Université de Msila – Département d'Informatique

Génie logiciel (GL2)

(3ème année SI-ISIL)

Dr BOUNIF M-E

- **III-1- Le *cycle de vie d'un logiciel***

- Le *cycle de vie d'un logiciel* (en anglais *software lifecycle*), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition.
- L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.

- L'origine de ce découpage provient du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. 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.

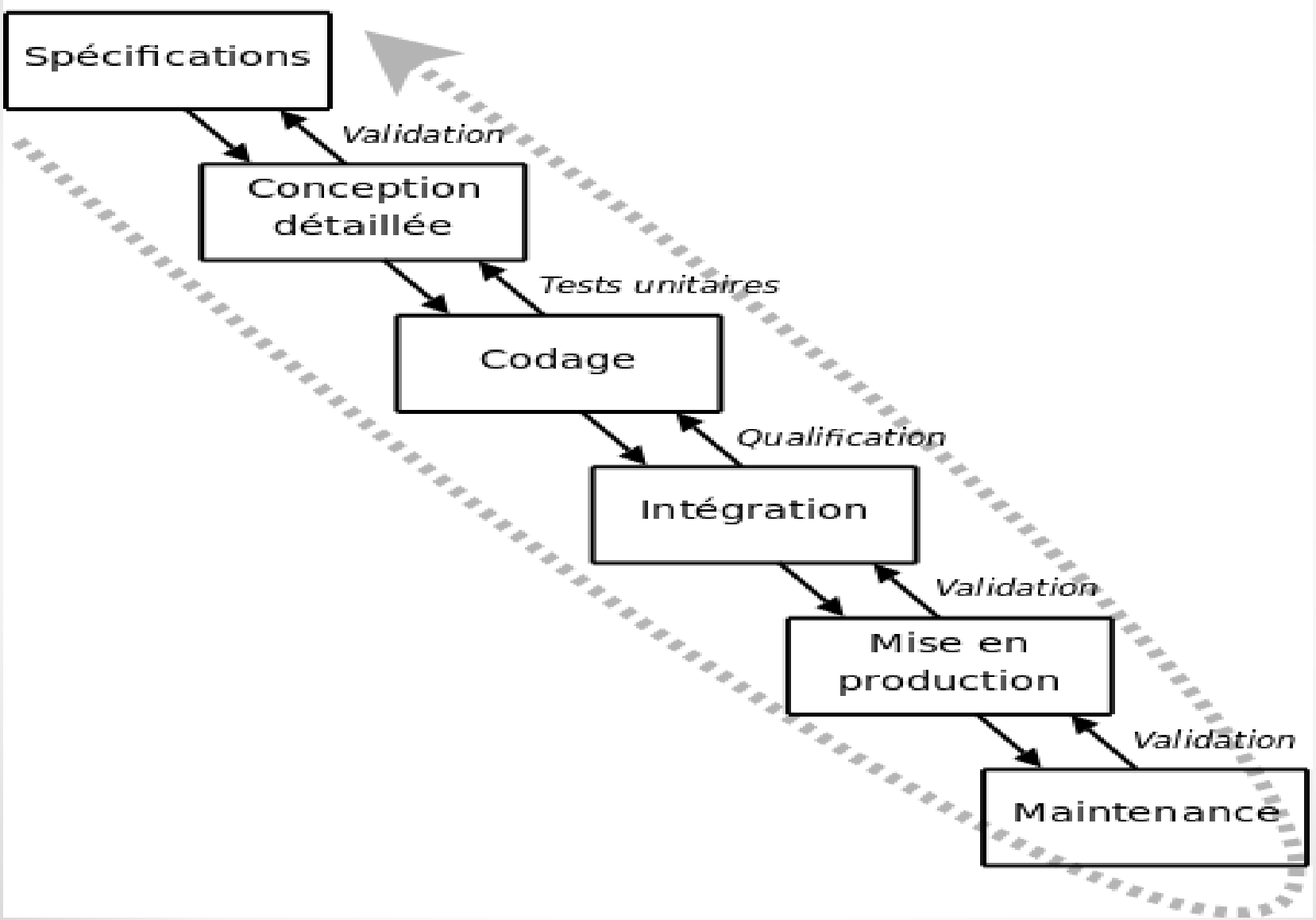
- Le cycle de vie du logiciel comprend généralement au minimum les étapes suivantes :

- **Analyse des besoins et faisabilité**
- c'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes, puis l'estimation de la faisabilité de ces besoins ;
- **Spécifications ou conception générale**
- il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel ;
- **Conception détaillée**
- cette étape consiste à définir précisément chaque sous-ensemble du logiciel ;
- **Codage (Implémentation ou programmation)**
- c'est la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception ;
- **Tests unitaires**
- ils permettent de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications ;

