

UML

Etape2_00

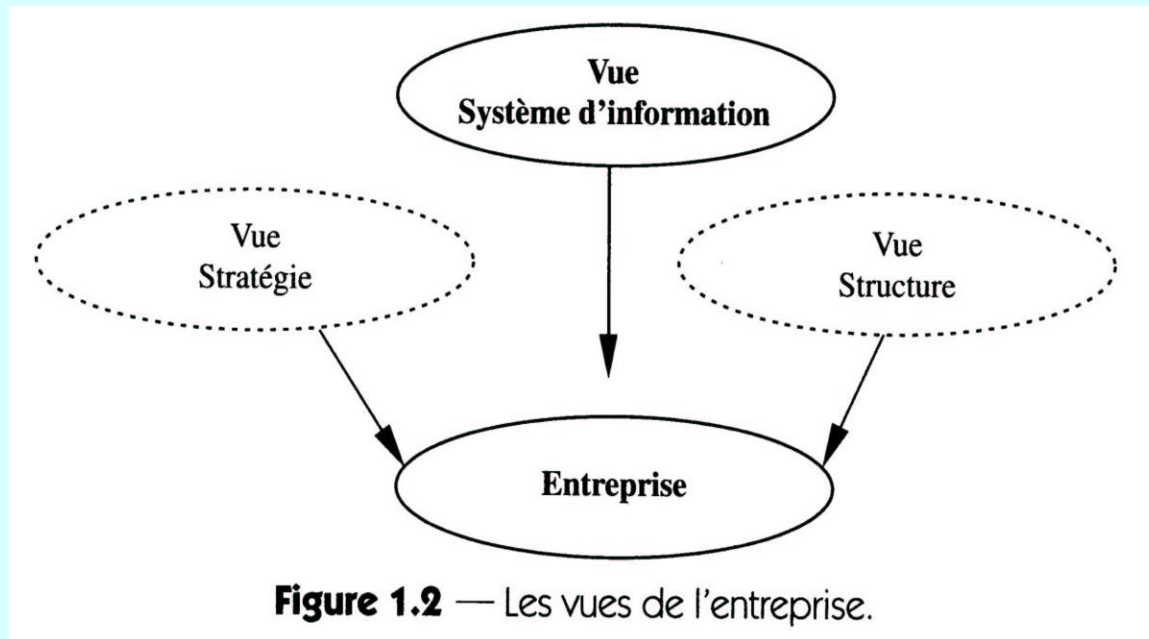
UML et l'Entreprise

UML

- **L'information et l'entreprise.**
- **Évènement, Acteur, Processus.**
- **UML et la conception de S.I.**
- **L'Orienté Objet.**
- **UML en pratique.**

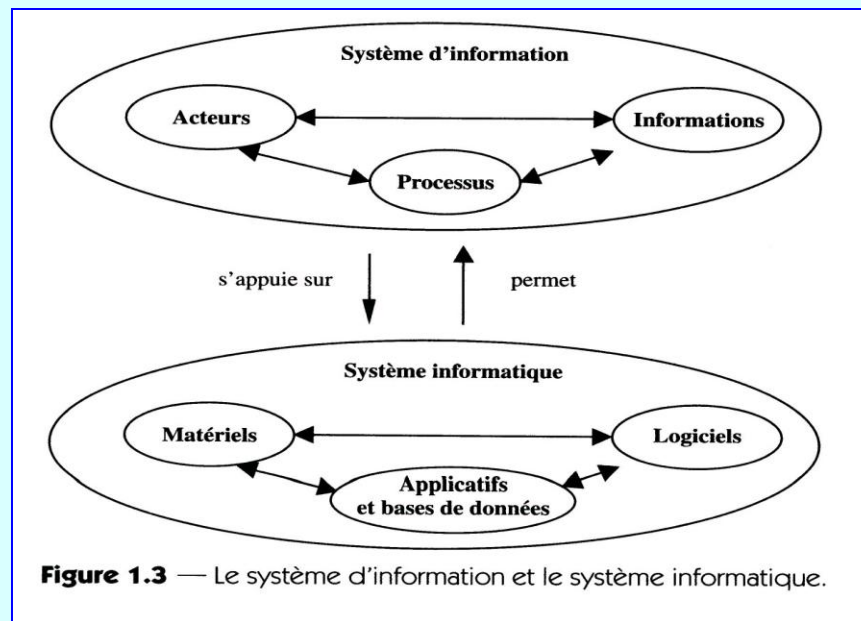
Une vision formelle de l'entreprise

- Le Système d'Information.
 - L'information, support pour l'action.
 - L'information conserve la trace des activités.
 - L'information apporte une aide à la prise de décision.
- Nouvelle perspective sur l'entreprise.



Une vision formelle de l'entreprise

- Le Système d'Information.
 - L'information, support pour l'action.
 - L'information conserve la trace des activités.
 - L'information apporte une aide à la prise de décision.
- Nouvelle perspective sur l'entreprise.



Une vision formelle de l'entreprise

Le système d'information est une vision du réel construite sur la base

- **d'informations organisées,**
 - **d'évènements ayant un effet sur ces informations, et**
 - **d'acteurs qui agissent sur ces informations ou à partir de ces informations,**
- selon des processus visant une finalité de gestion et utilisant les technologies de l'information**

Information et Informatisation : le Génie Logiciel

L'informatisation

- L'informatisation est le phénomène le plus important de notre époque.
- Actuellement, l'informatique est au cœur de toutes les grandes entreprises.
- Le système d'information d'une entreprise est composé de matériels et de logiciels : 20 % pour le matériel et 80 % pour le logiciel.
- Par exemple, aujourd'hui, 90% des nouvelles fonctionnalités des automobiles sont apportées par l'électronique et l'informatique embarquées.

Information et Informatisation : le Génie Logiciel

Les Logiciels

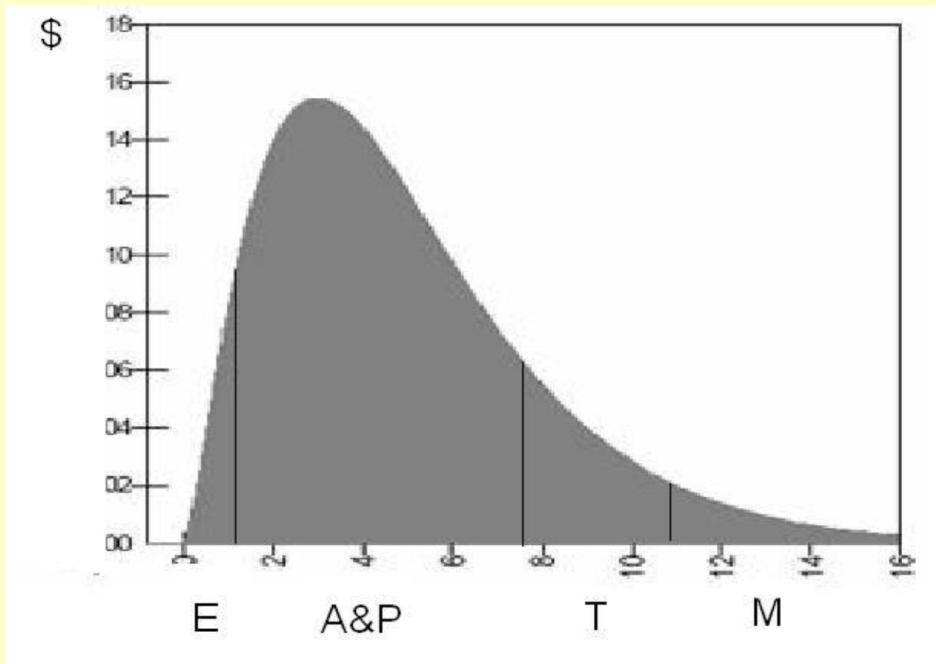
- Un logiciel ou une application est un ensemble de programmes, qui permet à un ordinateur ou à un système informatique d'assurer une tâche ou une fonction en particulier (exemple : logiciel de comptabilité, logiciel de gestion des prêts).
- En 1995, une étude du *Standish Group* dressait un tableau accablant de la conduite des projets informatiques.
- Reposant sur un échantillon représentatif de 365 entreprises, totalisant 8 380 applications, cette étude établissait que :
 - 16,2% seulement des projets étaient conformes aux prévisions initiales,
 - 52,7% avaient subi des dépassements en coût et délai d'un facteur 2 à 3 avec diminution du nombre des fonctions offertes,
 - 31,1% ont été purement abandonnés durant leur développement.

Information et Informatisation : le Génie Logiciel

Extrait de cours SQL

SQL

- **Courbe traditionnelle des projets informatiques**



E : Etudes

A&P : Analyse et programmation

T: Tests

M: Mise en place

\$: Millions de \$

Information et Informatisation : le Génie Logiciel

Les Logiciels

- Pour les grandes entreprises (qui lancent proportionnellement davantage de gros projets), le taux de succès est de 9% seulement, 37% des projets sont arrêtés en cours de réalisation, 50% aboutissent hors délai et hors budget.
- L'examen des causes de succès et d'échec est instructif : la plupart des échecs proviennent non de l'informatique, mais de la maîtrise d'ouvrage (*i.e.* le client).
- Pour ces raisons, le développement de logiciels dans un contexte professionnel suit souvent des règles strictes encadrant la conception et permettant le travail en groupe et la maintenance du code .

Ainsi, une nouvelle discipline est née : le **génie logiciel**.

