

الكلية متعددة التخصصات - ورزازات
+٥٢٤٣٦٠١٧ +٥٢٤٣٦٠٥٠
FACULTÉ POLYDISCIPLINAIRE DE OUARZAZATE



Génie logiciel

Sciences Mathématiques et informatique
S6

Professeur : Abdelhadi Bouain

Introduction

- Un système informatique:
 - Matériel + logiciel
- Un système d'information:
 - Le système d'information est un ensemble organisé de ressources (matériel, logiciel, humain) qui permet de collecter, stocker, traiter et distribuer de l'information au sein d'une organisation.
 - Le système d'information vise à s'aligner avec la stratégie métier (processus métiers) de l'entreprise.

Introduction

- Systèmes informatiques :
 - 80 % de logiciel
 - 20 % de matériel
- Depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement.
 - Le matériel est relativement fiable.
 - Le marché du matériel est standardisé.

Les problèmes liés à l'informatique sont essentiellement des problèmes de logiciel.

Introduction

- La crise du logiciel (fin des années 1960)
- Exemples:
 - L'abandon du projet d'informatisation des opérations boursières « la Bourse de Londres ». Projet Taurus (lancé en 1981). Plus de 100 millions de livres (**1 252 298 027 dh**).
 - bug de l'an 2000.
 - 1ère sonde Mariner vers Vénus : perdue dans l'espace (erreur Fortran).
 - Etc.

Introduction

- Étude sur 8 380 projets (Standish Group, 1995) :
 - Succès : 16 %;
 - Problématique : 53 % (budget ou délais non respectés, défaut de fonctionnalités) ;
 - Échec : 31 % (abandonné).

Le taux de succès décroît avec la taille des projets et la taille des entreprises.

- Comment faire des logiciels de qualité ?
- Qu'attend-on d'un logiciel ? Quels sont les critères de qualité ?

Naissance du « Génie Logiciel »

- P. Naur and B. Randell - La conférence du NATO « OTAN » 1968.
- Naissance du « *Génie Logiciel* » (software engineering).

« Le génie logiciel cherche à définir et à appliquer des méthodes, des outils et des pratiques propres à assurer la production de logiciel répondant à des besoins spécifiés et respectant certains critères de qualité, ainsi que les délais et les coûts »

Naissance du « Génie Logiciel »

- Les qualités d'un logiciel : (La norme **ISO/IEC 25010:2011**)
 - **La capacité fonctionnelle:** respecter les spécifications et surtout répondre aux attentes de l'utilisateur + l'interopérabilité + la sécurité.
 - **Fiabilité :** la capacité d'un logiciel de rendre des résultats corrects quelles que soient les conditions d'exploitation.
 - **Facilité d'utilisation.**
 - **Performance:** (temps de réponse, consommation des ressources,...)
 - **Maintenabilité:** (peu d'effort nécessaire pour y ajouter de nouvelles fonctions)
 - **La portabilité.**

Méthodologies de développement des logiciels

Le cycle de vie d'un logiciel

- Comme pour tout produit manufacturé complexe :
 - on **décompose** la production en « **phases** »
 - l'ensemble des phases constitue un « **cycle de vie** »

Le cycle de vie d'un logiciel

- Définition des besoins
- Analyse et spécification des besoins
- Planification
- Conception
- Programmation
- Intégration
- Vérification et validation
- Maintenance

Le cycle de vie d'un logiciel : Analyse des besoins

● Besoins fonctionnels

- Description des services(fonctions).
- Description des données manipulées
- "Comment souhaite-on pouvoir utiliser le système".

● Besoins non fonctionnels

- Description des contraintes
- Pour chaque service et pour le système global, il est possible d'exprimer différents types de contraintes:
- contraintes de performance
- contraintes de sécurité
- Contrainte de convivialité et d'apparence
- Etc.

Le cycle de vie d'un logiciel : Analyse des besoins

- Comment identifier les besoins ?
 - Entretiens, questionnaires.
 - Observation de l'existant (documents , logiciels).
 - Etude de situations similaires.
- Définir Chaque besoin (objectif) en s'assurant qu'il soit (SMART) :
 - Spécifique (clairement défini),
 - Mesurable (chiffrable),
 - Atteignable (en tenant compte des ressources nécessaires disponibles par exemple),
 - Réaliste (pertinent),
 - Temporellement défini (en se fixant une deadline).

Le cycle de vie d'un logiciel : Analyse des besoins

- A l'issue de cette phase : cahier des charges.
- Un cahier des charges est normalement *établi par le client* en interaction avec utilisateurs et encadrement:
 - Besoins fonctionnels.
 - Besoins non fonctionnels .

→ Possibilité d'utilisation des Use Case « Cas d'utilisation UML »

Le cycle de vie d'un logiciel : Cahier de charges

Cahier des charges fonctionnel

(Normes: AFNOR NF X50-151 et NF EN 16271)

- **1- Présentation générale du problème**

- **1.1 Projet**

- Présentation du projet, de ses finalités, de son possible retour sur investissement.

- **1.2 Contexte**

- Situation générale de l'organisation, autres projets en cours, études éventuelles menées sur le sujet (ou des sujets similaires), prestations attendues, parties concernées par le projet, confidentialité...

- **1.3 Énoncé du besoin**

- Services rendus par le produit pour l'utilisateur final.

- **1.4 Environnement du produit**

- Personnes, équipements, matériaux, contraintes de l'environnement, caractéristiques de chaque élément.

Le cycle de vie d'un logiciel : Cahier de charges

Cahier des charges fonctionnel

(Normes: AFNOR NF X50-151 et NF EN 16271)

- **2- Expression fonctionnelle du besoin**

- **2.1 Fonctions de service et de contrainte**

Fonctions de services principales, fonctions complémentaires (améliorant ou simplifiant le service rendu), contraintes pouvant entraver la liberté du réalisateur du produit.

- **2.2 Critères d'appréciation**

Souligner les critères déterminants pour l'évaluation des propositions.

- **2.3 Niveaux des critères d'appréciation et ce qui les caractérise**

Bien définir, d'une part, les niveaux indispensables et, d'autre part, les niveaux voulus mais révisables.

Le cycle de vie d'un logiciel : Cahier de charges

Cahier des charges fonctionnel

(Normes: AFNOR NF X50-151 et NF EN 16271)

- **3- Cadre de réponse**

Cette partie encadre la manière dont les réponses au besoin devront être formulées.

- **3.1 Pour chaque fonction**

Solution proposée par le fournisseur, niveau atteint pour chaque critère d'appréciation, modalités de contrôle., part du prix attribué à chaque fonction.

- **3.2 Pour l'ensemble du produit**

Définir le prix de la réalisation de base, lister les options et variantes éventuellement proposées, les mesures prises pour se soumettre aux contraintes et leur impact économique, les outils liés à la mise en place et à la maintenance, décomposition en sous-ensembles, prévisions et perspectives de fiabilité et d'évolution.

* Exemple de Plan-type (d'après la norme AFNOR X50-151)
Cahier des charges fonctionnel

1. Présentation générale du problème

1.1 Projet

1.1.1 Finalités

1.1.2 Espérance de retour sur investissement

1.2 Contexte

1.2.1 Situation du projet par rapport aux autres projets de l'entreprise

1.2.2 Études déjà effectuées

1.2.3 Études menées sur des sujets voisins

1.2.4 Suites prévues

1.2.5 Nature des prestations demandées

1.2.6 Parties concernées par le déroulement du projet et ses résultats (demandeurs, utilisateurs)

1.2.7 Caractère confidentiel s'il y a lieu

1.3 Enoncé du besoin (finalités du produit pour le futur utilisateur tel que prévu par le demandeur)

1.4 Environnement du produit recherché

1.4.1 Listes exhaustives des éléments (personnes, équipements, matières...) et contraintes (environnement)

1.4.2 Caractéristiques pour chaque élément de l'environnement

2. Expression fonctionnelle du besoin

2.1 Fonctions de service et de contrainte

2.1.1 Fonctions de service principales (qui sont la raison d'être du produit)

2.1.2 Fonctions de service complémentaires (qui améliorent, facilitent ou complètent le service rendu)

2.1.3 Contraintes (limitations à la liberté du concepteur-réalisateur)

2.2 Critères d'appréciation (en soulignant ceux qui sont déterminants pour l'évaluation des réponses)

2.3 Niveaux des critères d'appréciation et ce qui les caractérise

2.3.1 Niveaux dont l'obtention est imposée

2.3.2 Niveaux souhaités mais révisables

3. Cadre de réponse

3.1 Pour chaque fonction

3.1.1 Solution proposée

3.1.2 Niveau atteint pour chaque critère d'appréciation de cette fonction et modalités de contrôle

3.1.3 Part du prix attribué à chaque fonction

3.2 Pour l'ensemble du produit

3.2.1 Prix de la réalisation de la version de base

3.2.2 Options et variantes proposées non retenues au cahier des charges

3.2.3 Mesures prises pour respecter les contraintes et leurs conséquences économiques

3.2.4 Outils d'installation, de maintenance ... à prévoir

3.2.5 Décomposition en modules, sous-ensembles

3.2.6 Prévisions de fiabilité

3.2.7 Perspectives d'évolution technologique

Le cycle de vie d'un logiciel : Cahier de charges

➤ **Cahier des charges technique:**

Les éléments techniques nécessaires à mettre en œuvre pour réaliser le logiciel.

- **Quelques exemples :**

- les moyens de paiement en ligne,
- la solution d'hébergement,
- l'architecture des serveurs,
- le choix de la plateforme ou du CMS,
- les outils d'administration,
- les contraintes d'intégration,
- le langage informatique,
- la gestion de la sécurité des données,
- la maintenance,
- la migration,
- la compatibilité avec les navigateurs, etc.

Le cycle de vie d'un logiciel : Cahier de charges

➤ La charte graphique

- La charte graphique est un document de travail qui contient l'ensemble des règles fondamentales d'utilisation des signes graphiques qui constituent l'identité graphique d'une organisation, d'un projet, d'une entreprise.
- Le but de la charte graphique est de conserver une cohérence graphique dans les réalisations graphiques d'une même organisation, projet ou entreprise quels que soient les différents intervenants de la production (graphiste, directeur artistique...).
- Exemples d'éléments à spécifier :
 - le logo,
 - la typographie (police, taille, etc.),
 - les couleurs,
 - les illustrations, etc.

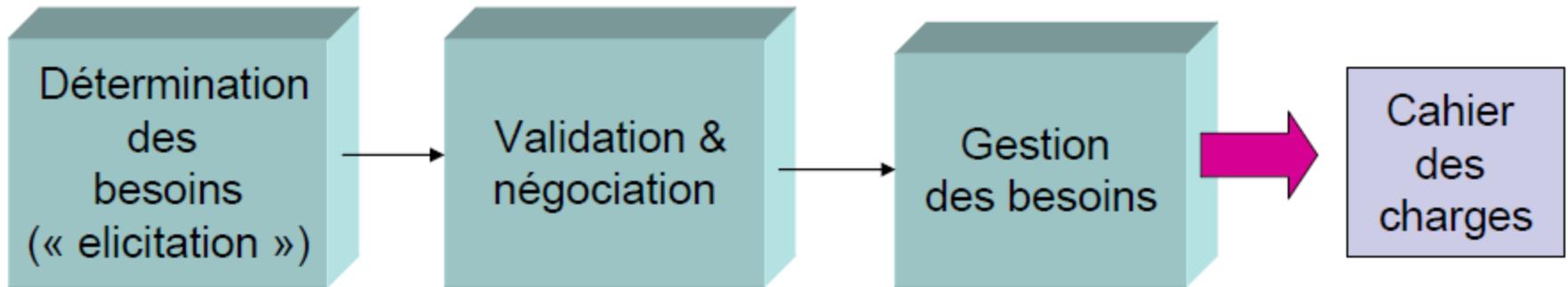
Le cycle de vie d'un logiciel : Cahier de charges

Autres informations :

- **Planning préliminaire**
- **Budget préliminaire**
- **Glossaire**
- **Documents et formulaires d'entreprise**
- **Références bibliographiques**

Le cycle de vie d'un logiciel : Analyse des besoins

A. Expression des besoins



B. Analyse et spécification



Le cycle de vie d'un logiciel : La spécification

➤ La spécification des besoins:

- La spécification aura pour but de décrire avec rigueur:
 - Les données du systèmes (**vue statique**).
 - Les fonctions du système (**vue fonctionnelle**).
 - Les changements d'états et le contrôle du système (**vue comportementale**).

Le cycle de vie d'un logiciel : La spécification

➤ La spécification:

- Schémas.
- Langages formels.
- **La Spécifications souvent incompréhensibles pour les non initiés.**

Le cycle de vie d'un logiciel : La spécification

- **La spécification des besoins:**
- ❖ **Participants:** analyste
- ❖ **Document (résultat) :** *dossier d'analyse et de spécification*
 - Notation graphique ou textuelle rigoureuse.
 - Rédigé par: l'analyste.
 - Découpage: modèles statique, fonctionnel et comportemental

Le cycle de vie d'un logiciel : La spécification

- **La spécification des besoins:** Degré de formalisme.
- Spécifications **informelles**:
 - Ex. langue naturelle, croquis, etc.
- Spécifications **semi-formelles**:
 - Notation graphique dont la sémantique n'est pas sémantique pas absolument précise. Ex. UML, Merise, etc.
- Spécifications **formelles**:
 - Ex.: Spéc. algébriques, spéc. logiques, réseaux de Petri, automates, méthode Z, Méthode B, etc.