- **Intégration**
- l'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de tests d'intégration consignés dans un document ;
- **Qualification (ou recette)**
- c'est-à-dire la vérification de la conformité du logiciel aux spécifications initiales ;
- **Documentation**
- elle vise à produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs ;
- **Mise en production**
- c'est le déploiement sur site du logiciel ;
- **Maintenance**
- elle comprend toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

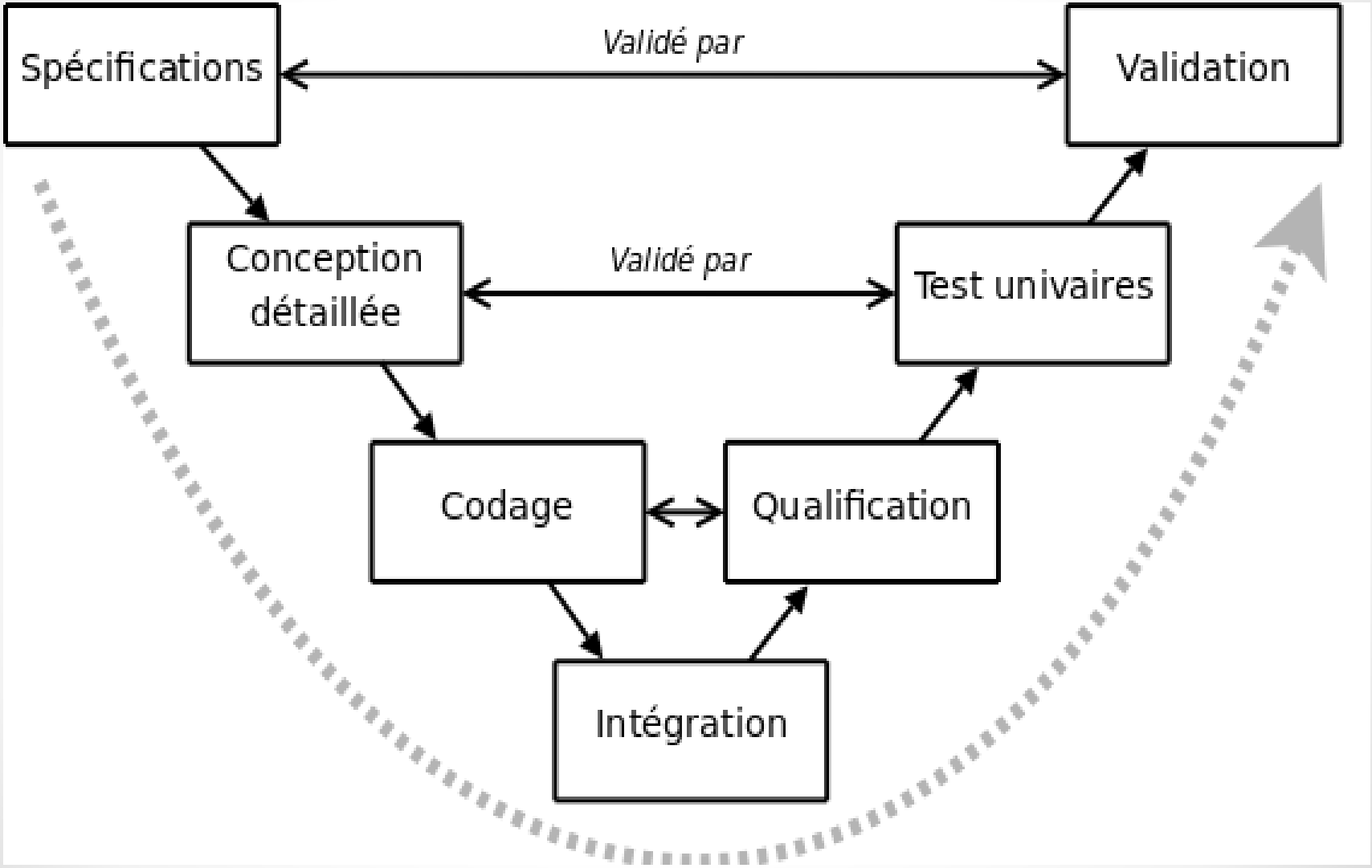
- La séquence et la présence de chacune de ces activités dans le cycle de vie dépendent du choix d'un modèle de cycle de vie entre le client et l'équipe de développement. Le cycle de vie permet de prendre en compte, en plus des aspects techniques, l'organisation et les aspects humains.