- Comparativement à sa production, le coût du développement d'un logiciel est extrêmement important. Mais la maintenance coûte également très cher.
- Souvent, les personnes qui doivent opérer des modifications ultérieures dans le code ne sont plus les personnes qui l'ont développé.

Information et Informatisation : le Génie Logiciel

Le Génie Logiciel

- Le génie logiciel est un domaine de recherche qui a été défini (fait rare) du 7 au 11 octobre 1968, à Garmisch-Partenkirchen, sous le parrainage de l'OTAN.
- Il a pour objectif de répondre à un problème qui s'énonçait en deux constatations : d'une part le logiciel n'était pas fiable, d'autre part, il était incroyablement difficile de réaliser dans des délais prévus des logiciels satisfaisant leur cahier des charges.
- L'objectif du génie logiciel est d'optimiser le coût de développement du logiciel. L'importance d'une approche méthodologique s'est imposée à la suite de la crise de l'industrie du logiciel à la fin des années 1970.
- Cette crise de l'industrie du logiciel était principalement due à :
 - l'augmentation des coûts ;
 - les difficultés de maintenance et d'évolution ;
 - la non-fiabilité ;
 - le non-respect des spécifications ;
 - le non-respect des délais.

Information et Informatisation : le Génie Logiciel

- La maintenance est devenue une facette très importante du cycle de vie d'un logiciel.
- En effet, une enquête effectuée aux É.-U. en 1986 auprès de 55 entreprises révèle que 53% du budget total d'un logiciel est affecté à la maintenance.
- Ce coût est réparti comme suit :
 - 34% maintenance évolutive (modification des spécifications initiales) ;
 - 10% maintenance adaptative (nouvel environnement, nouveaux utilisateurs) ;
 - 17% maintenance corrective (correction des bogues) ;
 - 16% maintenance perfective (améliorer les performances sans changer les spécifications) ;
 - 6% assistance aux utilisateurs ;
 - 6% contrôle qualité ;
 - 7% organisation/suivi ;
 - 4% divers.

Information et Informatisation : le Génie Logiciel

Pour apporter une réponse à tous ces problèmes, le génie logiciel s'intéresse particulièrement à la manière dont le code source d'un logiciel est spécifié puis produit.

- Ainsi, le génie logiciel touche au cycle de vie des logiciels :
 - l'analyse du besoin,
 - l'élaboration des spécifications,
 - la conceptualisation,
 - le développement,
 - la phase de test,
 - la maintenance.
 - Les projets relatifs à l'ingénierie logicielle sont généralement de grande envergure et dépassent souvent les 10 000 lignes de code.
 - C'est pourquoi ces projets nécessitent une équipe de développement bien structurée.
- La gestion de projet se retrouve naturellement intimement liée au génie logiciel.

Information et Informatisation : le Génie Logiciel

Notion de Qualité

En génie logiciel, divers travaux ont mené à la définition de la qualité du logiciel en termes de facteurs, qui dépendent, entre autres, du domaine de l'application et des outils utilisés.

Parmi ces derniers l'on peut citer :

Validité :

- aptitude d'un produit logiciel à remplir exactement ses fonctions, définies par le cahier des charges et les spécifications.

Fiabilité ou robustesse :

- aptitude d'un produit logiciel à fonctionner dans des conditions anormales.

Extensibilité (maintenance) :

- facilité avec laquelle un logiciel se prête à sa maintenance, c'est-à-dire à une modification ou à une extension des fonctions qui lui sont demandées.

Réutilisabilité :

- aptitude d'un logiciel à être réutilisé, en tout ou en partie, dans de nouvelles applications.

Information et Informatisation

Compatibilité :

- facilité avec laquelle un logiciel peut être combiné avec d'autres logiciels.

Efficacité :

- Utilisation optimale des ressources matérielles.

Portabilité :

- facilité avec laquelle un logiciel peut être transféré sous différents environnements matériels et logiciels.

Vérifiabilité :

- facilité de préparation des procédures de test.

Intégrité :

- aptitude d'un logiciel à protéger son code et ses données contre des accès non autorisés.

Facilité d'emploi :

- facilité d'apprentissage, d'utilisation, de préparation des données, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation.

Ces facteurs sont parfois contradictoires, le choix des compromis doit s'effectuer en fonction du contexte.

Modélisation, Cycle de Vie et Méthodes

Qu'est ce qu'un modèle?

- Un *modèle* est une représentation abstraite et simplifiée (*i.e.* qui exclut certains détails), d'une entité (phénomène, processus, système, etc.) du monde réel en vue de le décrire, de l'expliquer ou de le prévoir.

Modèle est synonyme de théorie, mais avec une connotation pratique : un modèle, c'est une théorie orientée vers l'action qu'elle doit servir.

- Concrètement, un modèle permet de réduire la complexité d'un phénomène en éliminant les détails qui n'influencent pas son comportement de manière significative.

Il reflète ce que le concepteur croit important pour la compréhension et la prédiction du phénomène modélisé.

- Les limites du phénomène modélisé dépendant des objectifs du modèle.

Modélisation, Cycle de Vie et Méthodes

Voici quelques exemples de modèles :

Modèle météorologique

- à partir de données d'observation (satellite...), il permet de prévoir les conditions climatiques pour les jours à venir.

Modèle économique

- peut par exemple permettre de simuler l'évolution de cours boursiers en fonction d'hypothèses macro-économiques (évolution du chômage, taux de croissance...).

Modèle démographique

- définit la composition d'un panel d'une population et son comportement, dans le but de fiabiliser des études statistiques, d'augmenter l'impact de démarches commerciales, etc.

Modélisation, Cycle de Vie et Méthodes

Plans

- Les plans sont des modèles qui donnent une vue d'ensemble du système concerné. Par exemple, dans le bâtiment, pour la construction d'un immeuble, il faut préalablement élaborer de nombreux plans :
 - plans d'implantation du bâtiment dans son environnement ;
 - plans généraux du bâtiment et de sa structure ;
 - plans détaillés des différents locaux, bureaux, appartements...
 - plans des câblages électriques ;
 - plans d'écoulements des eaux, etc.
- Les trois premiers exemples sont des modèles que l'on qualifie de **prédictifs**.
- Le dernier, plus **conceptuel**, possède différents niveaux de vues comme la plupart des modèles en génie logiciel.

Modélisation, Cycle de Vie et Méthodes

Pourquoi modéliser?

- Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence.
- Un modèle est un langage commun, précis, qui est connu par tous les membres de l'équipe et il est donc, à ce titre, un vecteur privilégié pour communiquer. Cette communication est essentielle pour aboutir à une compréhension commune aux différentes parties prenantes (notamment entre la maîtrise d'ouvrage et la maîtrise d'œuvre informatique) et précise d'un problème donné.
- Dans le domaine de l'ingénierie du logiciel, le modèle permet de mieux répartir les tâches et d'automatiser certaines d'entre elles.
C'est également un facteur de réduction des coûts et des délais.

Modélisation, Cycle de Vie et Méthodes

Pourquoi modéliser?

- Par exemple, les plateformes de modélisation savent maintenant exploiter les modèles pour faire de la génération de code (au moins au niveau du squelette) voire des allers-retours entre le code et le modèle sans perte d'information.
- Le modèle est enfin indispensable pour assurer un bon niveau de qualité et une maintenance efficace.

En effet, une fois mise en production, l'application va devoir être maintenue, probablement par une autre équipe et, qui plus est, pas nécessairement de la même société que celle ayant créé l'application.

- Le choix du modèle a donc une influence capitale sur les solutions obtenues. Les systèmes non triviaux sont mieux modélisés par un ensemble de modèles indépendants. Selon les modèles employés, la démarche de modélisation n'est pas la même

Modélisation, Cycle de Vie et Méthodes

Qui doit modéliser?

- La modélisation est souvent faite par la maîtrise d'œuvre informatique (MOE). C'est malencontreux, car les priorités de la MOE résident dans le fonctionnement de la plateforme informatique et non dans les processus de l'entreprise.
- Il est préférable que la modélisation soit réalisée par la maîtrise d'ouvrage (MOA) de sorte que le métier soit maître de ses propres concepts.
La MOE doit intervenir dans le modèle lorsque, après avoir défini les concepts du métier, on doit introduire les contraintes propres à la plateforme informatique.
- Il est vrai que certains métiers, dont les priorités sont opérationnelles, ne disposent pas toujours de la capacité d'abstraction et de la rigueur conceptuelle nécessaires à la formalisation.
- La professionnalisation de la MOA a pour but de les doter de ces compétences. Cette professionnalisation réside essentiellement dans l'aptitude à modéliser le système d'information du métier : le maître mot est *modélisation*.

Modélisation, Cycle de Vie et Méthodes

Qui doit modéliser?

- Lorsque le modèle du système d'information est de bonne qualité, sobre, clair, stable, la maîtrise d'œuvre peut travailler dans de bonnes conditions.
- Lorsque cette professionnalisation a lieu, elle modifie les rapports avec l'informatique et déplace la frontière des responsabilités, ce qui contrarie parfois les informaticiens dans un premier temps, avant qu'ils n'en voient apparaître les bénéfices.

Rappel :

Maître d'ouvrage (MOA) :

- Le MOA est une personne morale (entreprise, direction, etc.), une entité de l'organisation. Ce n'est jamais une personne.

Maître d'œuvre (MOE) :

- Le MOE est une personne morale (entreprise, direction, etc.) garante de la bonne réalisation technique des solutions.

Il a, lors de la conception du SI, un devoir de conseil vis-à-vis du MOA, car le SI doit tirer le meilleur parti des possibilités techniques.