Le cycle de vie d'un logiciel : La spécification

➤ La spécification des besoins: Spécification semi-formelle.

- SADT,
- SA-RT
- SSADM,
- JSD
- JSP
- ***Merise***
- Axial
- OOA, OMT
- ***UML***

Le cycle de vie d'un logiciel : La spécification

- **La spécification des besoins:** Spécification semi-formelle.
 - Il permet de couvrir les aspects du système (statique, dynamique, comportemental)
 - Il est facile à utiliser.
 - **MAIS** Il est impossible de raisonner/analyser formellement sur le système en vue.

Le cycle de vie d'un logiciel : La spécification

- **La spécification des besoins: Spécification formelle.**
- « **La rédaction des spécifications de l'application doit être réalisée dans un langage rigoureux ne donnant pas prise aux interprétations** ».
- Expression dans un langage formel du quoi d'un système à développer.
- Plusieurs formes possibles selon la nature du système.
- Langages ou formalismes de spécification formelle : Logique, Z, B, Langages de spécification algébriques, algèbres de processus, etc.

Le cycle de vie d'un logiciel : La spécification

- **La spécification des besoins:** **Spécification formelle.**
- On reconnaît trois axes mathématiques principaux qui servent de supports aux méthodes formelles :
 - La logique du premier ordre.
 - La théorie des ensembles.
 - La théorie des types.

Le cycle de vie d'un logiciel : La spécification

- **La spécification des besoins: Spécification formelle.** Base mathématiques.
 - ❖ **Logique du premier ordre (logique des prédictats)**
- La logique du premier est relativement simple et permet d'exprimer les propriétés de nombreux objets informatiques.
- Afin de désigner les objets de la spécification, la logique utilise des symboles d'individus ***tels*** que ***téléphone*** ou bien ***Caroline***.
- La logique utilise également des symboles de fonctions comme numéro : ***numéro (x)*** servirait à désigner le numéro de téléphone d'une personne x.
- Enfin, elle utilise des symboles de prédictats servant à exprimer des propriétés des objets. Ainsi, on peut choisir que le prédictat ***est_abonné (x)*** signifie que la personne x est un abonné du téléphone.

Le cycle de vie d'un logiciel : La spécification

- **La spécification des besoins: Spécification formelle.** Base mathématiques.
 - ❖ **Logique du premier ordre (logique des prédictats)**
- L'utilisateur a donc toute latitude concernant le vocabulaire qui servira à exprimer les formules élémentaires.
- Celles-ci peuvent ensuite être combinées au moyen de connecteurs qui eux sont bien définis :
 \wedge pour ET, \vee pour OU, \neg pour NON, \Rightarrow pour «implique » ;
 \forall pour « quel que soit », \exists pour « il existe ».

Le cycle de vie d'un logiciel : La spécification

➤ La spécification des besoins: Spécification formelle. Base mathématiques.

Exemple :

Supposons que nous avons une exigence logicielle qui indique : "***Le système ne doit autoriser que les utilisateurs âgés de plus de 18 ans à créer un compte.***"

- Nous pouvons exprimer cette exigence en utilisant la logique prédicative comme suit :
- Soit U l'ensemble de tous les utilisateurs et age(x) un prédicat qui retourne l'âge de l'utilisateur x.
- Nous pouvons définir un autre prédicat, createAccount(x), qui renvoie vrai si l'utilisateur x peut créer un compte, et faux sinon.
- Maintenant, nous pouvons exprimer l'exigence sous forme d'une formule logique : $\forall x \in U, \text{age}(x) \geq 18 \rightarrow \text{createAccount}(x)$
- Cette formule se lit comme suit : "Pour tous les utilisateurs x dans U, si l'âge de x est supérieur ou égal à 18, alors x doit pouvoir créer un compte."
- Cette formule logique peut être utilisée pour vérifier que l'implémentation logicielle respecte correctement l'exigence, et peut également aider à concevoir des cas de test pour s'assurer que l'exigence est satisfaite.

Le cycle de vie d'un logiciel : La spécification

- **La spécification des besoins: Spécification formelle.** Base mathématiques.
- **Exemple:** Traduisez les énoncés suivants en formules de la logique des prédicats (par exemple $Aime(x,y) = x$ aime y).
 - a. Jean est plus grand que Marie
 - b. Paul a vu Léa et elle ne l'a pas vu
 - c. Si Jean est un homme, alors il est mortel
 - d. Un chat est entré
 - e. Certains enfants ne sont pas malades
 - f. Tous les éléphants ont une trompe
 - g. Tous les hommes n'aiment pas Marie
 - h. Il y a une chanson qu'aucun enfant ne chante
 - i. Si tous les hommes aiment Marie, alors elle est contente
 - j. Tous les étudiants apprécient un professeur

Le cycle de vie d'un logiciel : La spécification

➤ La spécification des besoins: Spécification formelle. Base mathématiques.

- a. Jean est plus grand que Marie
- b. Paul a vu Léa et elle ne l'a pas vu
- c. Si Jean est un homme, alors il est mortel
- d. Un chat est entré
- e. Certains enfants ne sont pas malades
- f. Tous les éléphants ont une trompe
- g. Tous les hommes n'aiment pas Marie

- h. Il y a une chanson qu'aucun enfant ne chante

OU

- i. Si tous les hommes aiment Marie, alors elle est contente

$$\begin{aligned} & G(j, m) \\ & V(p, l) \wedge \neg V(l, p) \\ & H(j) \rightarrow M(j) \\ & \exists x(C(x) \wedge E(x)) \\ & \exists x(E(x) \wedge \neg M(x)) \\ & \forall x(E(x) \rightarrow T(x)) \\ & \forall x(H(x) \rightarrow \neg A(x, m)) \\ & \neq \neg \forall x(H(x) \rightarrow A(x, m)) \\ & \exists x \forall y((C(x) \wedge E(y)) \rightarrow \neg C(y, x)) \\ & = \exists x \neg \exists y(C(x) \wedge E(y) \wedge C(y, x)) \\ \\ & (\forall x(H(x) \rightarrow A(x, m)) \rightarrow C(m)) \end{aligned}$$

- j. Tous les étudiants apprécient un professeur

$$\forall x \exists y ((E(x) \wedge P(y)) \rightarrow A(x, y))$$

Le cycle de vie d'un logiciel : La spécification

➤ La spécification des besoins: Spécification formelle. Base mathématiques.

j. Tous les étudiants apprécient un professeur

1. Interprétation 1 : Il existe un professeur que tous les étudiants apprécient.

$$\exists p (\text{Professeur}(p) \wedge \forall e (\text{Étudiant}(e) \rightarrow \text{Apprécie}(e, p)))$$

Signification : Il existe un professeur p tel que pour tout étudiant e , e apprécie p .

2. Interprétation 2 : Chaque étudiant apprécie au moins un professeur (pas forcément le même).

$$\forall e (\text{Étudiant}(e) \rightarrow \exists p (\text{Professeur}(p) \wedge \text{Apprécie}(e, p)))$$

Signification : Pour tout étudiant e , il existe un professeur p que e apprécie.

Le cycle de vie d'un logiciel : La spécification

➤ La spécification des besoins: Spécification formelle. Base mathématiques.

Exercice : Traduire dans le langage des prédictats du premier ordre les phrases suivantes :

- Tous les hommes sont méchants.
- Seulement les hommes sont méchants.
- Il existe des hommes méchants.
- Il existe un homme qui n'est pas méchant.
- Il n'existe pas d'homme méchant.
- Il existe un homme qui aime toutes les femmes.
- Chaque chat connaît un chien qui le déteste.
- Tous les poissons, sauf les requins, sont gentils avec les enfants.
- Tous les oiseaux ne peuvent pas voler.
- Chaque personne aime quelqu'un et personne n'aime tout le monde, ou bien quelqu'un aime tout le monde et quelqu'un n'aime personne.
- Il y a des gens que l'on peut rouler tout le temps et quelquefois on peut rouler tout le monde, mais on ne peut pas rouler tout le monde à chaque fois.

Le cycle de vie d'un logiciel : Planification

Planification et estimation des coûts

Le cycle de vie d'un logiciel : Planification

- **Estimation des coûts et délais par la méthode COCOMO**

COCOMO est un acronyme pour **CO**nstructive **CO**st **M**odel. C'est une méthode pour estimer le coût d'un projet logiciel dans le but d'éviter les erreurs de budget et les retards de livraison, qui sont malheureusement habituels dans l'industrie de développement logiciel.

Le cycle de vie d'un logiciel : Planification

- **Estimation des coûts et délais par la méthode COCOMO**

Le premier modèle **COCOMO date de 1981**, et a été développé par Dr. Barry Boehm pour estimer le coût , **en nombre de mois-homme**, et le temps de développement d'un produit logiciel.

A l'origine il a été construit sur une étude de 63 projets logiciels de 2000 à 100.000 lignes de code dans l'entreprise TRW Inc., **mais une seule entreprise est-elle assez représentative comme base de développement de COCOMO?** De plus, il reste très lié au nombre de lignes de code, surtout le modèle de base, mais **plus les programmeurs sont experts (et leur salaire élevé), moins ils écrivent de lignes de code pour un même projet!**

Le cycle de vie d'un logiciel : Planification

- **Estimation des coûts et délais par la méthode COCOMO**

Aujourd'hui, **COCOMO II** est un nouveau produit beaucoup plus adapté à l'aspect réutilisation des composants (modules existants).

La version 1998 a été calibrée sur 161 points de données, en utilisant l'approche statistique 'Bayesian' pour combiner les données empiriques avec les avis experts. De plus elle peut être re-calibrée sur les données de l'entreprise.

Le cycle de vie d'un logiciel : Planification

- **Estimation des coûts et délais par la méthode COCOMO**

Modèle de Base :

Le modèle de base est assez simpliste. Il estime **l'effort** (le nombre de mois-homme) en fonction du nombre de lignes de code, **la productivité** (le nombre de lignes de code par personne par mois) et un facteur **d'échelle qui dépend du type de projet.**

Les 3 types de projet identifiés sont :

- organique** : organisation simple et petites équipes expérimentées. (ex: système de notes dans une école; Produit sans interaction avec matériel,etc.)
- semi-detaché** : entre organique et imbriqué. (Produit avec peu d'interaction avec le matériel Exemple : compilateur; Forte interaction avec le matériel)
- imbriqué** : techniques innovante, organisation complexe, couplage fort avec beaucoup d'interactions. (ex : système de contrôle aérospatial.)

TABLE 6-3 Distinguishing Features of Software Development Modes

Feature	Mode		
	Organic	Semidetached	Embedded
Organizational understanding of product objectives	Thorough	Considerable	General
Experience in working with related software systems	Extensive	Considerable	Moderate
Need for software conformance with pre-established requirements	Basic	Considerable	Full
Need for software conformance with external interface specifications	Basic	Considerable	Full
Concurrent development of associated new hardware and operational procedures	Some	Moderate	Extensive
Need for innovative data processing architectures, algorithms	Minimal	Some	Considerable
Premium on early completion	Low	Medium	High
Product size range	<50 KDSI	<300 KDSI	All sizes
Examples	Batch data reduction Scientific models Business models Familiar OS, compiler Simple inventory, production control	Most transaction processing systems New OS, DBMS Ambitious inventory, production control Simple command-control	Large, complex transaction processing systems Ambitious, very large OS Avionics Ambitious command-control

*KDSI : thousands of delivered line of code

Le cycle de vie d'un logiciel : Planification

- Estimation des coûts et délais par la méthode COCOMO

Modèle de Base :

Effort : $PM = A \times KLOC^E$ (hommes-mois)

$TDEV = C \times PM^F$ (mois)

La taille de l'équipe = $PM/TDEV$ (hommes)

Type	A	E	C	F
Autonome	2,40	1,05	2,50	0,38
Couplé	3,00	1,12	2,50	0,35
Embarqué	3,60	1,20	2,50	0,32

Le cycle de vie d'un logiciel : Planification

- **Estimation des coûts et délais par la méthode COCOMO**

Modèle de Base :

Exemple

- Une entreprise souhaite gérer les matières premières qu'elle utilise. Elle fait appel à ses informaticiens en interne, qui ont l'habitude de ce genre de projets.
- Étude initiale : environ 32000 instructions.

