilités.

Ces règles définissent un **processus** qui assure l'harmonie au sein d'un ensemble d'éléments coopératifs et qui explique comment il convient de se servir de la méthode.

En principe, il existe deux grandes classes de méthodes d'élaboration de logiciel :

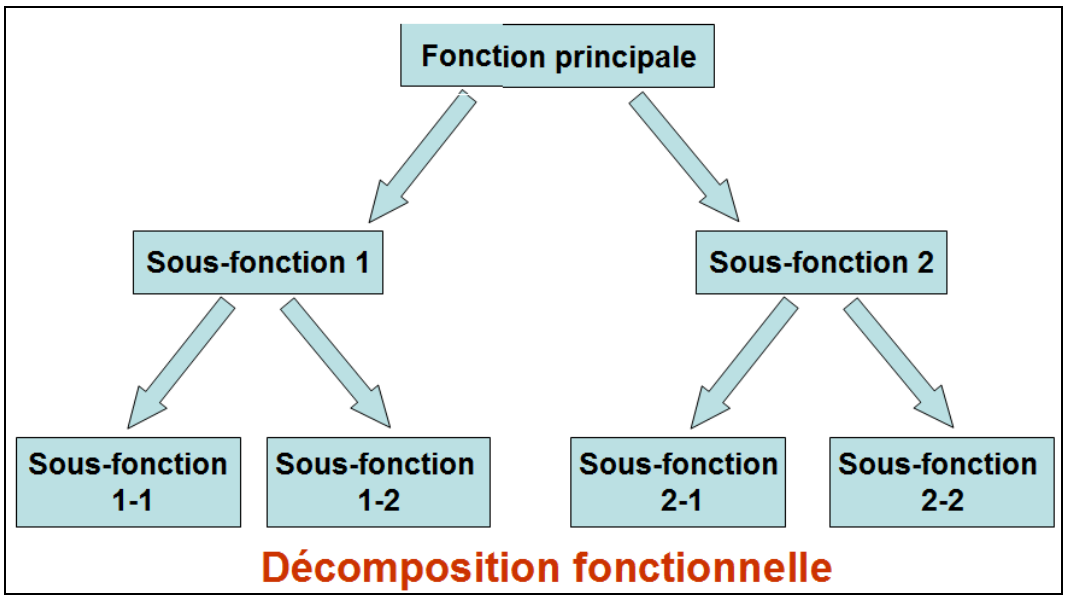
- Les méthodes de **décomposition fonctionnelle**.
- Les méthodes **orientées objet**.

Les méthodes fonctionnelles

Les méthodes de **décomposition fonctionnelle** sont caractérisées par le fait qu'elles appliquent une stratégie de modélisation qui se concentre sur **les fonctions** (traitements) que doit réaliser le système à concevoir. Suivant une approche fonctionnelle, le modèle du système est décomposé en fonctions, puis chaque fonction est décomposée à son tour en plusieurs sous-fonctions (**décomposition hiérarchique** et **descendante**).

Exemples de méthodes fonctionnelles :

- **SADT** *Structured Analysis & Design Technique* (Softech, USA).
- **SA** *Structured Analysis* (Marco).
- **SD** Structured Design (Yourdon & Costantine).



Avantages de l'approche fonctionnelle :