Modélisation, Cycle de Vie et Méthodes

- Le MOA est client du MOE à qui il passe commande d'un produit nécessaire à son activité.

Le MOE fournit ce produit ; soit il le réalise lui-même, soit il passe commande à un ou plusieurs fournisseurs (« entreprises ») qui élaborent le produit sous sa direction.

La relation MOA et MOE est définie par un contrat qui précise leurs engagements mutuels.

- Lorsque le produit est compliqué, il peut être nécessaire de faire appel à plusieurs fournisseurs. Le MOE assure leur coordination ; il veille à la cohérence des fournitures et à leur compatibilité.

Il coordonne l'action des fournisseurs en contrôlant la qualité technique, en assurant le respect des délais fixés par le MOA et en minimisant les risques.

Le MOE est responsable de la qualité technique de la solution. Il doit, avant toute livraison au MOA, procéder aux vérifications nécessaires (« recette usine »).

Modélisation, Cycle de Vie et Méthodes

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.

Modélisation, Cycle de Vie et Méthodes

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 ;

Modélisation, Cycle de Vie et Méthodes

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 ;

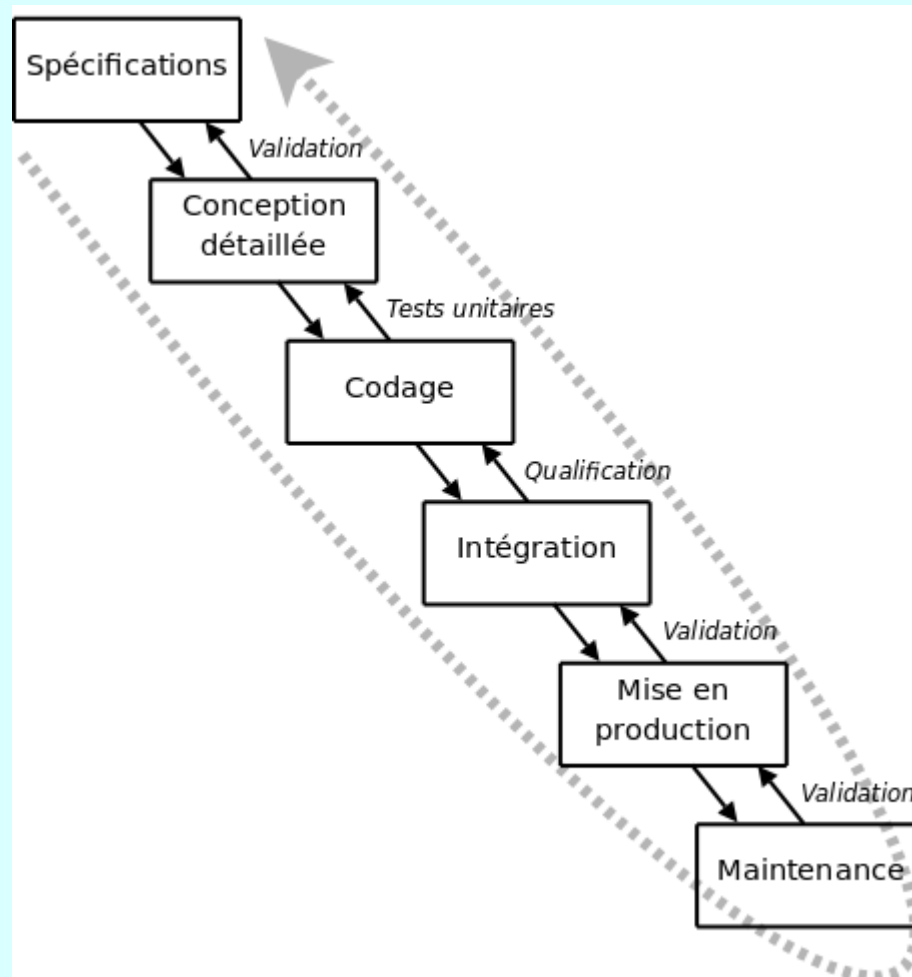
Modélisation, Cycle de Vie et Méthodes

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.

Modèles de cycle de vie d'un logiciel

- *Cycle de vie en cascade*



Modèles de cycle de vie d'un logiciel

Cycle de vie en cascade

- Le modèle de cycle de vie en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970.
- 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.

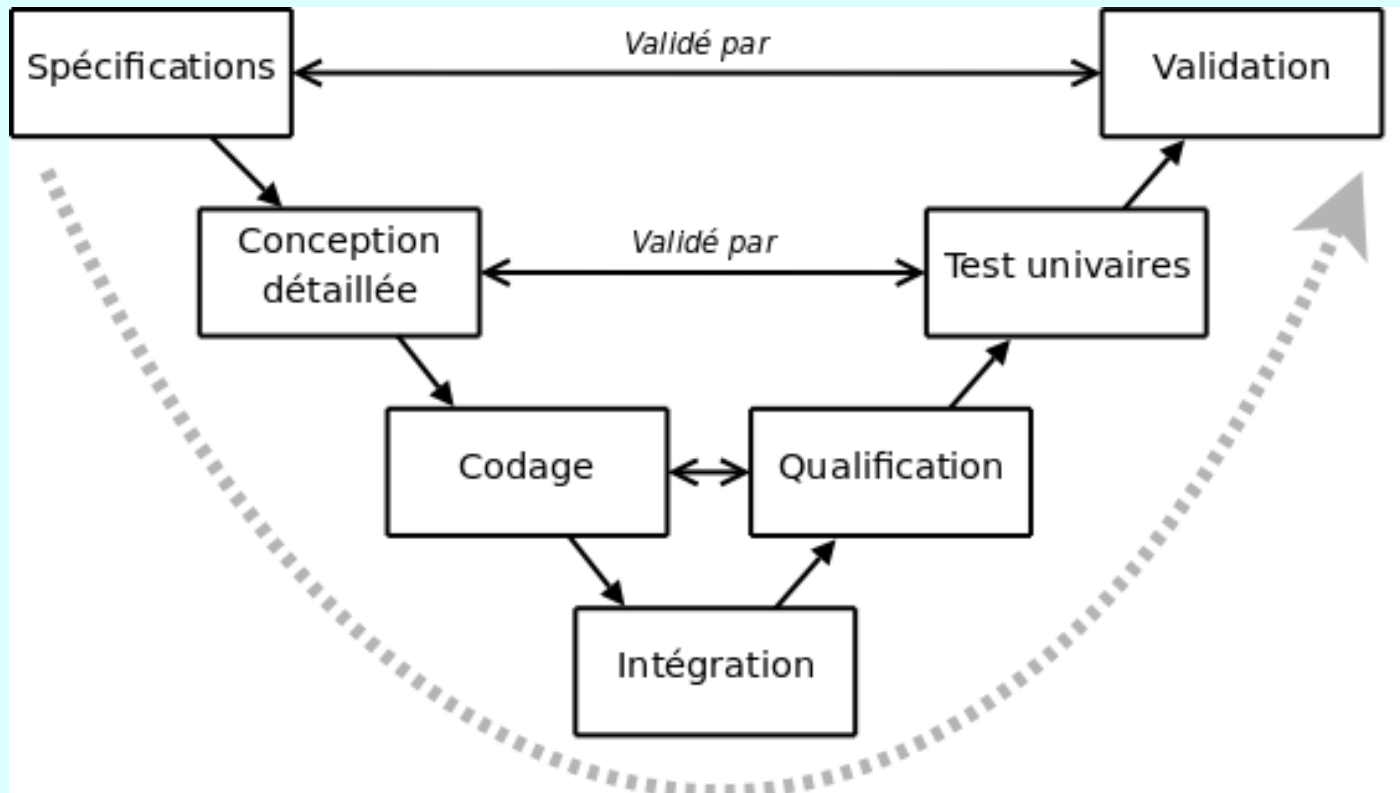
Celle-ci a été rajoutée ultérieurement sur la base qu'une étape ne remet en cause que l'étape précédente, ce qui, dans la pratique, s'avère insuffisant.

- 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.

Modèles de cycle de vie d'un logiciel

Modèles de cycle de vie d'un logiciel

- *Cycle de vie en V*



Modèles de cycle de vie d'un logiciel

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.

Modèles de cycle de vie d'un logiciel

Cycle de vie en Spirale

Proposé par B. Boehm en 1988, 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.

Modèles de cycle de vie d'un logiciel

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.

Modèles de cycle de vie d'un logiciel

Modèle par Incrément

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.