Le cycle de vie d'un logiciel : Planification

- **Estimation des coûts et délais par la méthode COCOMO**

Modèle de Base :

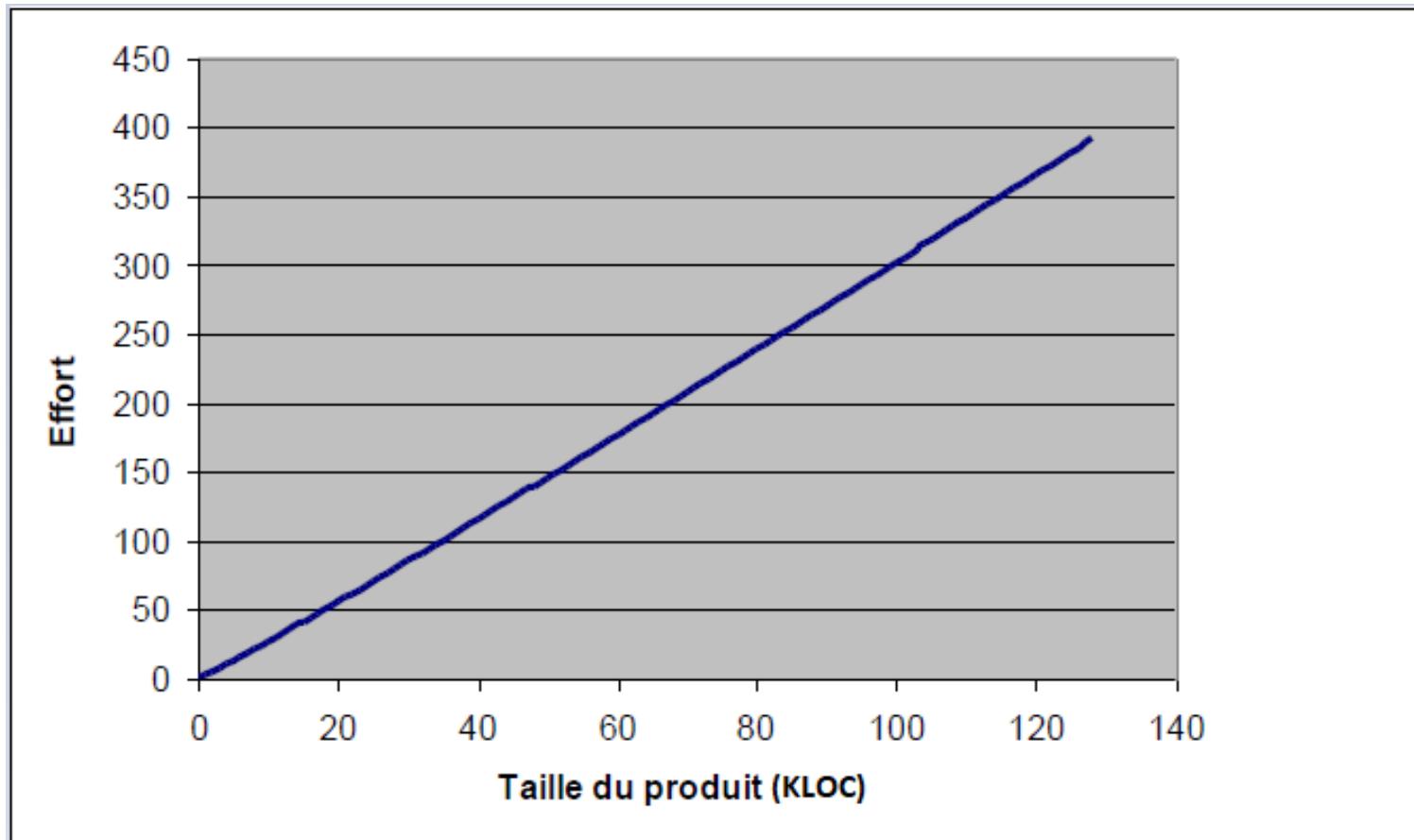
Exemple

- $PM = 2.4 \times 32^{1.05} = 91$ homme-mois
- $TDEV = 2.5 \times 91^{0.38} = 14$ mois
- Taille équipe = $91h\text{-m} / 14$ mois = 6.5 personnes à temps plein
- Productivité = Taille du projet / PM = 352 lignes/homme-mois

Le cycle de vie d'un logiciel : Planification

- Estimation des coûts et délais par la méthode COCOMO

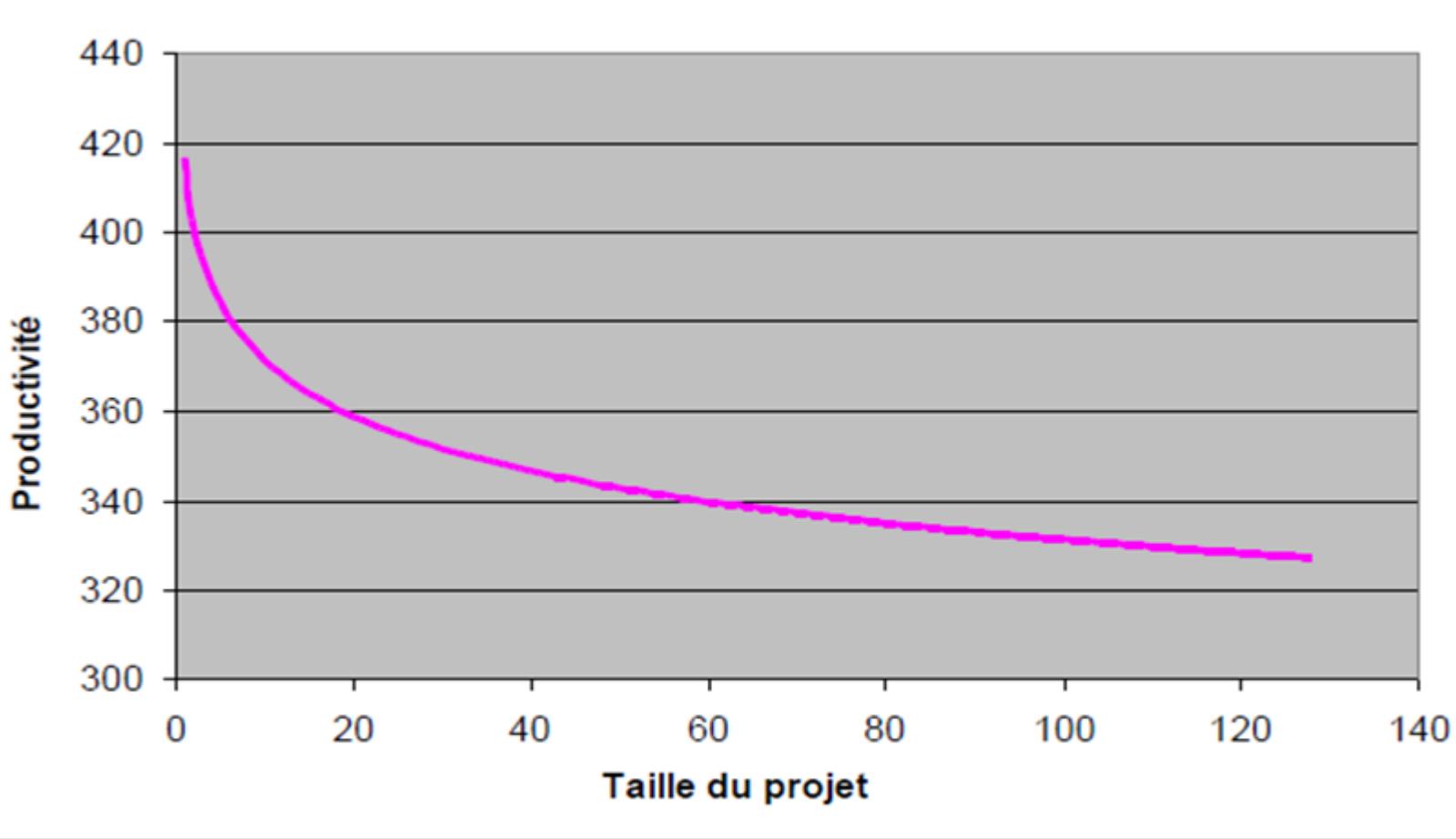
La relation entre effort (PM) et Taille du produit en KLOC



Le cycle de vie d'un logiciel : Planification

- **Estimation des coûts et délais par la méthode COCOMO**

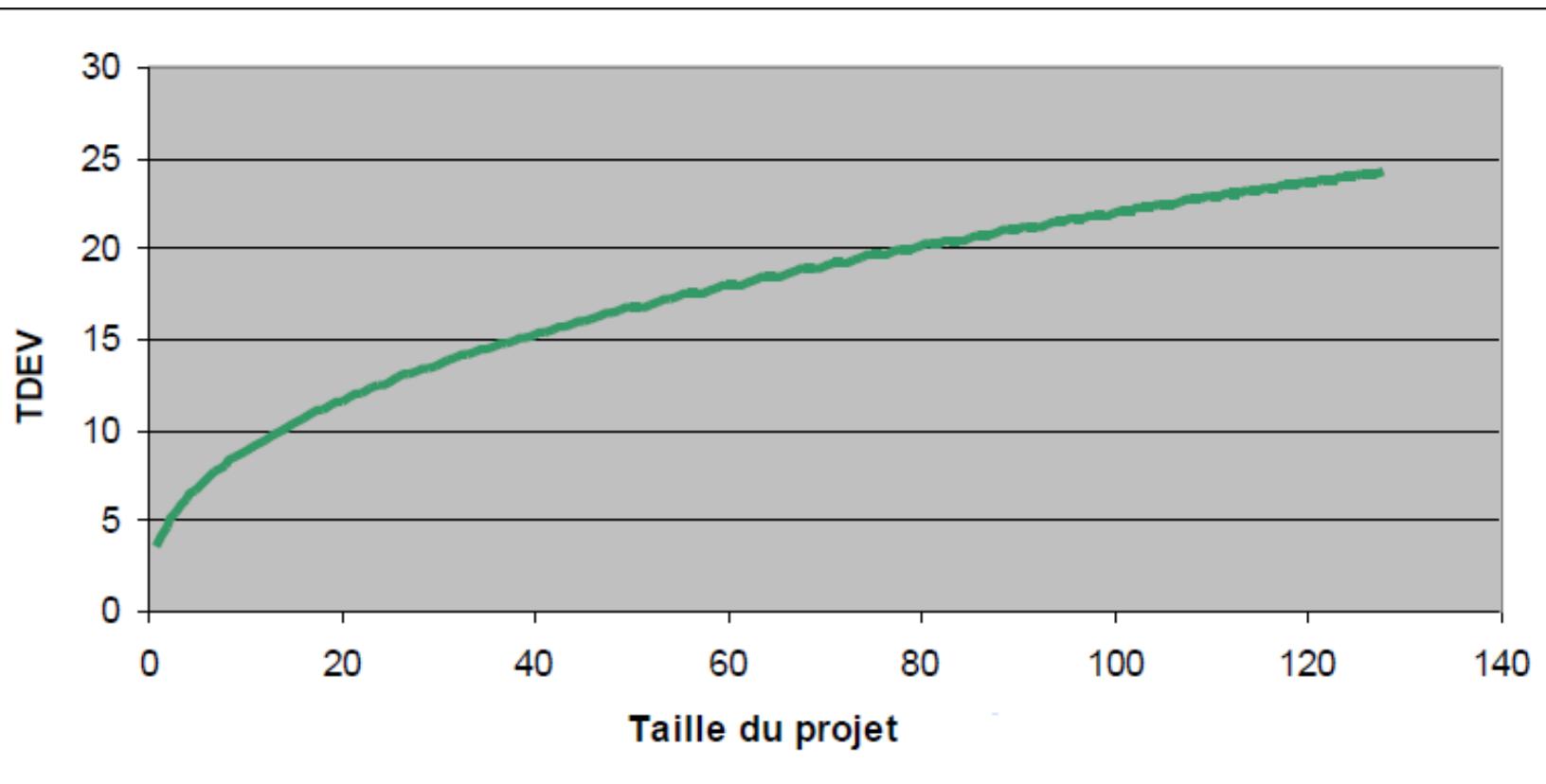
La relation entre productivité (lignes/h-mois) et Taille du produit en KLOC



Le cycle de vie d'un logiciel : Planification

- Estimation des coûts et délais par la méthode COCOMO

La relation entre TDEV (mois) et Taille du produit en KLOC



Le cycle de vie d'un logiciel : Planification

Complexité	Phase	Distribution par phase du temps de développement				En Pourcentage
		Taille de 2 KLS	Taille de 8 KLS	Taille de 32 KLS	Taille de 128 KLS	
S	Expression des besoins et planification	10	11	12	13	
	Conception générale	19	19	19	19	
	Programmation	63	59	55	51	
	Tests et intégration	18	22	26	30	
P	Expression des besoins et planification	16	18	20	22	24
	Conception générale	24	25	26	27	28
	Programmation	56	52	48	44	40
	Tests et intégration	20	23	26	29	32
E	Expression des besoins et planification	24	28	32	36	40
	Conception générale	30	32	34	36	38
	Programmation	48	44	40	36	32
	Tests et intégration	22	24	26	28	30

Le cycle de vie d'un logiciel : Planification

Le modèle COCOMO intermédiaire

Modèle de base + attributs

Attributs du produit

- RELY : sûreté du produit
- DATA : taille de la base de données
- CPLX : complexité du produit

Attributs de la machine

TIME : contraintes de temps d'exécution

STOR : contraintes de mémoire principale

VIRT : changement dans l'ensemble soft+hard

TURN : temps de retour d'une tâche soumise à l'ordinateur.