- III-2- Modèles de cycles de vie d'un logiciel
- a. Modèle de cycle de vie en cascade



- Dans ce modèle le principe est très simple : chaque phase se termine à une date précise par la production de certains documents ou logiciels. Les résultats sont définis sur la base des interactions entre étapes, ils sont soumis à une revue approfondie et on ne passe à la phase suivante que s'ils sont jugés satisfaisants.
- Le modèle original ne comportait pas de possibilité de retour en arrière. Celle-ci a été rajoutée ultérieurement sur la base qu'une étape ne remet en cause que l'étape précédente, ce qui, dans la pratique, s'avère insuffisant.
- L'inconvénient majeur du modèle de cycle de vie en cascade est que la vérification du bon fonctionnement du système est réalisée trop tardivement : lors de la phase d'intégration, ou pire, lors de la mise en production.

- **b. Modèle de cycle de vie en V**



- **Le modèle en V demeure actuellement le cycle de vie le plus connu et certainement le plus utilisé. Il s'agit d'un modèle en cascade dans lequel le développement des tests et du logiciel sont effectués de manière synchrone.**
- Le principe de ce modèle est qu'avec toute décomposition doit être décrite la recomposition et que toute description d'un composant est accompagnée de tests qui permettront de s'assurer qu'il correspond à sa description.
- Ceci rend explicite la préparation des dernières phases (validation-vérification) par les premières (construction du logiciel), et permet ainsi d'éviter un écueil bien connu de la spécification du logiciel : énoncer une propriété qu'il est impossible de vérifier objectivement après la réalisation.
- Cependant, ce modèle souffre toujours du problème de la vérification trop tardive du bon fonctionnement du système.

- **c. Modèle de cycle de vie en spirale**

- ce modèle est beaucoup plus général que le précédent. Il met l'accent sur l'activité d'analyse des risques : chaque cycle de la spirale se déroule en quatre phases :
- Détermination, à partir des résultats des cycles précédents, ou de l'analyse préliminaire des besoins, des objectifs du cycle, des alternatives pour les atteindre et des contraintes ;
- Analyse des risques, évaluation des alternatives et, éventuellement maquettage ;
- Développement et vérification de la solution retenue, un modèle « classique » (cascade ou en V) peut être utilisé ici ;
- Revue des résultats et vérification du cycle suivant.
- L'analyse préliminaire est affinée au cours des premiers cycles. Le modèle utilise des maquettes exploratoires pour guider la phase de conception du cycle suivant. Le dernier cycle se termine par un processus de développement classique.