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 ;

Méthodes d'Analyse et de Conception

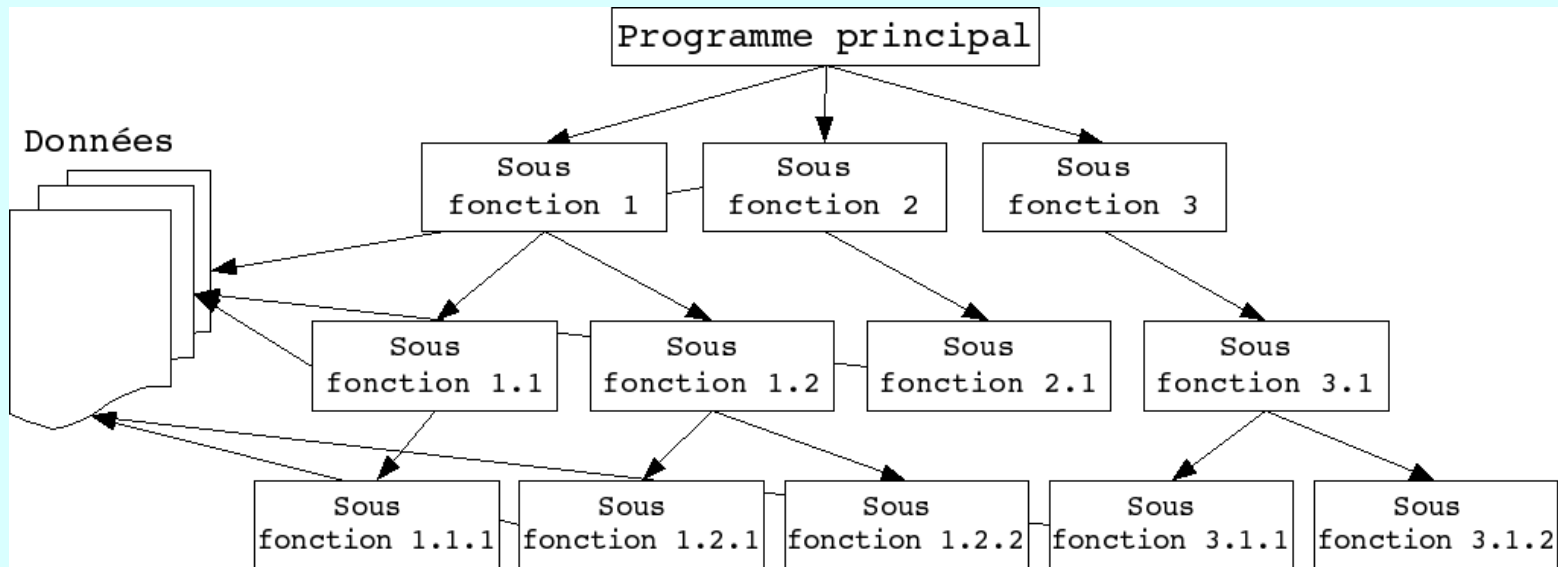
La distinction entre fonctionnelle (dirigée par le traitement) et orientée objet :

- 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.

Méthodes d'Analyse et de Conception

Méthodes fonctionnelles ou structurées



Méthodes d'Analyse et de Conception

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.
- Ces méthodes utilisent intensivement les raffinements successifs pour produire des spécifications dont l'essentiel est sous forme de notation graphique en diagrammes de flots de données.
- Le plus haut niveau représente l'ensemble du problème (sous forme d'activité, de données ou de processus, selon la méthode).
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.
Sur la figure, les données du problème sont représentées sur la gauche.

Méthodes d'Analyse et de Conception

- Des flèches transversales matérialisent la manipulation de ces données par des sous-fonctions.
- Cet accès peut-être direct (c'est parfois le cas quand les données sont regroupées dans une base de données), ou peut être réalisé par le passage de paramètre depuis le programme principal.
- La SADT (*Structured Analysis Design Technique*) est probablement la méthode d'analyse fonctionnelle et de gestion de projets la plus connue.
- Elle permet non seulement de décrire les tâches du projet et leurs interactions, mais aussi de décrire le système que le projet vise à étudier, créer ou modifier, en mettant notamment en évidence les parties qui constituent le système, la finalité et le fonctionnement de chacune, ainsi que les interfaces entre ces diverses parties.
- Le système ainsi modélisé n'est pas une simple collection d'éléments indépendants, mais une organisation structurée de ceux-ci dans une finalité précise.
- **En résumé, l'architecture du système est dictée par la réponse au problème (i.e. la fonction du système)**

Méthodes d'Analyse et de Conception

- Document en appui : « Historique Méthodes.pdf »
aperçu historique des méthodes de développement de projets.

L'approche Orientée Objet

De la programmation structurée à l'approche orientée objet

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.

De la programmation structurée à l'approche orientée objet

Comme on vient de le dire, un objet est caractérisé par plusieurs notions :

L'identité

- L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état.

On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro de série, etc.) ;

Les attributs

- Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations sur l'état de l'objet ;

Les méthodes

- Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées opérations) que l'objet est à même de réaliser.

De la programmation structurée à l'approche orientée objet

- Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets).
De plus, les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier.
- La difficulté de cette modélisation consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle (chien, voiture, ampoule, personne...) ou bien virtuelle (client, temps...).
- La Conception Orientée Objet (COO) est la méthode qui conduit à des architectures logicielles fondées sur les objets du système, plutôt que sur la fonction qu'il est censé réaliser

En résumé, l'architecture du système est dictée par la structure du problème.

Approche fonctionnelle vs approche objet

- Selon la thèse de Church-Turing, tout langage de programmation non trivial équivaut à une machine de Turing.
- Il en résulte que tout programme qu'il est possible d'écrire dans un langage pourrait également être écrit dans n'importe quel autre langage.
- Ainsi, **tout ce que l'on fait avec un langage de programmation par objets pourrait être fait en programmation impérative.**

La différence entre une approche fonctionnelle et une approche objet n'est donc pas d'ordre logique, mais pratique.

- L'approche structurée privilégie la fonction comme moyen d'organisation du logiciel. Ce n'est pas pour cette raison que l'approche objet est une approche *non fonctionnelle*. En effet, les méthodes d'un objet sont des fonctions.
- Ce qui différencie sur le fond l'approche objet de l'approche fonctionnelle, c'est que les fonctions obtenues à l'issue de la mise en œuvre de l'une ou l'autre méthode sont distinctes.

L'approche objet est une approche orientée donnée.

- Dans cette approche, les fonctions se déduisent d'un regroupement de champs de données formant une entité cohérente, logique, tangible et surtout stable quant au problème traité.

Approche fonctionnelle vs approche objet

- Présenter L'approche structurée classique privilégie une organisation des données postérieure à la découverte des grandes, puis petites **fonctions** qui les décomposent, l'ensemble constituant les services qui répondent aux besoins.
- En approche objet, l'évolution des besoins aura le plus souvent tendance à se présenter comme un **changement de l'interaction des objets**.
- S'il faut apporter une modification aux données, seul l'objet incriminé (encapsulant cette donnée) sera modifié.
Toutes les fonctions à modifier sont bien identifiées : elles se trouvent dans ce même objet : ce sont ses méthodes.
- Dans une approche structurée, l'évolution des besoins entraîne souvent une dégénérescence, ou une profonde remise en question, de la topologie typique de la figure présentée précédemment, car la décomposition des unités de traitement (du programme principal aux sous-fonctions) est directement dictée par ces besoins.
- D'autre part, une modification des données entraîne généralement une modification d'un nombre important de fonctions éparpillées et difficiles à identifier dans la hiérarchie de cette décomposition.

Approche fonctionnelle vs approche objet

- En fait, la modularité n'est pas antinomique de l'approche structurée.
Les modules résultant de la décomposition objet sont tout simplement différents de ceux émanant de l'approche structurée.
- Les unités de traitement, et surtout leur dépendance dans la topologie de la figure ci-dessus sont initialement bonnes.
C'est leur résistance au temps, contrairement aux modules objet, qui est source de problème.
- La structure d'un logiciel issue d'une approche structurée est beaucoup moins malléable, adaptable, que celle issue d'une approche objet.
- Ainsi la technologie objet est la conséquence ultime de la modularisation du logiciel, démarche qui vise à maîtriser sa production et son évolution.
Mais malgré cette continuité logique les langages objet ont apporté en pratique un profond changement dans l'art de la programmation : ils impliquent en effet un changement de l'attitude mentale du programmeur.

Concepts importants de l'approche objet

- Dans la section d'introduction, nous avons dit que l'approche objet rapproche les données et leurs traitements.
- Mais cette approche ne fait pas que ça, d'autres concepts importants sont spécifiques à cette approche et participent à la qualité du logiciel.

Notion de Classe

Tout d'abord, introduisons la notion de **classe**.

- Une classe est un type de données abstrait qui précise des caractéristiques (attributs et méthodes) communes à toute une famille d'objets et qui permet de créer (instancier) des objets possédant ces caractéristiques.
- Les autres concepts importants qu'il nous faut maintenant introduire sont l'encapsulation, l'héritage et l'agrégation.

Concepts importants de l'approche objet

Encapsulation

- L'encapsulation consiste à masquer les détails d'implémentation d'un objet, en définissant une **interface**. L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.
- L'encapsulation facilite l'évolution d'une application, car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface, et donc la façon dont l'objet est utilisé.
- L'encapsulation garantit l'intégrité des données, car elle permet d'interdire, ou de restreindre, l'accès direct aux attributs des objets

Concepts importants de l'approche objet

Héritage, spécialisation, généralisation et polymorphisme

- L'héritage est un mécanisme de **transmission des caractéristiques** d'une classe (ses attributs et méthodes) vers une sous-classe.
Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines.
- Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.
- Ainsi, la spécialisation et la généralisation permettent de construire des hiérarchies de classes.
L'héritage peut être simple ou multiple.
L'héritage évite la duplication et encourage la réutilisation.
- Le **polymorphisme** représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.
Le polymorphisme augmente la généricité, et donc la qualité, du code.

Concepts importants de l'approche objet

Aggrégation

- Il s'agit d'une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe.
Une relation d'agrégation permet donc de définir des objets composés d'autres objets.
- L'agrégation permet donc **d'assembler des objets de base**, afin de construire des objets plus complexes.