Le cycle de vie d'un logiciel : Planification

Le modèle COCOMO intermédiaire

Attributs du personnel

- ACAP : aptitude de l'analyste
- AEXP : expérience de ces applications
- PCAP : aptitude du programmeur
- VEXP : expérience de l'ensemble soft+hard
- LEXP : expérience du langage de programmation

Attributs du projet

- MODP : pratique des techniques modernes de programmation
- TOOL : utilisation d'outils
- SCED : agenda imposé

TABLE 8-3 Software Cost Driver Ratings

Cost Driver	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
RELY	Effect: slight inconvenience	Low, easily recoverable losses	Moderate, recoverable losses	High financial loss	Risk to human life	
DATA		$\frac{\text{DB bytes}}{\text{Prog. DS}}$ < 10	$10 \leq \frac{D}{P} < 100$	$100 \leq \frac{D}{P} < 1000$	$\frac{D}{P} \geq 1000$	
CPLX	See Table 8-4					
Computer attributes						
TIME			$\leq 50\%$ use of available execution time	70%	85%	95%
STOR			$\leq 50\%$ use of available storage	70%	85%	95%
VIRT		Major change every 12 months Minor: 1 month	Major: 6 months Minor: 2 weeks	Major: 2 months Minor: 1 week	Major: 2 weeks Minor: 2 days	
TURN		Interactive	Average turnaround <4 hours	4–12 hours	>12 hours	
Personnel attributes						
ACAP	15th percentile ^a	35th percentile	55th percentile	75th percentile	90th percentile	
AEXP	≤ 4 months experience	1 year	3 years	6 years	12 years	
PCAP	15th percentile ^a	35th percentile	55th percentile	75th percentile	90th percentile	
VEXP	≤ 1 month experience	4 months	1 year	3 years		
LEXP	≤ 1 month experience	4 months	1 year	3 years		
Project attributes						
MODP	No use	Beginning use	Some use	General use	Routine use	
TOOL	Basic microprocessor tools	Basic mini tools	Basic midi/maxi tools	Strong maxi programming, test tools	Add requirements, design, management, documentation tools	
SCED	75% of nominal	85%	100%	130%	160%	

^a Team rating criteria: analysis (programming) ability, efficiency, ability to communicate and cooperate

Le cycle de vie d'un logiciel : Planification

Le modèle COCOMO intermédiaire

□ Pour chaque attribut

- On associe un facteur multiplicatif
- On multiplie ces 15 facteurs
- Puis équations :
 - Organique : $MM_{nominal} = 3.2 \times KDSI^{1.05}$
 - Semi détaché : $MM_{nominal} = 3.0 \times KDSI^{1.12}$
 - Embarqué : $MM_{nominal} = 2.8 \times KDSI^{1.20}$
- Finalement :
 - $MM = ProduitFacteurs \times MM_{nominal}$

*MM == PM

*KDSI == KLOC

Le cycle de vie d'un logiciel : Planification

Le modèle COCOMO intermédiaire : Facteurs multiplicatifs

TABLE 8-2 Software Development Effort Multipliers

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY Required software reliability	.75	.88	1.00	1.15	1.40	
DATA Data base size		.94	1.00	1.08	1.16	
CPLX Product complexity	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME Execution time constraint			1.00	1.11	1.30	1.66
STOR Main storage constraint			1.00	1.06	1.21	1.56
VIRT Virtual machine volatility ^a		.87	1.00	1.15	1.30	
TURN Computer turnaround time		.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP Applications experience	1.29	1.13	1.00	.91	.82	
PCAP Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP Virtual machine experience ^a	1.21	1.10	1.00	.90		
LEXP Programming language experience	1.14	1.07	1.00	.95		
Project Attributes						
MODP Use of modern programming practices	1.24	1.10	1.00	.91	.82	
TOOL Use of software tools	1.24	1.10	1.00	.91	.83	
SCED Required development schedule	1.23	1.08	1.00	1.04	1.10	

Le cycle de vie d'un logiciel : Planification

Le modèle COCOMO intermédiaire : Facteurs multiplicatifs

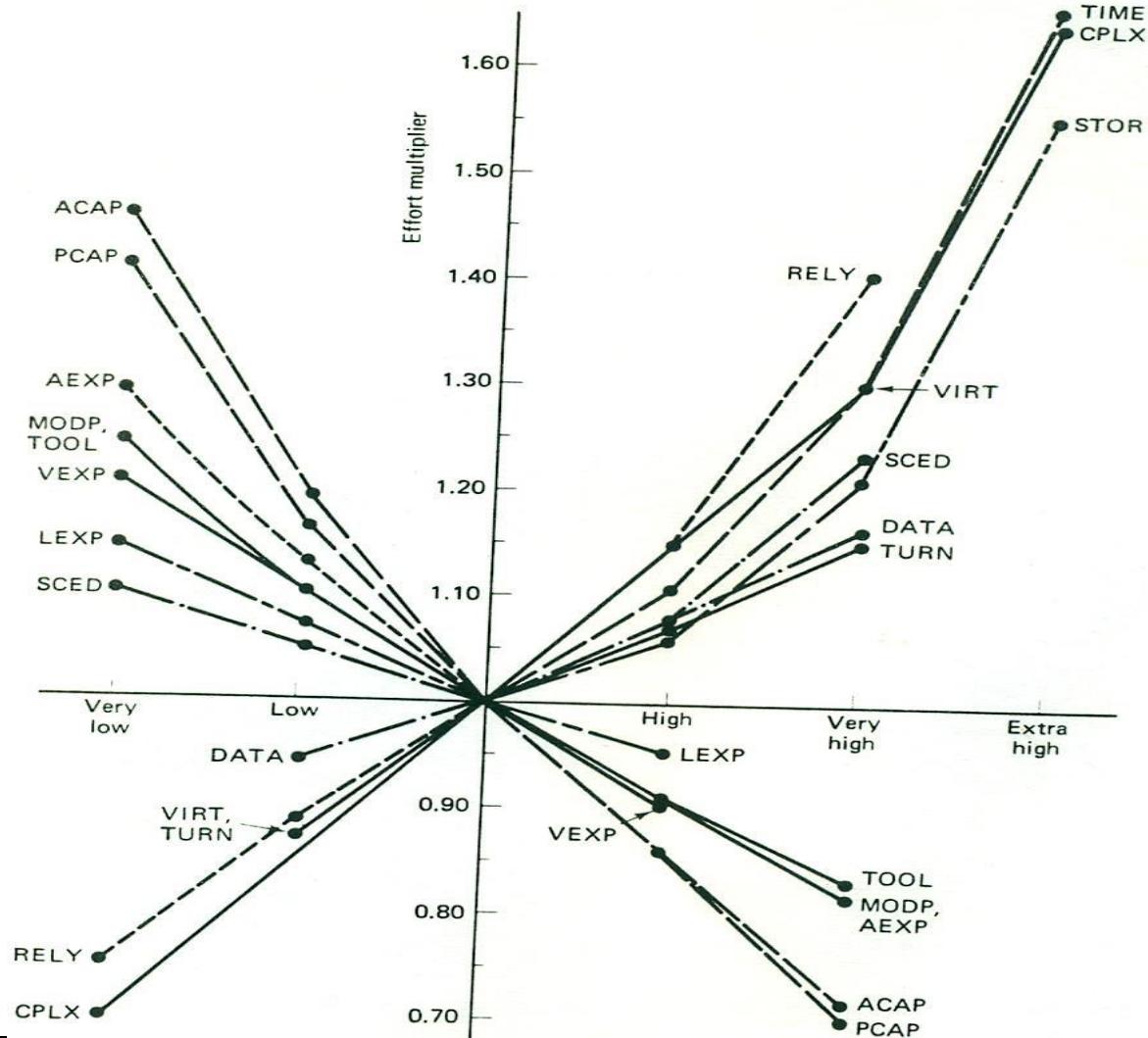


FIGURE 8-2 Intermediate COCOMO effort multipliers

Le cycle de vie d'un logiciel : Planification

Méthode d'estimation à trois points

La méthode d'estimation à 3 points, se base sur des dates estimées (temps moyen/temps optimiste/temps pessimiste) et sur l'écart type.

TM: temps moyen estimé (travail dans des conditions normales).

TO: temps optimiste (conditions idéales, pas d'obstacles, temps minimum pour accomplir la tâche).

TP: temps pessimiste (temps maximum pour accomplir la tâche dans les pires conditions).

- Pour une probabilité avec une distribution normale, la formule est :
$$P = TO + (TP - TO) / 2$$
- Pour une probabilité avec une distribution Beta (à double triangulaire) formule de répartition sera la suivante : $P = (TP + 4 TM + TO) / 6$

Le cycle de vie d'un logiciel : Planification

Méthode des Potentiels et antécédents Métra « M.P.M »

- La Méthode des Potentiels et antécédents Métra (MPM) est, comme le PERT, une technique d'ordonnancement basée sur la théorie des graphes, visant à optimiser la planification des tâches d'un projet.
- mise au point en 1958 par un chercheur français, Bernard Roy, au sein de la société de conseil Métra.

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode M.P.M:

- Étape 1 : Énumérer les activités

Nom du projet	Tâche 1	Sous-tâche 1.1	Élément de travail 1.1.1
		Sous-tâche 1.2	Élément de travail 1.1.2
	Tâche 2	Sous-tâche 2.1	Élément de travail 1.2.1
			Élément de travail 1.1.2
			Élément de travail 2.1.1
			Élément de travail 1.1.1

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM:

Étape 2 : Établir les dépendances (séquence d'activités)

- Quelle tâche doit être réalisée avant que cette tâche ne se réalise ?
- Quelles tâches doivent être achevées en même temps que celle-ci ?
- Quelles tâches doivent être réalisées immédiatement après celle-ci?

Tâches	Durée	Antériorité
A	2	-
B	4	-
C	4	A
D	5	A,B
E	6	C,D

TABLEAU D'ANTÉRIORITÉ DU PROJET

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 3 : Dessiner le graphe MPM

La méthode des potentiels Métra permet de représenter l'ensemble de ces tâches sur un graphe orienté, à partir duquel il sera possible d'identifier leurs dates au plus et au plus tard et de calculer leurs marges.

Un graphe orienté est un réseau composé d'une entrée et d'une sortie, ainsi que de points (appelés "sommets") reliés entre eux par des flèches (appelées "arcs").

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 3 : Dessiner le graphe MPM

Les principales conventions d'un réseau MPM sont les suivantes :

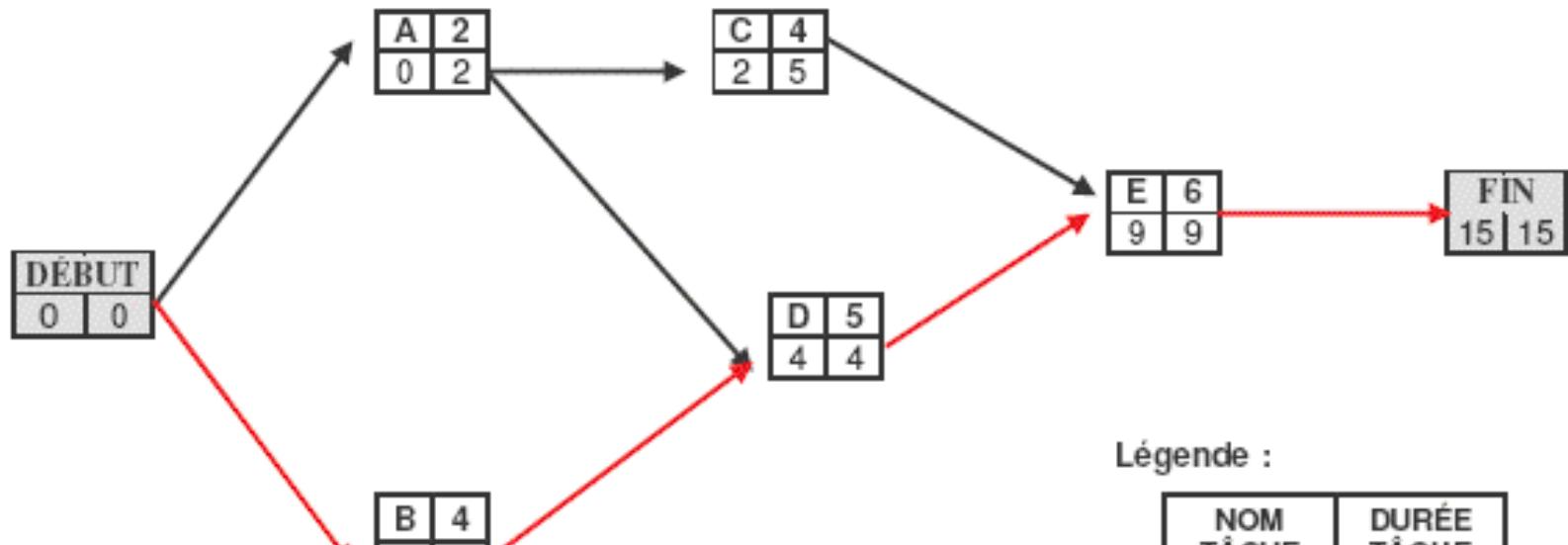
- chaque tâche est représentée par un sommet
- les contraintes de succession sont symbolisées par les arcs
- chaque tâche est renseignée sur sa durée ainsi que sur la date à laquelle elle peut commencer au plus tôt ("date au plus tôt") et au plus tard ("date au plus tard") pour respecter le délai optimal de réalisation du projet.
- le graphe commence et termine sur 2 sommets, respectivement appelés "Début" et "Fin" symbolisant les début et fin des opérations (mais ne correspondant pas une tâche).

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 3 : Dessiner le graphe MPM



Légende :

NOM TÂCHE	DURÉE TÂCHE
Date au plus tôt	Date au plus tard

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :détermination des dates "au plus tôt" et "au plus tard" dans un réseau MPM

La date au plus tôt d'un réseau MPM correspond à la date à laquelle une tâche peut commencer au plus tôt.