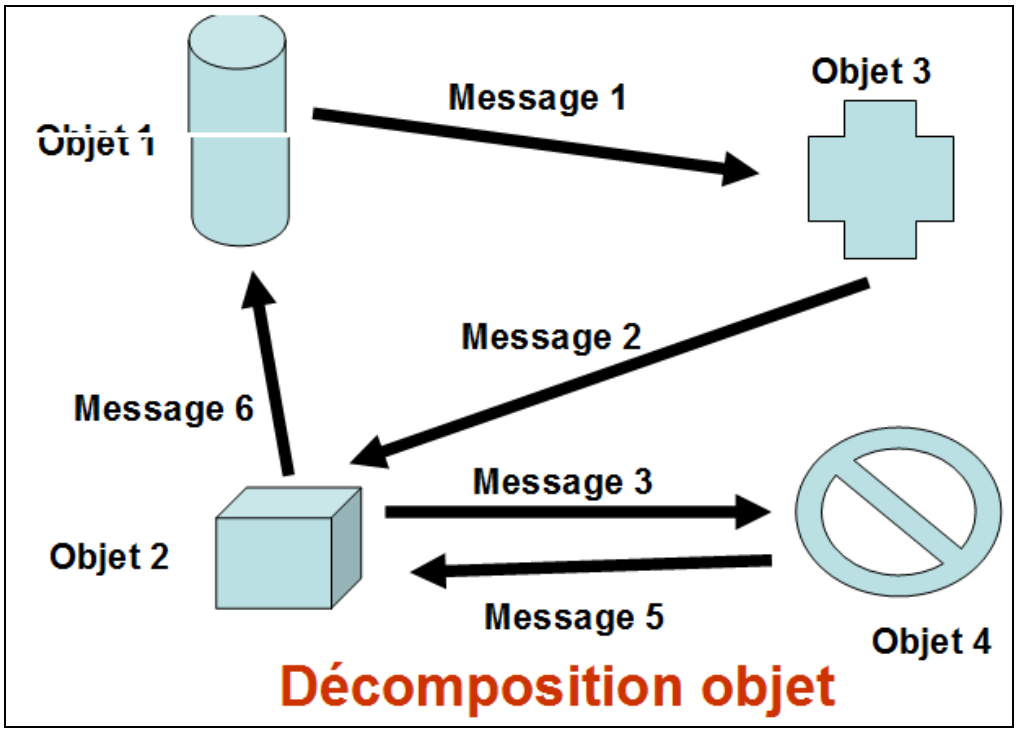
- Logique.
- Intuitive de la programmation.
- Ordonnée : structure hiérarchique des fonctions.

Inconvénients de l'approche fonctionnelle :

- Difficile si l'on veut assurer l'évolution du logiciel.
- Difficile si l'on veut réutiliser les parties déjà développées.
- Pas de structuration des données.

Les méthodes orientée Objet

Les **méthodes orientés objet** appliquent une stratégie de modélisation qui se base sur **les données** → les **objets**. Selon l'approche objet, le modèle du système est composé d'objets. On cherche à faire **collaborer un ensemble d'objets pour accomplir une fonction ou un service**. Cette collaboration s'effectue par **l'envoi de messages** entre les objets.



Avantages de l'approche orienté objet :

- Simple et naturel.
- Adaptée à un processus incrémental itératif.

Inconvénients de l'approche fonctionnelle :

- Parfois moins intuitive.
- Pas de concept de base qui assure comment modéliser les objets de manière pertinente.
- L'application du processus nécessite une grande maturité et beaucoup d'expérience.

Quelques méthodes orientés objet :

- **OOA** *Object-Oriented Analysis* (Coad).
- **OOD** *Object-Oriented Design* (Grady Booch).
- **HOOD** *Hierarchical Object-Oriented Design* (Cisi Ingénierie & CRI).
- **OMT** *Object Modeling Technique* (James Rambaugh).
- **OOSE** *Object-Oriented Software Engineering* (Ivar Jacobson).

Quelques langages de programmation orientés objet :

C++, Java, Ada, Simula, CLOS, Eiffel, Smalltalk.

Bibliographie du chapitre

1. ***Conception et programmation par objets : techniques, outils et applications*** [Jean-Pascal Aubert & Patrick Dixneuf].
2. ***Conception et programmation orientées objet*** [Bertrand Meyer].
3. ***Analyse orientée objets & modélisation pour la simulation*** [David R.C. Hill].
4. ***Modélisation objet avec UML*** [Pierre-Alain Muller].