Concepts importants de l'approche objet

Historique de la programmation par Objet

- Les premiers langages de programmation qui ont utilisé des objets sont Simula I (1961-64) et Simula 67 (1967), conçus par les informaticiens norvégiens Ole-Johan Dahl et Kristan Nygaard. Simula 67 contenait déjà les objets, les classes, l'héritage, l'encapsulation, etc.
- Alan Kay, du PARC de Xerox, avait utilisé Simula dans les années 1960. Il réalisa en 1976 Smalltalk qui reste, aux yeux de certains programmeurs, le meilleur langage de programmation par objets.
- Bjarne Stroustrup a mis au point C++, une extension du langage C permettant la programmation orientée objet, aux Bell Labs d'AT&T en 1982.
- C++ deviendra le langage le plus utilisé par les programmeurs professionnels. Il arrivera à maturité en 1986, sa standardisation ANSI / ISO date de 1997.
- Java est lancé par Sun en 1995. Comme il présente plus de sécurité que C++, il deviendra le langage favori de beaucoup de programmeurs professionnels.

Introduction

- La description de la programmation par objets a fait ressortir l'étendue du travail conceptuel nécessaire : définition des classes, de leurs relations, des attributs et méthodes, des interfaces, etc.
- Pour programmer une application, il ne convient pas de se lancer tête baissée dans l'écriture du code : il faut d'abord organiser ses idées, les documenter, puis organiser la réalisation en définissant les modules et étapes de la réalisation. C'est cette démarche antérieure à l'écriture que l'on appelle *modélisation* ; son produit est un *modèle*.
- Les spécifications fournies par la maîtrise d'ouvrage en programmation impérative étaient souvent floues : les articulations conceptuelles (structures de données, algorithmes de traitement) s'exprimant dans le vocabulaire de l'informatique, le modèle devait souvent être élaboré par celle-ci.

UML

- L'approche objet permet en principe à la maîtrise d'ouvrage de s'exprimer de façon précise selon un vocabulaire qui, tout en transcrivant les besoins du métier, pourra être immédiatement compris par les informaticiens.
- En principe seulement, car la modélisation demande aux maîtrises d'ouvrage une compétence et un professionnalisme qui ne sont pas aujourd'hui répandus.

UML : HISTORIQUE

Histoire des modélisations par objet

- Les méthodes utilisées dans les années 1980 pour organiser la programmation impérative (notamment Merise) étaient fondées sur la **modélisation séparée des données et des traitements**.
- Lorsque la programmation par objets prend de l'importance au début des années 1990, la nécessité d'une méthode qui lui soit adaptée devient évidente. Plus de cinquante méthodes apparaissent entre 1990 et 1995 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE, etc.), mais aucune ne parvient à s'imposer.

En 1994, le consensus se fait autour de trois méthodes :

- **OMT** de James Rumbaugh (*General Electric*) fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système ;
- **OOD** de Grady Booch, définie pour le *Department of Defense*, introduit le concept de paquetage (*package*) ;
- **OOSE** d'Ivar Jacobson (Ericsson) fonde l'analyse sur la description des besoins des utilisateurs (cas d'utilisation, ou *use cases*).

UML : HISTORIQUE

Chaque méthode avait ses avantages et ses partisans.

Le nombre de méthodes en compétition s'était réduit, mais le risque d'un éclatement subsistait : la profession pouvait se diviser entre ces trois méthodes, créant autant de continents intellectuels qui auraient du mal à communiquer.

- Événement considérable et presque miraculeux, les trois gourous qui régnaient chacun sur l'une des trois méthodes se mirent d'accord pour définir une méthode commune qui fédérerait leurs apports respectifs (on les surnomme depuis « the Amigos »).
- UML (*Unified Modeling Language*) est né de cet effort de convergence. L'adjectif *unified* est là pour marquer qu'UML unifie, et donc remplace.
- En fait, et comme son nom l'indique, UML n'a pas l'ambition d'être exactement une méthode : c'est un **langage**.

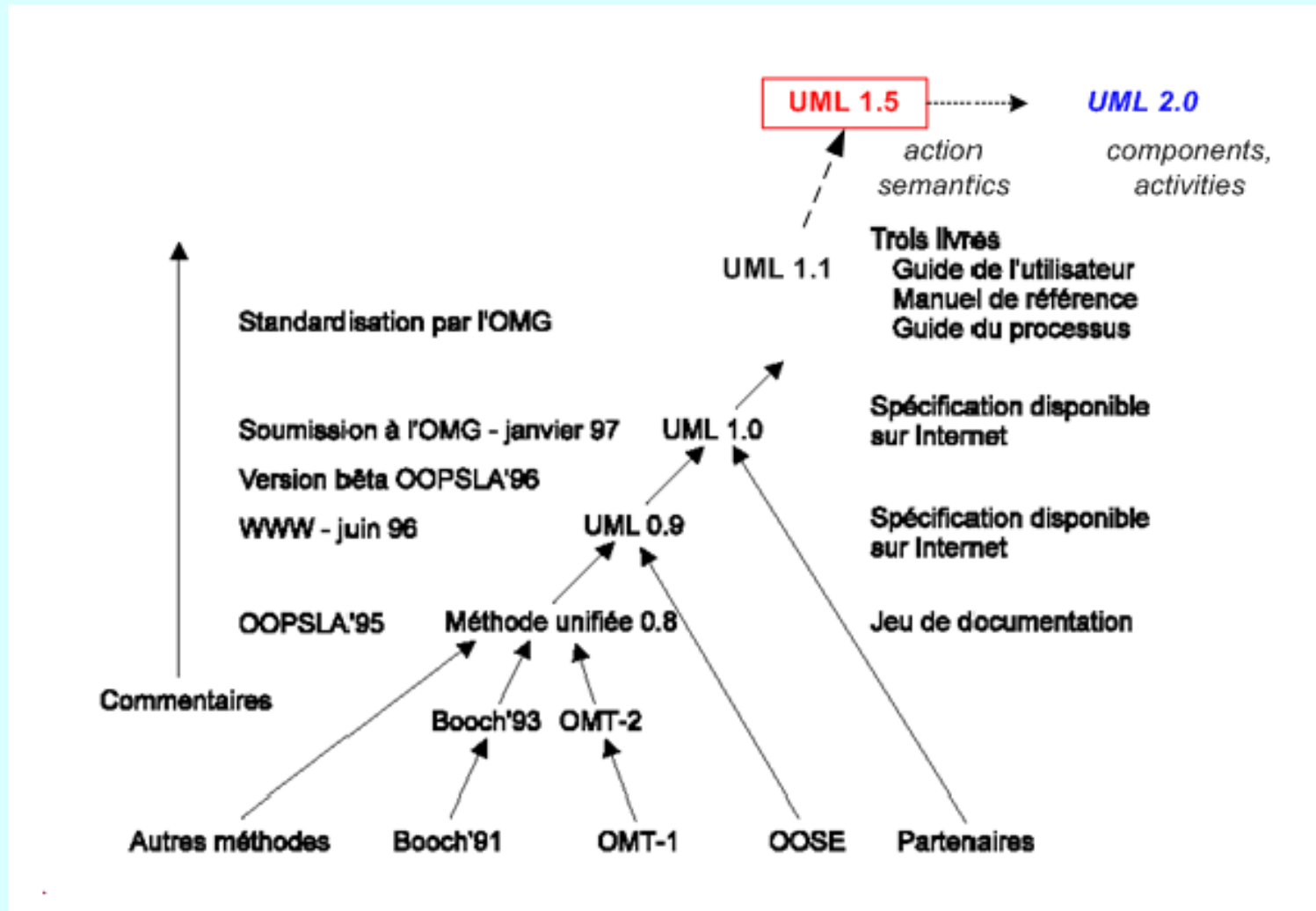
UML : HISTORIQUE

L'unification a progressé par étapes.

- En 1995, Booch et Rumbaugh (et quelques autres) se sont mis d'accord pour construire une méthode unifiée, *Unified Method 0.8* ; en 1996, Jacobson les a rejoints pour produire UML 0.9 (notez le remplacement du mot *méthode* par le mot *langage*, plus modeste).
- Les acteurs les plus importants dans le monde du logiciel s'associent alors à l'effort (IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys, etc.) et UML 1.0 est soumis à l'OMG .
- L'OMG adopte en novembre 1997 UML 1.1 comme langage de modélisation des systèmes d'information à objets. La version d'UML en cours en 2008 est UML 2.1.1, aujourd'hui UML 2.2 et les travaux d'amélioration se poursuivent.
- UML est donc non seulement un outil intéressant, mais une norme qui s'impose en technologie à objets et à laquelle se sont rangés tous les grands acteurs du domaine, acteurs qui ont d'ailleurs contribué à son élaboration.

.

UML : HISTORIQUE



UML : HISTORIQUE

- L'OMG (*Object Management Group*) est une association américaine à but non lucratif créée en 1989 dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes.

L'OMG est notamment à la base des spécifications UML, MOF, CORBA et IDL.

L'OMG est aussi à l'origine de la recommandation MDA

UML : Un Vocabulaire

- Définitions:
 - **L'événement** est quelque chose que l'entreprise veut prendre en compte et qui aura des conséquences sur les activités effectuées et les informations gérées.
 - **L'acteur** est un être humain ou un dispositif mécanique qui manipule les informations (création, transformation), ou qui est sollicité par celles-ci (leur existence, ou leur valeur).
 - **Le processus** est un plan d'ensemble (sur la base des informations gérées) indiquant comment les acteurs collaborent pour traiter une catégorie d'évènements.
- UML = Une nouvelle approche de la complexité.

UML : Un Ensemble de Modèles

Définition

UML met à disposition un ensemble d'outils conceptuels (à base essentiellement de diagrammes) pour exprimer et décrire les process de l'entreprise.