- Elle s'obtient très simplement en ajoutant à la date au plus tôt de la tâche précédente la durée de la tâche en question :

$$\text{Date au plus tôt tâche T} = \text{Date au plus tôt tâche S} + \text{Durée tâche S}$$

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :détermination des dates "au plus tôt" et "au plus tard" dans un réseau MPM

Lorsque plusieurs arcs arrivent à un même sommet (c'est-à-dire que plusieurs tâches sont immédiatement antérieures à la tâche considérée), il convient, d'effectuer ce calcul pour toutes les tâches précédant la tâche en question et de retenir comme "date au plus tôt" de cette dernière le maximum des valeurs ainsi trouvée (en effet, cette tâche ne pourra vraiment débuter que lorsque toutes les tâches qui lui sont immédiatement antérieures auront été terminées).

La formule précédente devient donc :

$$\text{Date au plus tôt tâche T} = \text{Max. (Date plus tôt tâches S + Durée tâches S)}$$

*Dans cette formule, "S" représente l'ensemble des tâches immédiatement antérieures à "T"

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :détermination des dates "au plus tôt" et "au plus tard" dans un réseau MPM

La détermination des dates au plus tard des différentes tâches se fait à rebours du graphe, par calculs successifs, en partant du sommet "Fin" (pour lequel, par convention, on considère que la date au plus tard est égale à sa date au plus tôt).

Date au plus tard tâche S = Min. (date au plus tard tâches T - durée tâche S)

*Dans cette formule, "T" représente l'ensemble des tâches immédiatement postérieures à "S"

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :calcul des différentes marges d'une tâche dans un réseau MPM

On appelle "**marge**" d'une tâche le retard qu'il est possible de tolérer dans la réalisation de celle-ci, sans que la durée optimale prévue du projet global en soit affectée.

Il est possible de calculer trois types de marges : **la marge totale, la marge certaine et la marge libre.**

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :calcul des différentes marges d'une tâche dans un réseau MPM

La marge totale d'une tâche indique le retard maximal que l'on peut admettre dans sa réalisation (sous réserve qu'elle ait commencé à sa date au plus tôt) sans allonger la durée optimale du projet.

Marge totale tâche S = Date plus tard tâche S - Date plus tôt tâche S

*cas particulier, un retard correspondant à la marge totale d'une tâche se traduit par une modification des dates au plus tôt des tâches qui lui succèdent et entraîne, généralement, l'apparition d'un 2° chemin critique. Il n'est donc pas possible de cumuler des retards correspondant à leur marge totale sur plusieurs tâches successives, sans remettre en cause la durée optimale prévue pour le projet.

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :calcul des différentes marges d'une tâche dans un réseau MPM

La marge libre d'une tâche indique **le retard que l'on peut admettre dans sa réalisation** (sous réserve qu'elle ait commencé à sa date au plus tôt) **sans modifier les dates au plus tôt des tâches suivantes et sans allonger la durée optimale du projet.**

Elle se calcule en retirant la durée de la tâche en question à l'écart existant entre sa date au plus tôt de la date au plus tôt de la tâche suivante :

$$\text{Marge libre tâche S} = \text{Date plus tôt tâche T} - \text{Date plus tôt tâche S} - \text{Durée tâche S}$$

*T : Tâche suivante à S.

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :calcul des différentes marges d'une tâche dans un réseau MPM

Lorsque plusieurs arcs partent d'un même sommet, il convient de faire ce calcul pour toutes les tâches succédant à la tâche en question et de retenir comme "marge libre" de la tâche en question la valeur minimale des marges ainsi déterminées :

Marge libre tâche S = Min (Date plus tôt tâches T - Date plus tôt tâche S - Durée tâche S)

- Dans cette formule T représente l'ensemble des tâches succédant immédiatement à S
- Un retard correspondant à la marge libre d'une tâche reste sans conséquence sur les marges des tâches qui lui succèdent. Il est donc **possible de cumuler des retards**, s'inscrivant dans leur marge libre, **pour plusieurs tâches successives**, sans remettre en cause la durée optimale prévue pour le projet.

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :calcul des différentes marges d'une tâche dans un réseau MPM

La marge certaine d'une tâche indique le retard que l'on peut admettre dans sa réalisation (quelle que soit sa date de début) sans allonger la durée optimale du projet.

Elle se calcule en retirant la durée de la tâche en question à l'écart qu'il peut y avoir entre sa date au plus tard de début et sa date au plus tôt de fin :

$$\text{Marge certaine tâche S} = \text{Max} [0 , \text{Min} (\text{Date au plus tôt tâche T} - \text{Date au plus tard tâche S} - \text{Durée tâche S})]$$

* D'après cette formule, la marge certaine est considérée comme nulle lorsque son calcul donne un nombre négatif

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :calcul des différentes marges d'une tâche dans un réseau MPM

Lorsque plusieurs arcs partent d'un même sommet , il convient de faire ce calcul pour toutes les tâches succédant à la tâche en question et de retenir comme "marge certaine" de cette dernière la valeur minimale des marges ainsi déterminées :

Marge certaine tâche S = Max [0, Min (Date au plus tôt tâches T - Date au plus tard tâche S - Durée tâche S)]

*Dans cette formule T représente l'ensemble des tâches succédant immédiatement à S

- Un retard correspondant à **la marge certaine d'une tâche** reste sans conséquence sur les marges des tâches qui lui succèdent, même si elle a commencé à sa date au plus tard.
- Il est donc **possible de cumuler des retards**, s'inscrivant dans leur marge certaine, pour plusieurs tâches successives, même si elles commencent à leur date au plus tard, sans remettre en cause la durée optimale prévue pour le projet.

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

Étapes clés de la méthode du MPM

Étape 4 :calcul des différentes marges d'une tâche dans un réseau MPM

Les marges des tâches composant le **chemin critique** sont nécessairement **nulles**, puisqu'il s'agit de tâches pour lesquels, par définition, **aucun retard n'est possible** sans remettre en cause la durée optimale prévue pour le projet.

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

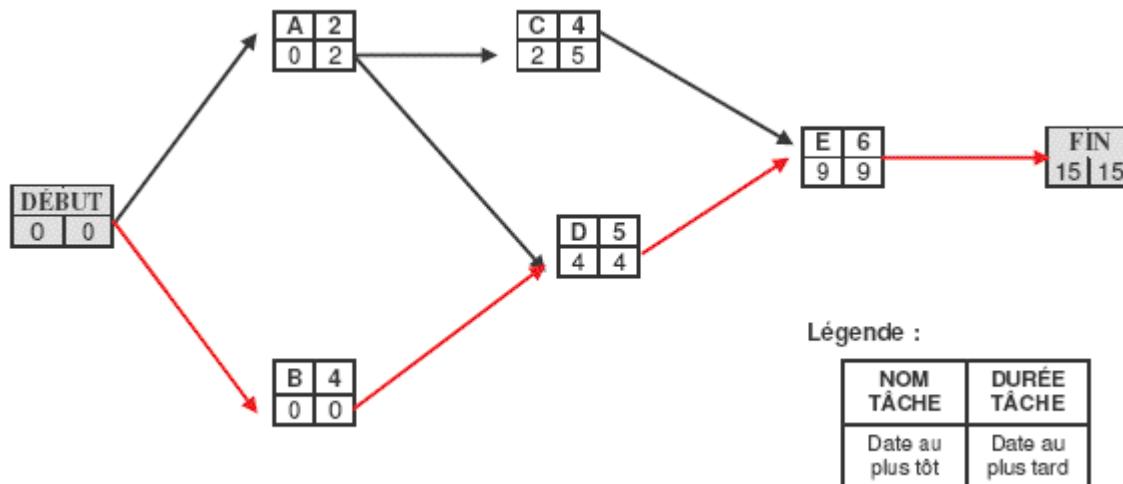
- Exemple : calculer les dates débuts au plus tôt et au plus tard, les marges et déterminer le chemin critique du projet suivant:

Tâches	Durée	Antériorité
A	2	-
B	4	-
C	4	A
D	5	A,B
E	6	C,D

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

- Graphe MPM



Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

- Les marges

Tâches	Marge Totale	Marge libre	Marge certaine
A	2	0	0
B	0	0	0
C	3	3	0
D	0	0	0
E	0	0	0

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN RÉSEAU MPM

- Exercice :

Un projet informatique « P » consiste à développer un site web:

Tâche	Durée (jours)	Antécédent
a. Planification	2	--
b. Conception	5	a
c. Développement de la base de données	7	b
d. Développement du front-end	14	a
e. Développement du back-end	14	c
f. Intégration du front-end et du back-end	5	d,e
g. Tests	4	d,e
h. mise en ligne et déploiement	2	f,g

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Le **PERT** (Programm of Evaluation and Review Technic) est, comme la MPM, une technique d'ordonnancement basée sur la théorie des graphes, visant à optimiser la planification des tâches d'un projet.
- Cette technique aurait été conçue sur la base de la méthode **CPM (Critical Method Path)** par la marine américaine, en 1958, pour coordonner les tâches des milliers d'entreprises impliquées dans son projet "Polaris" (programme de développement de missiles à ogive nucléaire).
- Elle aurait permis de réduire de 14 à 7 ans la durée globale de réalisation du projet Polaris) elle s'est rapidement imposée dans les organisations, gouvernementales ou non, ayant à gérer des projets importants (programme Apollo de la NASA, construction d'autoroute, etc.) au détriment du diagramme de Gantt.

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Program of Evaluation and Review Technic »

- L'utilisation du PERT permet, de déterminer *la durée minimum nécessaire pour mener à bien un projet* et les dates auxquelles peuvent ou doivent débuter les différentes tâches nécessaires à sa réalisation pour que cette durée minimum soit respectée.

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Program of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

1- identifiées les différentes tâches nécessaires à la réalisation d'un projet, leur durée et leurs relations d'antériorité .

Tâches	Durée	Antériorité(s)
A	2	-
B	4	-
C	4	A
D	5	A,B
E	6	C,D

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

Les principales conventions d'un réseau PERT sont les suivantes :

- chaque **tâche est symbolisée par un arc**, auquel est associé une valeur numérique correspondant à sa durée.
- Les **sommets** auxquels aboutissent les arcs correspondent donc à **des étapes**, qui marquent l'aboutissement d'une ou plusieurs tâches.
- **Chaque étape** est identifiée par un numéro d'ordre et renseignée sur la date à laquelle elle peut être atteinte au plus tôt ("**date au plus tôt**") et au plus tard ("**date au plus tard**") pour respecter le délai optimal de réalisation du projet.
- Le graphe possède une entrée (sommet sans antécédent) et une sortie (sommet sans descendant) qui correspondent respectivement aux étapes "Début des opérations" et "Fin des opérations".
- il est parfois nécessaire d'introduire des "**tâches fictives**" pour **traduire correctement sur un graphe les relations d'antériorité** de certaines tâches, notamment lorsque celles-ci partagent avec d'autres une partie de leurs antécédents

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

2-Construction d'un graphe PERT

La démarche la plus appropriée consiste à procéder par "niveau" :

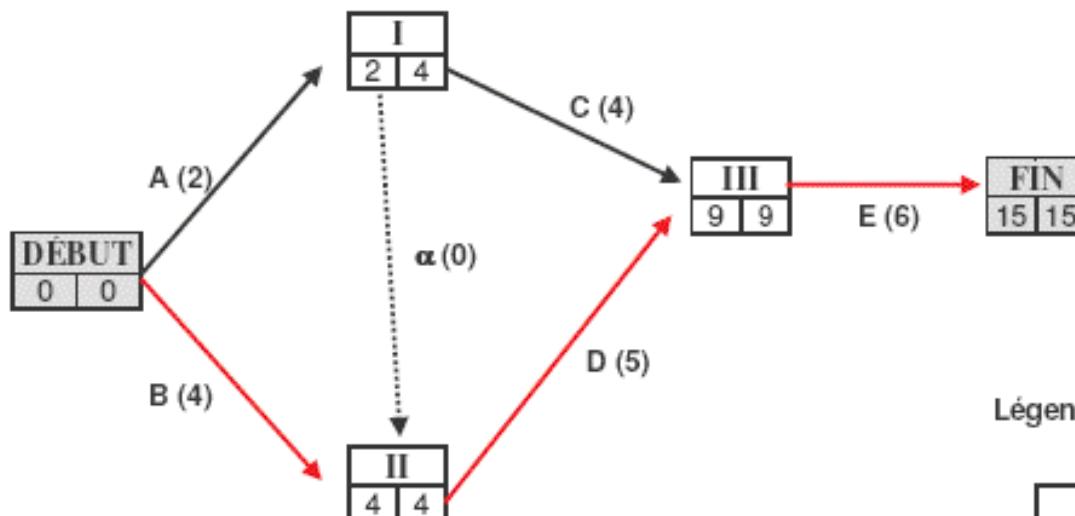
- Déterminer les tâches sans antécédent (tâches de niveau 1) et les relier à l'étape de "Début".
- Identifier ensuite les tâches de niveau 2, c'est-à-dire celles dont les antécédents sont exclusivement du niveau 1 et les positionner sur le graphique en fonction de des derniers,
- continuer ainsi, jusqu'à ce que toutes les tâches aient pu être positionnées entre elles et relier celles n'ayant pas de descendant à l'étape de "Fin".