- **d. Modèle par incrément**

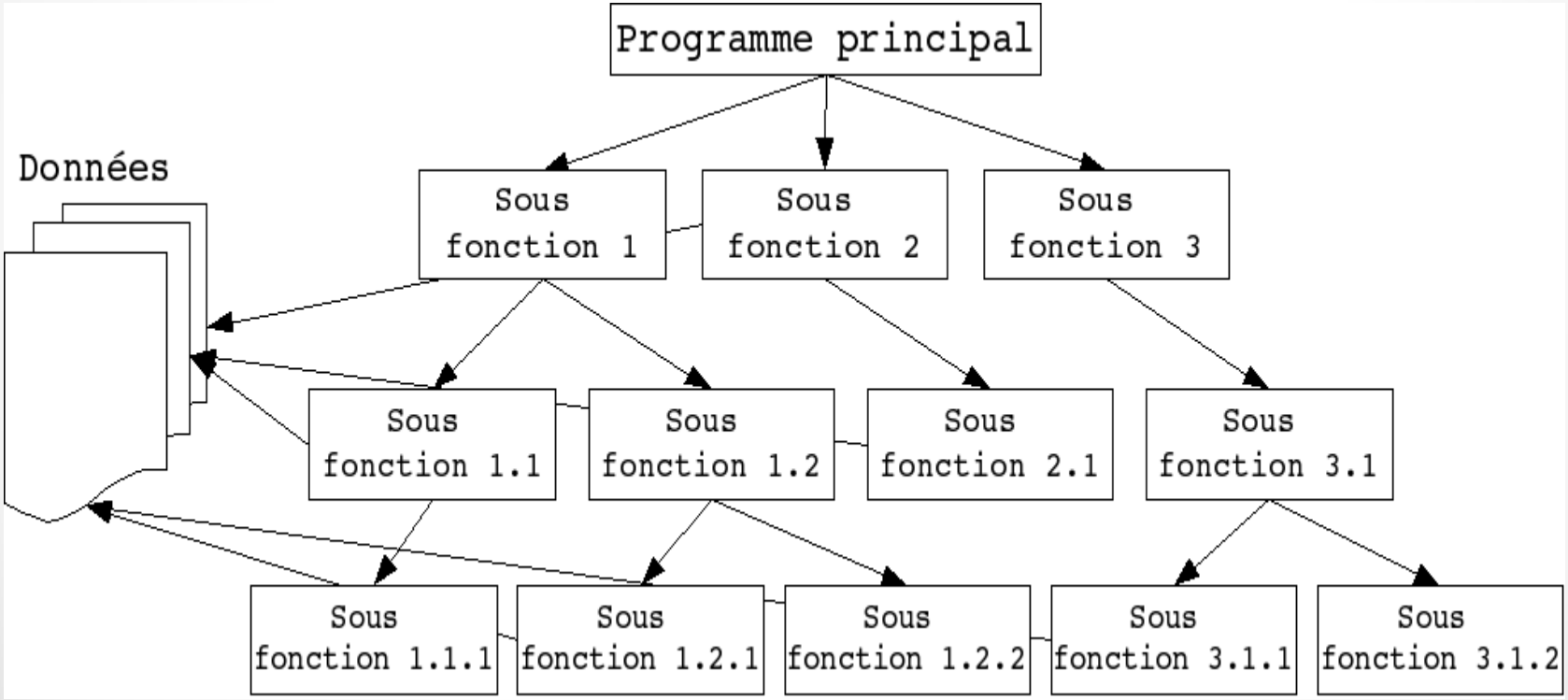
- Dans les modèles précédents, un logiciel est décomposé en composants développés séparément et intégrés à la fin du processus.
- Dans les modèles par incrément un seul ensemble de composants est développé à la fois : des incréments viennent s'intégrer à un noyau de logiciel développé au préalable. Chaque incrément est développé selon l'un des modèles précédents.
- Les avantages de ce type de modèle sont les suivants :
- chaque développement est moins complexe ;
- les intégrations sont progressives ;
- il est ainsi possible de livrer et de mettre en service chaque incrément ;
- il permet un meilleur lissage du temps et de l'effort de développement grâce à la possibilité de recouvrement (parallélisation) des différentes phases.
- Les risques de ce type de modèle sont les suivants :
- remettre en cause les incréments précédents ou pire le noyau ;
- ne pas pouvoir intégrer de nouveaux incréments.
- Les noyaux, les incréments ainsi que leurs interactions doivent donc être spécifiés globalement, au début du projet. Les incréments doivent être aussi indépendants que possible, fonctionnellement, mais aussi sur le plan du calendrier du développement.

- **4. Méthodes d'analyse et de conception**

- Les méthodes d'analyse et de conception fournissent une méthodologie et des notations standards qui aident à concevoir des logiciels de qualité. Il existe différentes manières pour classer ces méthodes, dont :

- La distinction entre composition et décomposition :
- Elle met en opposition d'une part les méthodes ascendantes qui consistent à construire un logiciel par composition à partir de modules existants et, d'autre part, les méthodes descendantes qui décomposent récursivement le système jusqu'à arriver à des modules programmables simplement ;
- La distinction entre fonctionnelle (dirigée par le traitement) et orientée objet :
- Dans la stratégie fonctionnelle (également qualifiée de structurée) un système est vu comme un ensemble hiérarchique d'unités en interaction, ayant chacune une fonction clairement définie. Les fonctions disposent d'un état local, mais le système a un état partagé, qui est centralisé et accessible par l'ensemble des fonctions. Les stratégies orientées objet considèrent qu'un système est un ensemble d'objets interagissants. Chaque objet dispose d'un ensemble d'attributs décrivant son état et l'état du système est décrit (de façon décentralisée) par l'état de l'ensemble.

- VI- De la programmation structurée à l'approche orientée objet
- VI-1 Méthodes fonctionnelles ou structurées



- Les méthodes fonctionnelles (également qualifiées de méthodes structurées) trouvent leur origine dans les langages procéduraux. Elles mettent en évidence les fonctions à assurer et proposent une approche hiérarchique descendante et modulaire.
- Chaque niveau est ensuite décomposé en respectant les entrées/sorties du niveau supérieur. La décomposition se poursuit jusqu'à arriver à des composants maîtrisables.
- L'approche fonctionnelle dissocie le problème de la représentation des données, du problème du traitement de ces données.

- **b. L'approche orientée objet**
- L'approche considère le logiciel comme une collection d'objets dissociés, identifiés et possédant des caractéristiques. Une caractéristique est soit un attribut (*i.e.* une donnée caractérisant l'état de l'objet), soit une entité comportementale de l'objet (*i.e.* une fonction). La fonctionnalité du logiciel émerge alors de l'interaction entre les différents objets qui le constituent. L'une des particularités de cette approche est qu'elle rapproche les données et leurs traitements associés au sein d'un unique objet.