UML : Ce que l'on peut en attendre

- UML **n'est pas une méthode** (*i.e.* une description normative des étapes de la modélisation) : ses auteurs ont en effet estimé qu'il n'était pas opportun de définir une méthode en raison de la diversité des cas particuliers.
- Ils ont préféré se borner à définir un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système d'information. Aux graphiques sont bien sûr associés des textes qui expliquent leur contenu.
- UML est donc un **métalangage**, car il fournit les éléments permettant de construire le modèle qui, lui, sera le langage du projet.
- Il est impossible de donner une représentation graphique complète d'un logiciel, ou de tout autre système complexe, de même qu'il est impossible de représenter entièrement une statue (à trois dimensions) par des photographies (à deux dimensions).

Mais il est possible de donner sur un tel système des *vues* partielles, analogues chacune à une photographie d'une statue, et dont la conjonction donnera une idée utilisable en pratique sans risque d'erreur grave.

UML : Ce que l'on peut en attendre

UML 2.0 comporte ainsi treize types de diagrammes représentant autant de *vues* distinctes pour représenter des concepts particuliers du système d'information. Ils se répartissent en deux grands groupes :

Diagrammes structurels ou diagrammes statiques (*UML Structure*)

- Diagramme de classes (*Class diagram*)
- Diagramme d'objets (*Object diagram*)
- Diagramme de composants (*Component diagram*)
- Diagramme de déploiement (*Deployment diagram*)
- Diagramme de paquetages (*Package diagram*)
- Diagramme de structures composites (*Composite structure diagram*)

UML : Ce que l'on peut en attendre

Diagrammes comportementaux ou diagrammes dynamiques (*UML Behavior*)

- Diagramme de cas d'utilisation (*Use case diagram*)
- Diagramme d'activités (*Activity diagram*)
- Diagramme d'états-transitions (*State machine diagram*)

Diagrammes d'interaction (*Interaction diagram*)

- Diagramme de séquence (*Sequence diagram*)
- Diagramme de communication (*Communication diagram*)
- Diagramme global d'interaction (*Interaction overview diagram*)
- Diagramme de temps (*Timing diagram*)

UML : Ce que l'on peut en attendre

Ces diagrammes, d'une utilité variable selon les cas, ne sont pas nécessairement tous produits à l'occasion d'une modélisation.

Beaucoup n'ont qu'une utilisation anecdotique, ou très spécifique.

Les plus utiles pour la maîtrise d'ouvrage sont les **diagrammes d'activités, de cas d'utilisation, de classes, d'objets, de séquence et d'états-transitions**.

Les diagrammes **de composants, de déploiement et de communication** sont surtout utiles pour la maîtrise d'œuvre à qui ils permettent de formaliser les contraintes de la réalisation et la solution technique

Ce sont sur ces diagrammes que le présent cours, cours d'initiation à UML, a été construit.

Synthèse

UML en pratique (avec illustration de quelques diagrammes)

UML en pratique

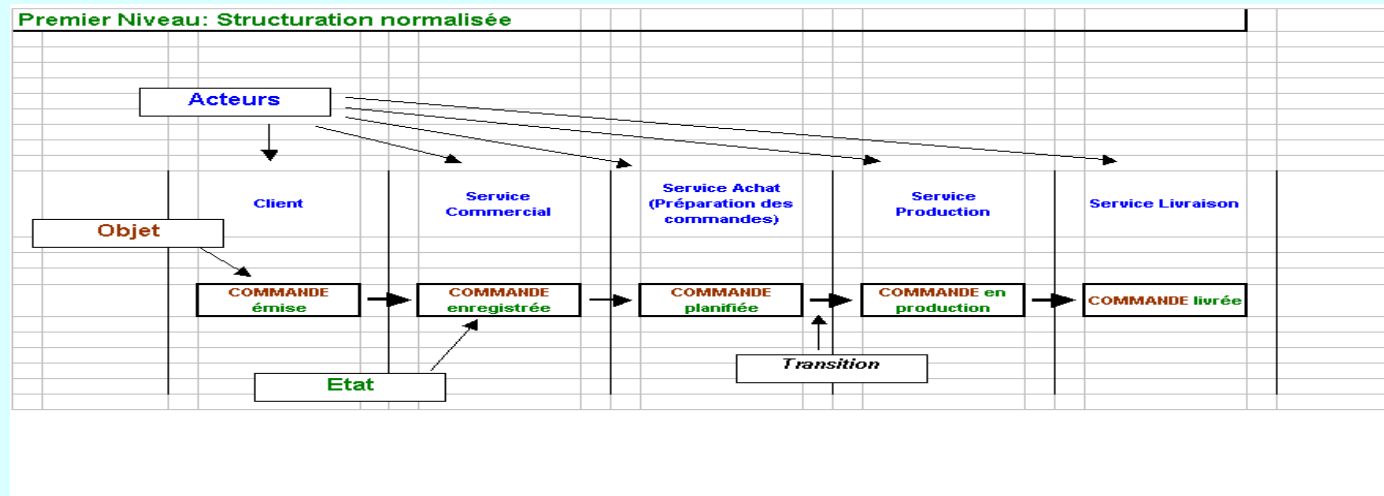
- UML « difficile d'accès ».
 - UML propose, mais n'impose pas.
 - UML s'applique dans toutes les phases du projet.
 - UML est un langage riche.
- Le succès d'UML.
 - Accès gratuit aux spécifications.
 - Adhésion des développeurs.
 - Capacité à enrichir la démarche de conception.
- OMG et l'approche MDA (Model Driven Architecture).
 - Lancer la modélisation dès le début du cycle.

Les Concepts Essentiels en résumé

- **Approche « intuitive ».**
- **Le concept d'Objet.**
- **Les Acteurs.**
- **Les Processus.**
- **Le Workflow**

Approche « Intuitive »

- Exemple: Traitement d'une commande
 - Identification des Acteurs
 - La commande, concept unique
 - Objet, États, Transitions, Évènements
 - Couloirs d'Activités, Diagrammes d'Activités



Le concept d'Objet

- Les Principes
 - **Le principe de distinction**
 - **Le principe de permanence**
 - **Le principe d'activité**
 - **Applications du concept**

Le concept d'Objet

- CLASSES
 - Objets qui nous intéressent
 - L'organisation des Objets en catégories
 - Détermination d'une Classe
 - La taxonomie des Classes

Le concept d'Objet

- ENTITES
 - Les types d'Entités
 - Entités de gestion
 - Entités de référence
 - Entités de reporting
 - La description du Système d'Information

Le concept d'Acteurs

- Définition
- Typologie

Le concept de Processus

- **Définition**
- **Notion d'Évènement**
- **Typologie**
- **Domaine**
- **Variante et Scénario**

Le Workflow

- Les diagrammes d'Activités
- Les diagrammes d'États-Transitions
- Les diagrammes d'Interaction

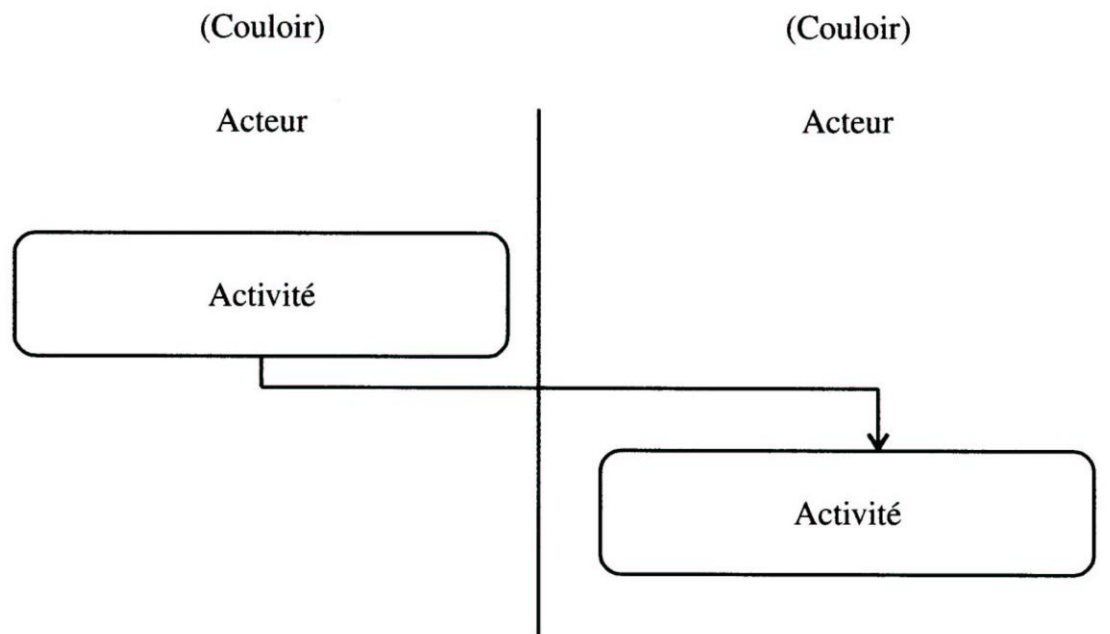
Avec en introduction :