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

2-Construction d'un graphe PERT



Légende :

N° ÉTAPE	Date au plus tôt	Date au plus tard

Remarque : sur ce graphique il a été nécessaire d'introduire une "tâche fictive α " (de durée nulle) pour traduire le fait que la tâche D ne pouvait commencer qu'après complet achèvement des tâches A et B.

NOM TÂCHE (Durée tâche)

TÂCHE FICTIVE (Durée nulle)

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

2-Détermination des dates "au plus tôt" et "au plus tard" dans un réseau PERT

Date au plus tôt "étape j" = Date au plus tôt "étape i" + Durée tâche "ij"

Date au plus tôt "étape j" = Max. (Date plus tôt "étapes i" + Durée tâches "ij")

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

2-Détermination des dates "au plus tôt" et "au plus tard" dans un réseau PERT

Date au plus tard "étape i" = Date plus tard "étape j" - Durée tâche "ij"

Date au plus tard "étape i" = Min. (date au plus tard "étapes j" - Durée tâches "ij")

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

3-Détermination des Marges:

Marge totale tâche "ij" = Date au plus tard "étape j" - Date au plus tôt "étape i" - Durée tâche "ij"



Marge totale tâche "ij" = $T_j - t_i - \text{Durée tâche "ij"}$

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

3-Détermination des Marges:

Marge libre tâche "ij" = Date au plus tôt "étape j" - Date au plus tôt "étape i" - Durée tâche "ij"



Marge libre tâche "ij" = $tj - ti - \text{durée } ij$

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

3-Détermination des Marges:

Marge certaine tâche "ij" = Max [0 , (Date au plus tôt "étape j" - Date au plus tard "étape i" - Durée tâche "ij")]



Marge certaine tâche "ij" = Max [0 , ($t_j - T_i - \text{Durée } ij$)]

Le cycle de vie d'un logiciel : Planification

Le diagramme de PERT « Programm of Evaluation and Review Technic »

- Méthodologie de construction d'un réseau PERT

3-Détermination le chemin critique :

On remarque que l'ensemble des marges des tâches composant le **chemin critique sont nécessairement nulles**, puisqu'il s'agit de tâches pour lesquels, par définition, aucun retard n'est possible sans remettre en cause la durée optimale prévue pour le projet.

Le cycle de vie d'un logiciel : Planification

Diagramme de PERT

- Exercice :

Un projet informatique « P » consiste à développer un site web:

Tâche	Durée (jours)	Antécédent
a. Planification	2	--
b. Conception	5	a
c. Développement de la base de données	7	b
d. Développement du front-end	14	a
e. Développement du back-end	14	c
f. Intégration du front-end et du back-end	5	d,e
g. Tests	4	d,e
h. mise en ligne et déploiement	2	f,g

Le cycle de vie d'un logiciel : Planification

Diagramme de Gantt

- Le **Diagramme de Gantt** a été conçu en 1917, par Henry L. Gantt.
- Le Diagramme de Gantt se présente **sous la forme d'un planning** présentant en ligne **les tâches élémentaires** d'un projet et en colonne **l'échelle de temps** retenue (jours, semaine, etc.).

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN DIAGRAMME DE GANTT

1. Déterminer et structurer la liste des tâches.
2. Estimer les durées et les ressources des tâches identifiées.
3. Réaliser le "réseau logique" : les relations d'antériorité des tâches définies.

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN DIAGRAMME DE GANTT

4) Tracer le diagramme de GANTT :

- Apparaître en ordonnée la liste des "lots de tâches" (**workpackages**) et en abscisse l'échelle de temps adopté (jours, semaines ou mois).
- Les tâches sont figurées sous la forme de traits ou de rectangles d'une longueur proportionnelle à leur durée, leur position horizontale reflétant leur ordre logique d'exécution.
- La méthode la plus simple consiste à commencer par représenter les tâches n'ayant aucune antériorité, puis celles qui peuvent immédiatement leur succéder et ainsi de suite jusqu'à ce que toutes les tâches aient été positionnées.
- Éventuellement plusieurs tâches peuvent être réalisées simultanément (sous réserve d'une disponibilité des ressources nécessaires) ce qui permet de diminuer la durée totale d'exécution du projet et, donc, son coût.

Le cycle de vie d'un logiciel : Planification

MÉTHODOLOGIE DE CONSTRUCTION D'UN DIAGRAMME DE GANTT

DIAGRAMME DE GANTT DU PROJET X

ACTIVITÉS	Durée (sem.)	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11
Tâche 1	2											
Tâche 2	3,5											
Tâche 3	4,5											
Tâche 4	4,5											
Tâche 5	3,5											

Le cycle de vie d'un logiciel : Planification

DIAGRAMME DE GANTT : EXERCICES

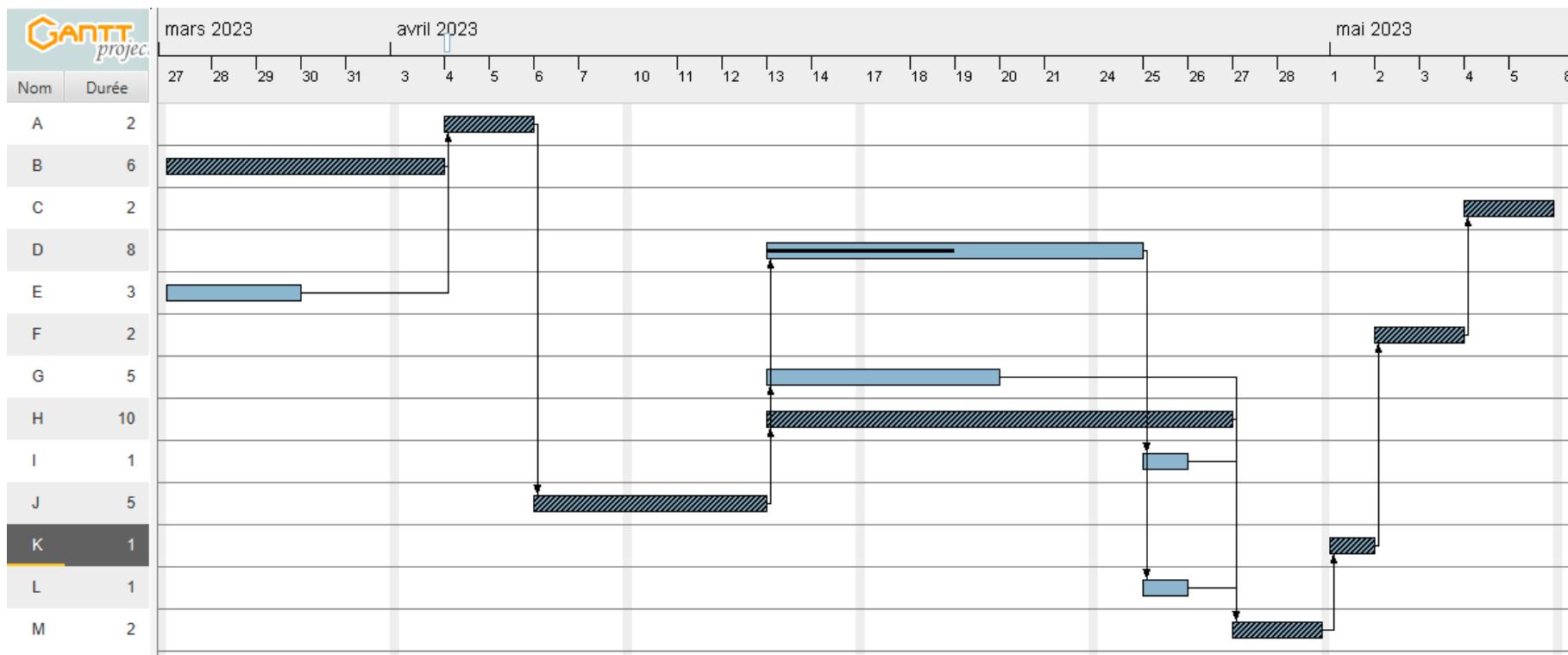
Tracer le diagramme de gantt du projet suivant :

Code de la tâche	Désignation de la tâche	Durée en jours	Tâches antérieures
A	Définition des contraintes	2	B E
B	Mise en place du projet	6	-
C	Mise à jour des droits d'accès	2	F
D	Achat des composants matériels	8	J
E	Définition du budget	3	-
F	Mise à jour des gr. utilisateurs	2	K
G	Formation de l'administrateur réseau	5	J
H	Câblage	10	J
I	Commande de Novell Netware 5	4	D
J	Choix des fournisseurs	5	A
K	Mise à jour logicielle des postes	1	M
L	Mise à jour matérielle des postes	2	D
M	Installation Novell Netware 5	2	LIHG

Le cycle de vie d'un logiciel : Planification

DIAGRAMME DE GANTT : EXERCICES

Tracer le diagramme de gantt du projet suivant :



Le cycle de vie d'un logiciel : Conception

- ***La phase d'analyse (définition des besoins et spécification fonctionnelle et techniques)*** est suivie de la phase de conception.
- Généralement ***la conception*** est décomposée en deux phases successives :
 - ***Conception générale*** ou conception architecturale (preliminary design ou architectural design)
 - ***Conception détaillée (detailed design)***

Le cycle de vie d'un logiciel : Conception

- *Conception générale :*

Décrire l'architecture de la solution, c'est-à-dire son organisation en entités, les interfaces de ces entités et les interactions entre ces entités.

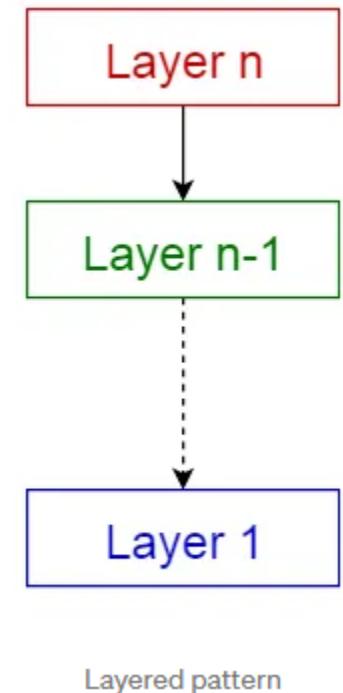
Le cycle de vie d'un logiciel : Conception

- **Conception générale** : Architecture logicielle courantes

- **Modèle en couches (Layered pattern)**

Les 4 couches les plus courantes d'un système d'information général sont les suivantes. -

- Couche de présentation (également appelée couche d'interface utilisateur) .
- Couche d'application (également appelée couche de service)
- Couche de logique métier (également appelée couche de domaine) .
- Couche d'accès aux données (également appelée couche de persistance).



Le cycle de vie d'un logiciel : Conception

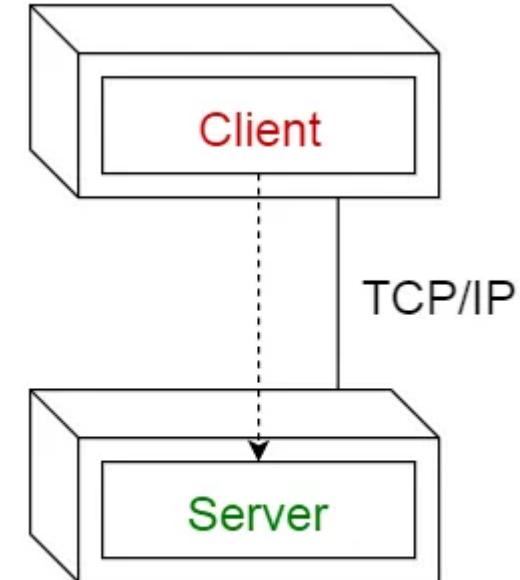
- ***Conception générale*** : Architecture logicielle courantes

- **Modèle client-serveur (Client-server pattern)**

Ce modèle se compose de deux parties; un serveur et plusieurs clients.

Les clients demandent des services au serveur

Le serveur fournit des services pertinents à ces clients.



Client-server pattern

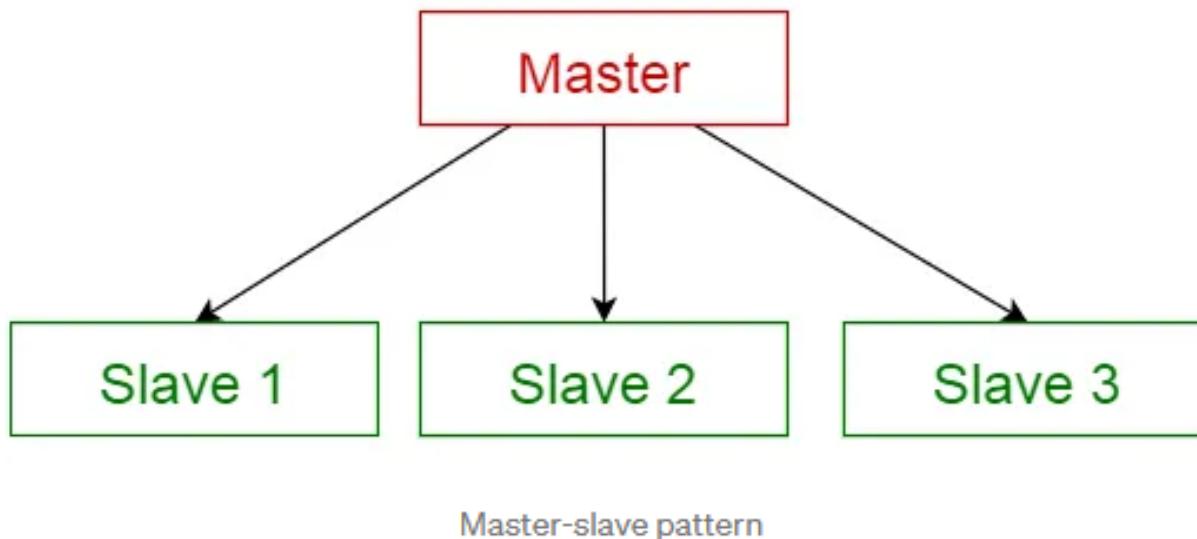
Le cycle de vie d'un logiciel : Conception

- ***Conception générale*** : Architecture logicielle courantes

- Modèle maître-esclave (Master-slave pattern)**

Ce modèle se compose de deux parties: maître et esclaves.