- Les diagrammes de Cas d'Utilisation

Les principaux schémas de synthèse

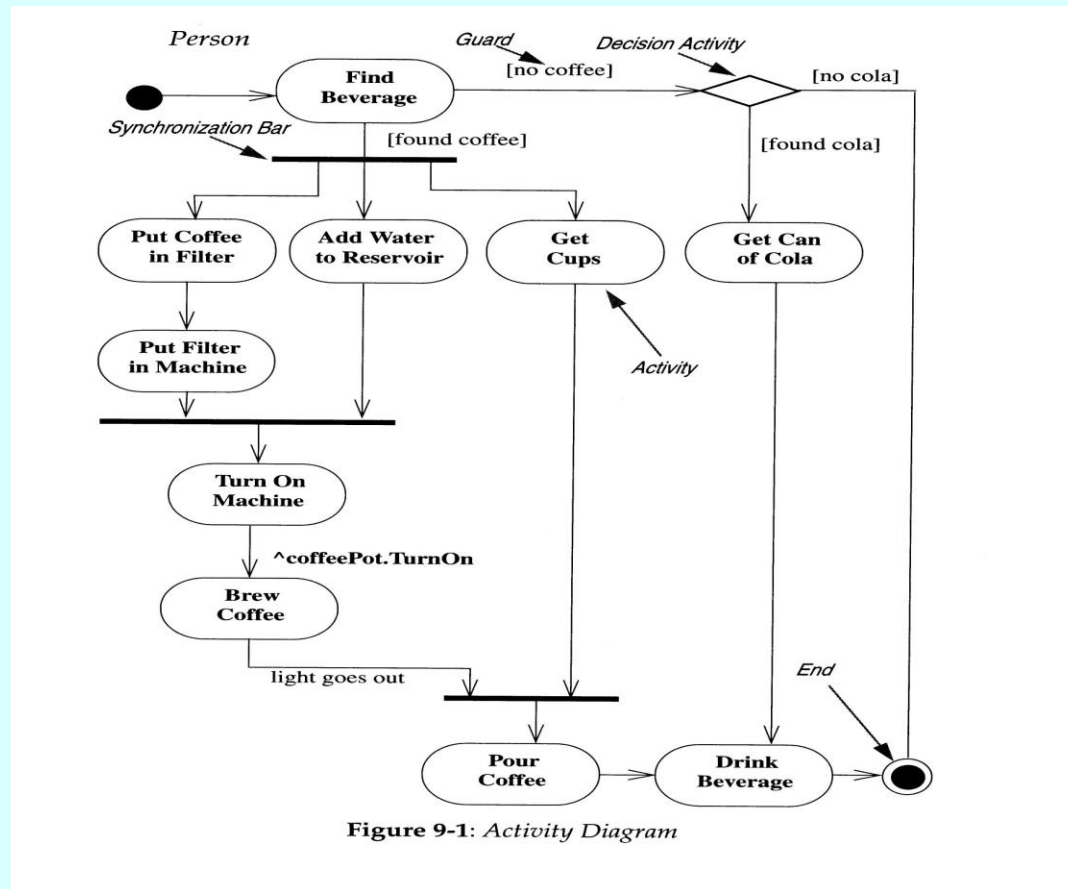
- Diagramme d'Activité

Formalisme



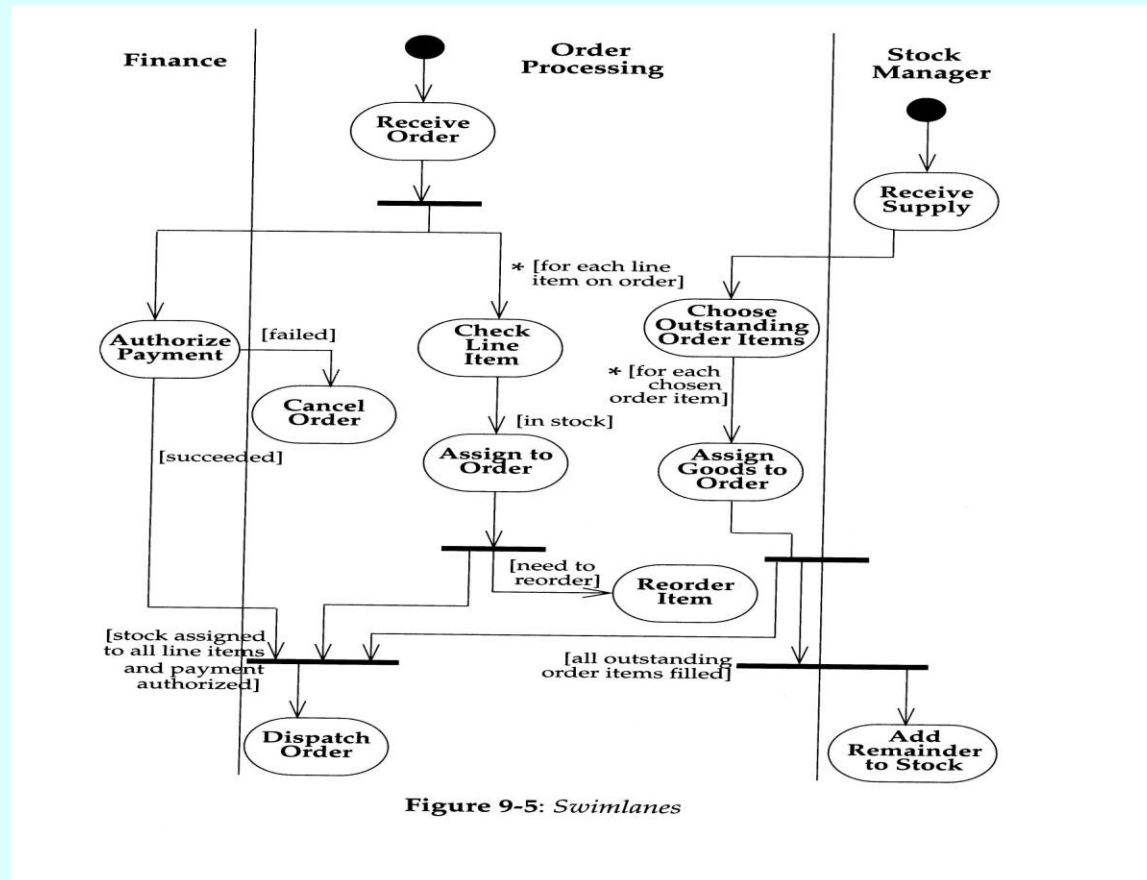
Schémas de synthèse

- Diagramme d'Activité



Schémas de synthèse

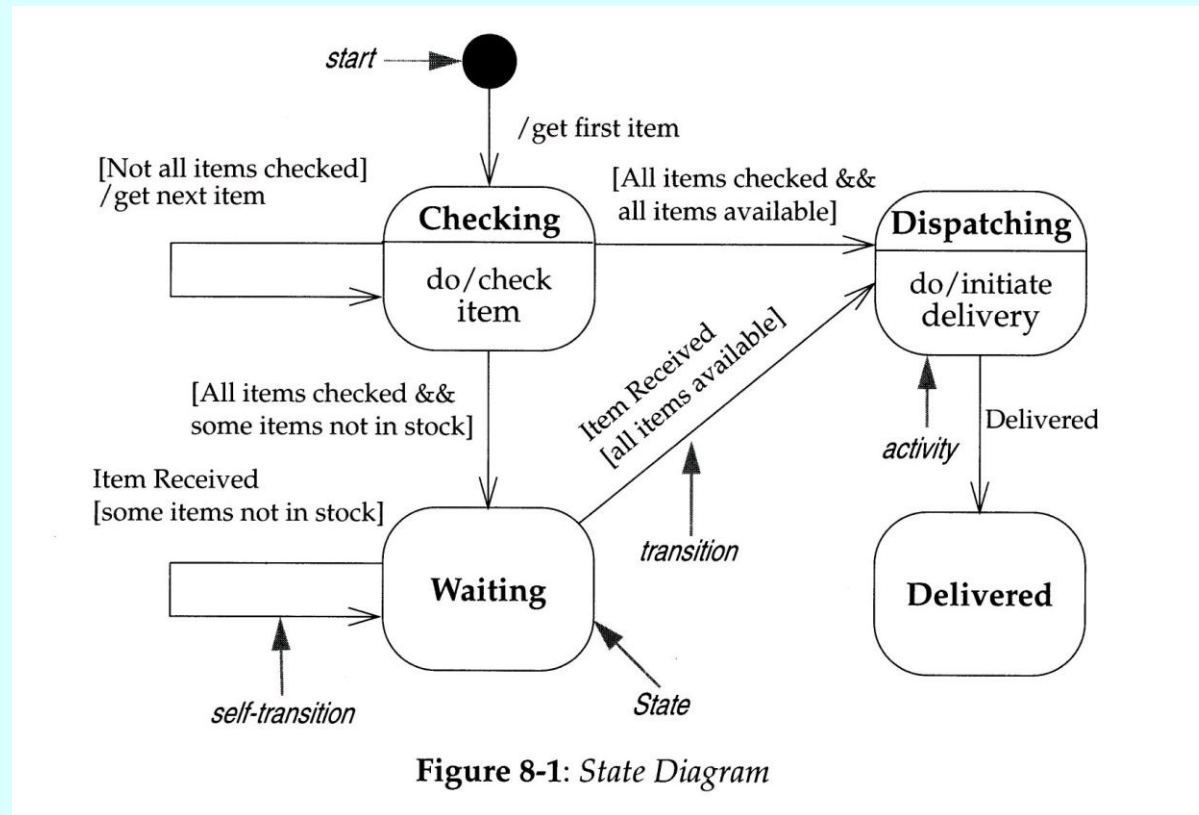
- Diagramme d'Activité avec partitions



Schémas de synthèse

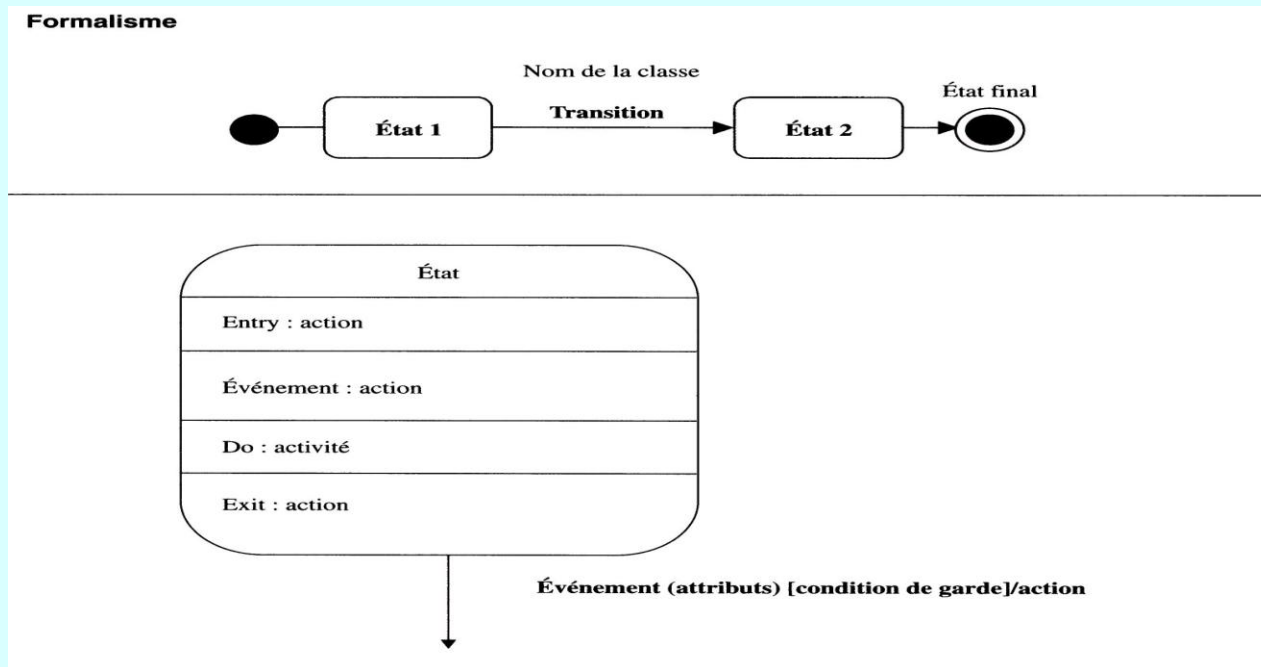
- Diagrammes d'Interaction

Diagramme de communication (également : diagramme de séquence)



Schémas de synthèse

- **Diagramme d'Etats-Transition**
 - L' « Etat » et la « Transition »
 - La hiérarchie des Etats
 - Etats Prédéfinis
 - Utilisation des diagrammes



Schémas de synthèse

- Diagramme d'Etats-Transition

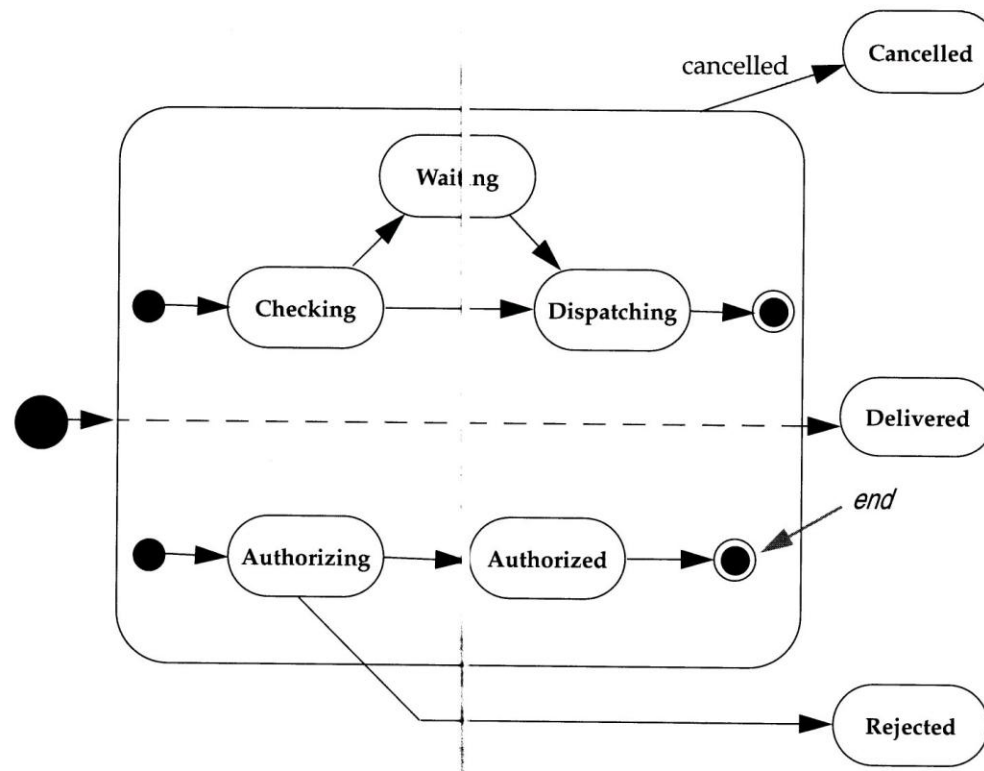


Figure 8-5: Concurrent State Diagram

Schémas de synthèse : En résumé

Diagramme des Cas d'Utilisation

- Le diagramme de cas d'utilisation représente la structure des grandes fonctionnalités nécessaires aux utilisateurs du système.
- C'est le premier diagramme du modèle UML, celui où s'assure la relation entre l'utilisateur et les objets que le système met en œuvre.

Diagramme d'Activités

- Le diagramme d'activités n'est autre que la transcription dans UML de la représentation du processus telle qu'elle a été élaborée lors du travail qui a préparé la modélisation :
- il montre l'enchaînement des activités qui concourent au processus.

Schémas de synthèse : En résumé

Diagramme de Classe

- Le diagramme de classes est généralement considéré comme le plus important dans un développement orienté objet.
Il représente l'architecture conceptuelle du système :
Il décrit les classes que le système utilise, ainsi que leurs liens, que ceux-ci représentent un emboîtement conceptuel (héritage) ou une relation organique (agrégation).

Diagramme d'Objets

- Le diagramme d'objets permet d'éclairer un diagramme de classes en l'illustrant par des exemples.
- Il est, par exemple, utilisé pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles.

Diagramme d'Etats-Transitions

- Le diagramme d'états-transitions représente la façon dont évoluent (*i.e.* cycle de vie) les objets appartenant à une même classe.
- La modélisation du cycle de vie est essentielle pour représenter et mettre en forme la dynamique du système.

Schémas de synthèse : En résumé

Diagramme de Séquence et de Communication (interaction)

- Le diagramme de séquence représente la succession chronologique des opérations réalisées par un acteur.
- Il indique les objets que l'acteur va manipuler et les opérations qui font passer d'un objet à l'autre.