Le composant maître distribue le travail entre des composants esclaves identiques et calcule un résultat final à partir des résultats renvoyés par les esclaves.



Le cycle de vie d'un logiciel : Conception

- ***Conception générale*** : Architecture logicielle courantes

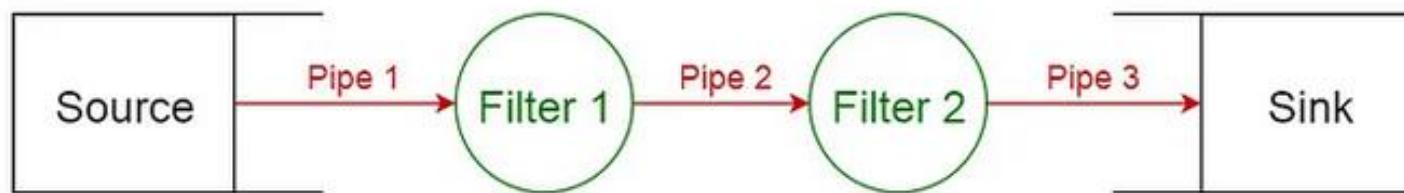
- Modèle de canal-filtre (Pipe-filter pattern)**

Ce modèle peut être utilisé pour structurer des systèmes qui produisent et traitent un flux de données.

Chaque étape de traitement est enfermée dans un composant de filtre.

Les données à traiter passent par des canaux.

Ces canaux peuvent être utilisés pour la mise en mémoire tampon ou à des fins de synchronisation.

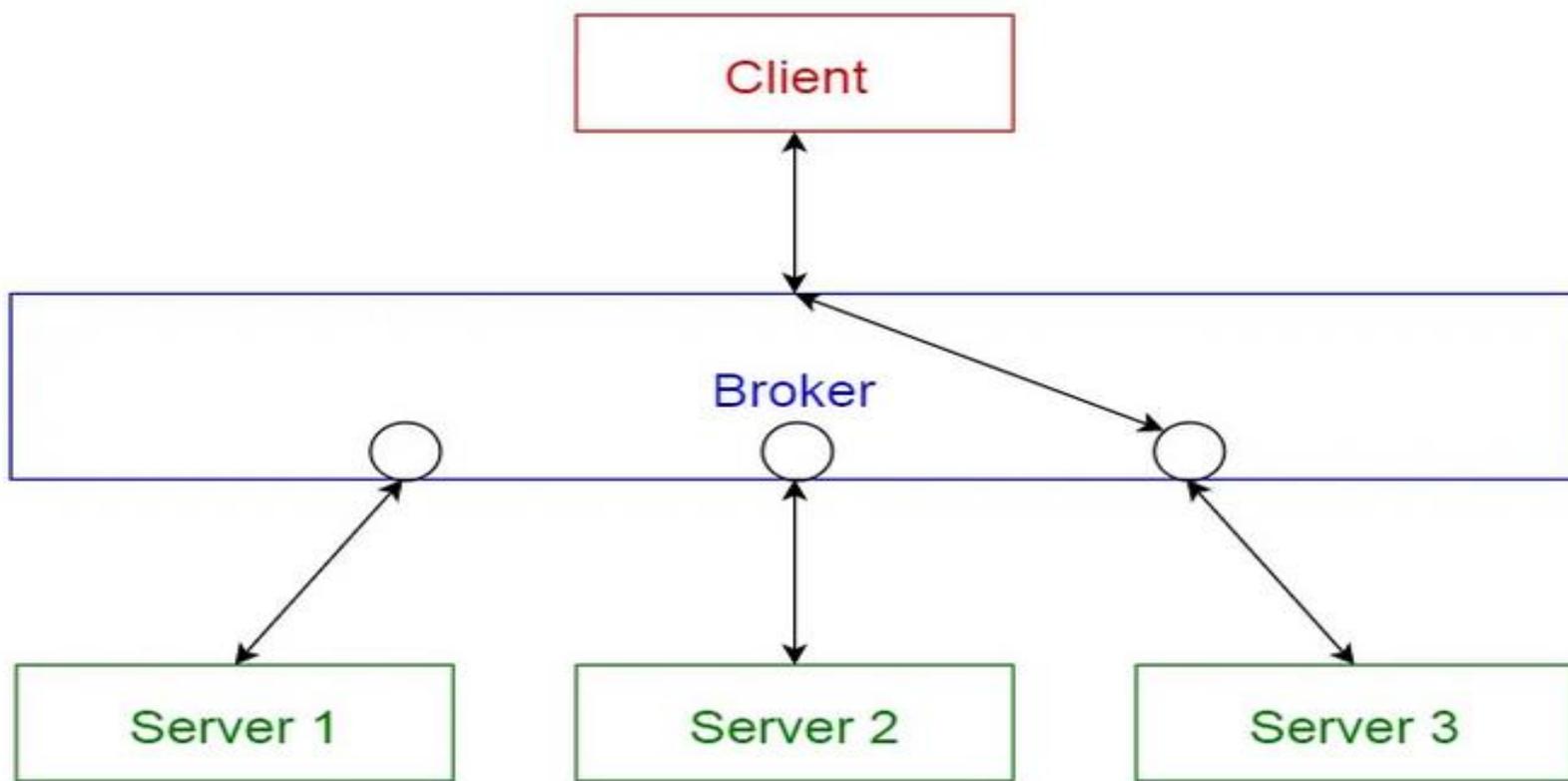


Le cycle de vie d'un logiciel : Conception

- ***Conception générale*** : Architecture logicielle courantes
 - **Modèle de courtier (Broker pattern)**
 - Ce modèle est utilisé pour structurer des systèmes distribués avec des composants découplés.
 - Ces composants peuvent interagir les uns avec les autres par des appels de service à distance.
 - Un composant courtier est responsable de la coordination de la communication entre les composants.
 - Les serveurs publient leurs capacités (services et caractéristiques) à un courtier.
 - Les clients demandent un service au courtier, et le courtier redirige ensuite le client vers un service approprié à partir de son registre.

Le cycle de vie d'un logiciel : Conception

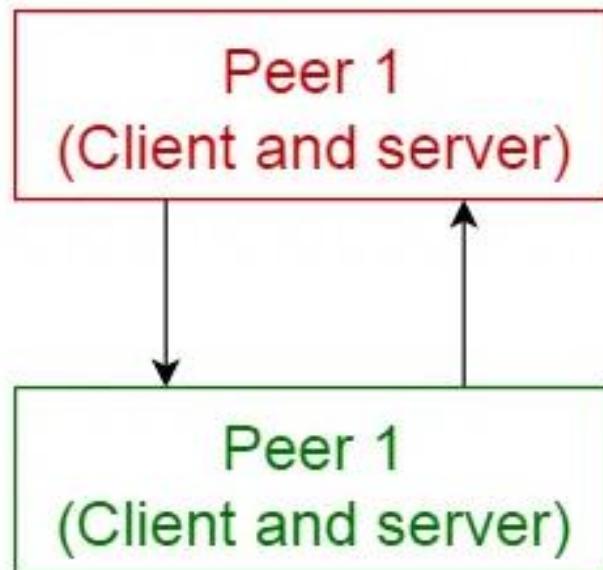
- **Conception générale** : Architecture logicielle courantes
 - Modèle de courtier (Broker pattern)



Le cycle de vie d'un logiciel : Conception

- ***Conception générale*** : Architecture logicielle courantes

Modèle Peer-to-Peer



Peer-to-peer pattern

Le cycle de vie d'un logiciel : Conception

- ***Conception générale*** : Architecture logicielle courantes
 - ***Modèle Evénement – Bus (Event-bus pattern)***

Ce modèle traite principalement des événements et comporte 4 composants principaux : ***Source d'événement, écouteur d'événement, canal et bus d'événement.***

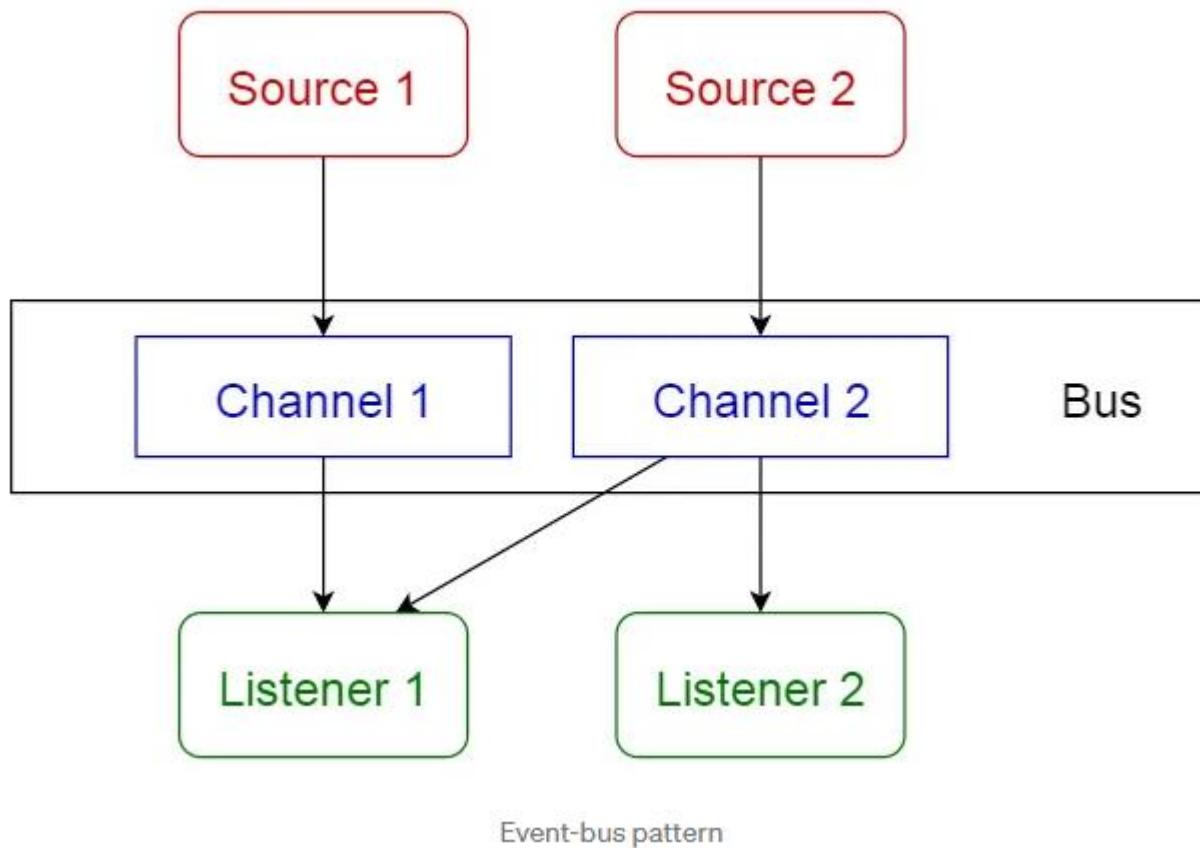
Les sources publient des messages sur des canaux particuliers sur un bus d'événements.

Les auditeurs s'abonnent à des chaînes particulières.

Les auditeurs sont informés par des messages qui sont publiés sur une chaîne à laquelle ils se sont déjà abonnés.