- On peut représenter les mêmes opérations par un diagramme de communication , graphe dont les nœuds sont des objets et les arcs (numérotés selon la chronologie) les échanges entre objets.
- En fait, diagramme de séquence et diagramme de communication sont deux vues différentes, mais logiquement équivalentes (on peut construire l'une à partir de l'autre) d'une même chronologie.
- Ce sont des diagrammes d'interaction.

Présentation du modèle

Comment présenter un modèle UML

- La présentation d'un modèle UML se compose de plusieurs documents écrits en langage courant et d'un document formalisé : elle ne doit pas se limiter au seul document formalisé, car celui-ci est pratiquement incompréhensible si on le présente seul.
- Un expert en UML sera capable dans certains cas de reconstituer les intentions initiales en lisant le modèle, mais pas toujours ; et les experts en UML sont rares.

Présentation du modèle

Voici la liste des documents qui paraissent nécessaires :

Présentation stratégique :

- elle décrit pourquoi l'entreprise a voulu se doter de l'outil considéré, les buts qu'elle cherche à atteindre, le calendrier de réalisation prévu, etc. ;

Présentation des processus de travail par lesquels la stratégie entend se réaliser :

- pour permettre au lecteur de voir comment l'application va fonctionner en pratique, elle doit être illustrée par une esquisse des écrans qui seront affichés devant les utilisateurs de terrain ;

Explication des choix qui ont guidé la modélisation formelle :

- il s'agit de synthétiser, sous les yeux du lecteur, les discussions qui ont présidé à ces choix ;

Présentation du modèle

Modèle formel :

- C'est le document le plus épais et le plus difficile à lire. Il est préférable de le présenter sur l'Intranet de l'entreprise.

En effet, les diagrammes peuvent être alors équipés de liens hypertextes permettant l'ouverture de diagrammes plus détaillés ou de commentaires.

On doit présenter en premier le **diagramme de cas d'utilisation** qui montre l'enchaînement des cas d'utilisation au sein du processus, enchaînement immédiatement compréhensible ;

- Puis le **diagramme d'activités**, qui montre le contenu de chaque cas d'utilisation ;
- Puis le **diagramme de séquence**, qui montre l'enchaînement chronologique des opérations à l'intérieur de chaque cas d'utilisation.
- Enfin, le **diagramme de classes**, qui est le plus précis conceptuellement, mais aussi le plus difficile à lire, car il présente chacune des classes et leurs relations (agrégation, héritage, association, etc.).