Le cycle de vie d'un logiciel : Conception

- **Conception générale** : Architecture logicielle courantes
- **Modèle Evénement – Bus (Event-bus pattern)**



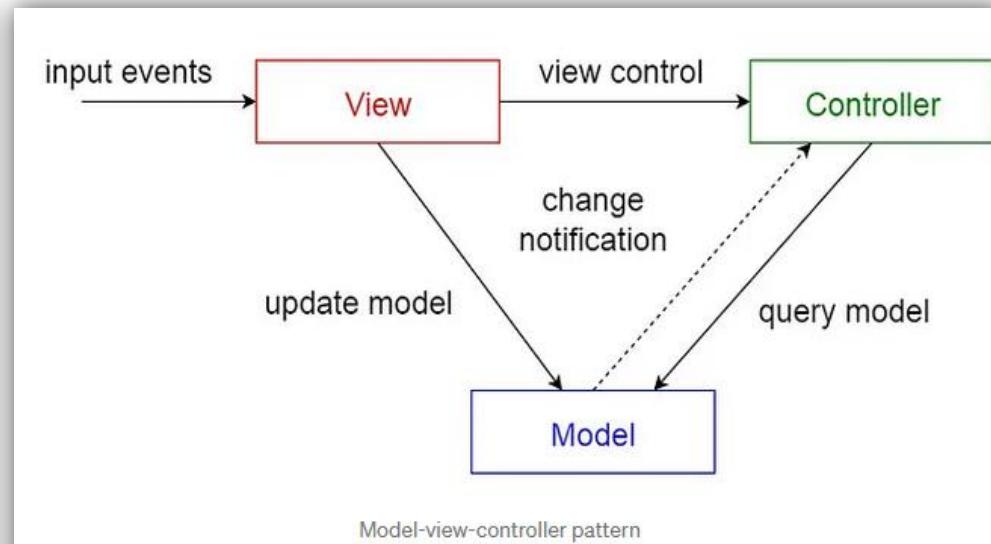
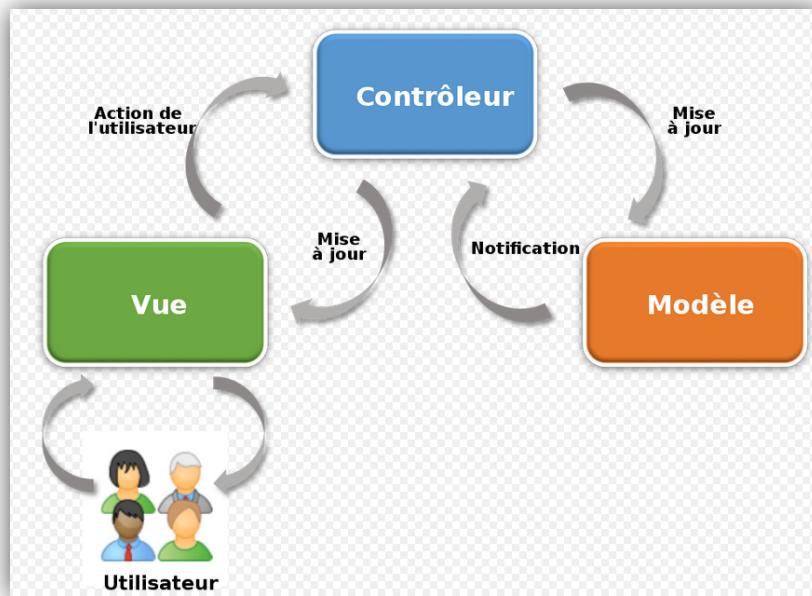
Le cycle de vie d'un logiciel : Conception

- **Conception générale** : Architecture logicielle courantes

- Modèle MVC (modèle-vue-contrôleur)

Divise une application interactive en 3 parties

- Un modèle (Model) contient les données à afficher ;
- Une vue (View) contient la présentation de l'interface graphique ;
- Un contrôleur (Controller) contient la logique concernant les actions effectuées par l'utilisateur.



Le cycle de vie d'un logiciel : Conception

- **Conception détaillé (detailed design) :**
 - La conception détaillée **affine** la conception générale.
 - **Décompose** les entités découvertes lors de la conception générale en entités plus élémentaires (des **composants** logiciels élémentaires).
 - Il faut ensuite **décrire chaque composant logiciel en détail** : *son interface, les algorithmes utilisés, le traitement des erreurs, ses performances, etc.*

L'ensemble de ces descriptions constitue **le document de conception détaillée**.

- Exemple : voir document «**Software Detailed Design Template** »

Le cycle de vie d'un logiciel : Tests

Tests

Le test est un ensemble d'activités utilisé pour tester le code source afin de découvrir (et corriger) les erreurs avant la livraison du logiciel client.

Objectif : avoir un maximum de garanties sur le bon fonctionnement du système.

Classification des tests :

1. Les modalités de test : **Statique / Dynamique**
2. Les méthodes de test : **Structurelle / Fonctionnelle**
3. **Manuel / Automatique**
4. Les niveaux de tests : **Unitaire / Intégration / Système / Non-régression**

Le cycle de vie d'un logiciel : Tests

Tests : statique ou dynamique

Test dynamique

on exécute le programme avec des valeurs en entrée et on observe le comportement

Test statique

relecture / revue de code et analyse automatique (vérification de propriétés, règles de codage, typage ...)

Le cycle de vie d'un logiciel : Tests

- ***Les niveaux des Tests :***

On peut classer les tests logiciels en différentes catégories :

Tests unitaire :

- **Un test unitaire** est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un programme.
- En programmation procédurale, une unité est une fonction.
- En programmation orientée objet, une unité est une classe.

Exemple : tester une classe en vérifiant que toutes ces méthodes n' engendrent pas d'erreur d'exécution.

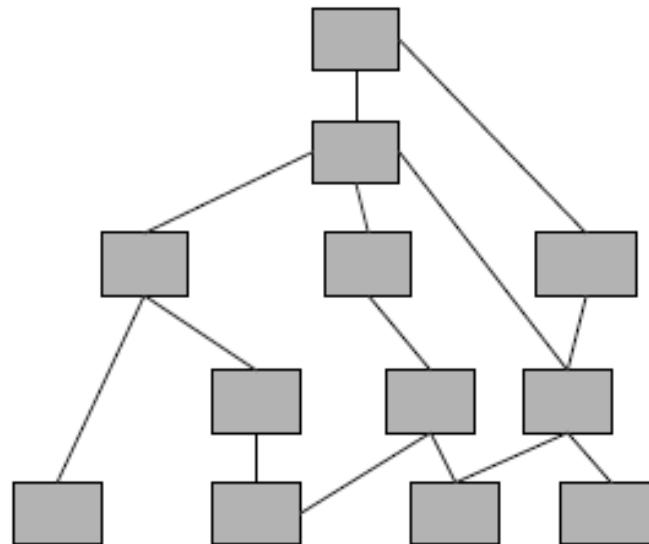
Le cycle de vie d'un logiciel : Tests

Tests d'intégration:

- ***Un test d'intégration*** est une procédure permettant de vérifier le bon fonctionnement d'un logiciel suite à l'assemblage de modules indépendants.

Difficultés :

Construire un **ordre permettant de tester graduellement l'assemblage des unités (problème de dépendance)**.



Le cycle de vie d'un logiciel : Tests

Tests de validation :

- s'assure que **le produit final** correspond à ce qui a été demandé (dans le document de spécification du logiciel) .
- Lors de ces tests, on valide la globalité du système, i.e. les fonctions offertes, souvent à partir de l'interface.
- Ce genre de tests généralement est manuel mais il peut aussi être automatisé.

Le cycle de vie d'un logiciel : Tests

Tests de non régression :

Un test consiste à vérifier que des modifications apportées au logiciel n'ont pas introduit de nouvelles erreurs.

Quand ?

- Dans la phase de maintenance,
- Après modification du code,
- ajout/suppression de fonctionnalités,
- après la correction d'une faute.

Le cycle de vie d'un logiciel : Tests

Les méthodes de test : Structurelle/Fonctionnelle

- ❑ Lors d'un **test fonctionnel**, on se réfère uniquement à l'observation de la sortie pour certaines valeurs d'entrée.
Aucune tentative d'analyser le code, (**Test boîte noire**)

- ❑ Lors d'un **test structurel**, on exploite la connaissance de la structure et la mise en œuvre du logiciel. (**Test boîte blanche**)
Possibilité de fixer finement la valeur des entrées pour sensibiliser des chemins particuliers du code;

Le cycle de vie d'un logiciel : Tests

Les méthodes de test : Structurelle/Fonctionnelle

- Exemple de **test fonctionnel**:

Code python de l'addition

```
def addition(x,y):  
    if y==0:  
        somme = x  
    else:  
        somme = x + y  
    return somme
```

Exemple de test fonctionnel

```
def test_addition():  
    x = 4  
    y = 5  
    valeur_attendue = 9  
    if addition(x,y) == valeur_attendue:  
        print("test réussi")  
    else:  
        print("test échoué")
```

Le cycle de vie d'un logiciel : Tests

Les méthodes de test : Structurelle/Fonctionnelle

- Exemple de **test structurelle**:

Code python pgcd

```
def pgcd(p,q):  
    while(p!=q):  
        if p>q:  
            p=p-q  
        else:  
            q=q-p  
    return p
```

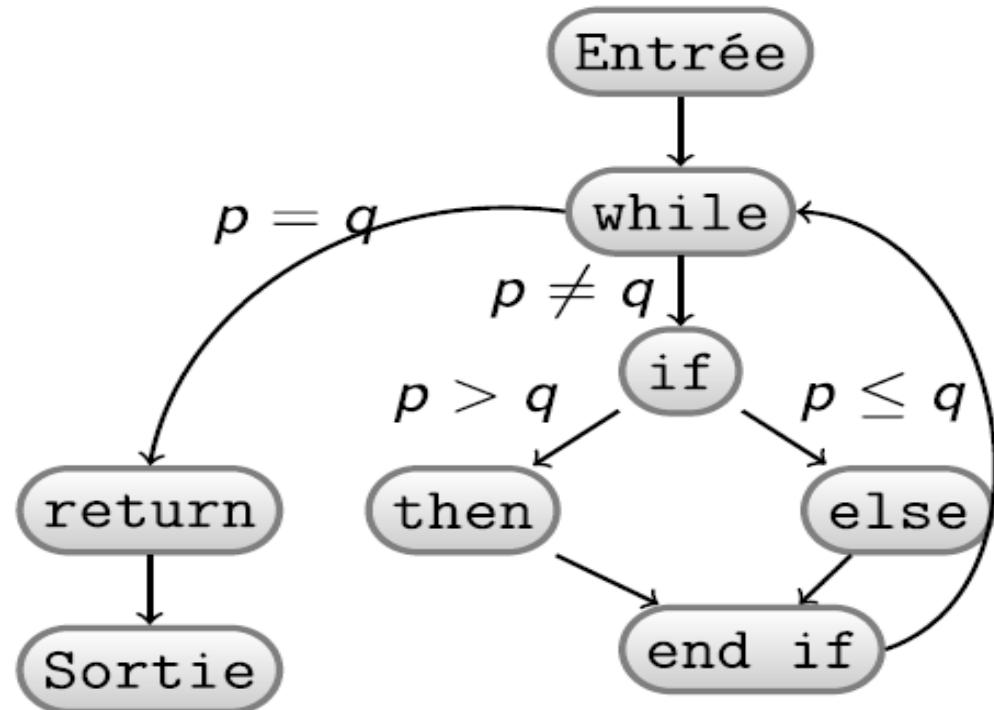


FIGURE – Graphe de contrôle

On cherche des jeux de tests à partir du graphe de contrôle afin de couvrir toutes les instructions, tous les chemins, ...

Le cycle de vie d'un logiciel : Tests

Les méthodes de test : Structurelle/Fonctionnelle

□ Test *structurelle*: Niveau de couverture

Tous les noeuds (C0) :

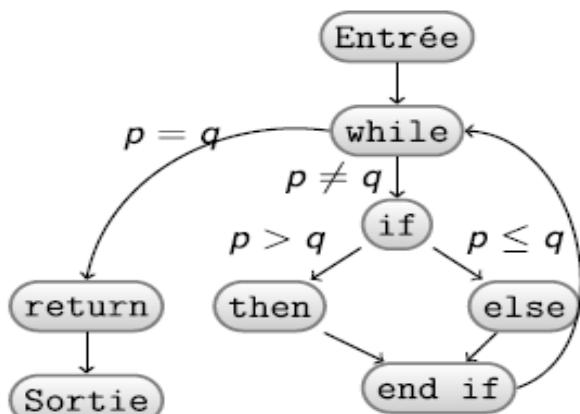
- (Entrée, while, if, then, return, Sortie)
- (Entrée, while, if, else, return, Sortie)

Tous les décisions (C1) :

- Chemins de (C0)
- (Entrée, while, return, Sortie)

Tous les 2-chemins (C-2) :

- Chemins de (C1)
- (E, while, if, then, while, if, then,return)
- (E, while, if, then, while, if, else,return)
- (E, while, if, else, while, if, then,return)
- (E, while, if, else, while, if, else,return)



Le cycle de vie d'un logiciel : Tests

Test manuel / test automatisé

Test manuel

- le testeur entre les données de test par ex via une interface;
- lance les tests;
- observe les résultats et les compare avec les résultats attendus;
- fastidieux, possibilité d'erreur humaine;
- ingérable pour les grosses applications;

Test automatisé

- Avec **le support d'outils qui déchargent le testeur** :
- du lancement des tests;
- de l'enregistrement des résultats;
- génération automatique de cas de test

Le cycle de vie d'un logiciel : Tests

Tests automatisés : Exemple d' Outils

JUnit

JUnit est un framework open source pour le développement et l'exécution de tests unitaires automatisables. Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications. Plus généralement, ce type de tests est appelé tests unitaires de non-régression.



Le cycle de vie d'un logiciel : Tests

Les Tests : une discipline à part entière.

Très importants dans le cycle de développement

Les tests unitaires :

Malheureusement ne couvrent pas tous les cas,

Restent néanmoins considérés comme une garantie de qualité

Même si vous couvrez 99% des cas, le 1% restant peut cacher un bug majeur !

« *Le test de programmes peut être une façon très efficace de montrer la présence de bugs mais est désespérément inadéquat pour prouver leur absence* » - *Edsger Dijkstra*

Le cycle de vie d'un logiciel : Tests

Le plan de teste :

Le plan de test peut être **considéré comme un manuel (document)**. Il **décrit les objectifs des tests** (ce qu'il faut vérifier et/ou valider), **la portée des tests** (ce qui sera et ne sera pas testé), ainsi que **le calendrier général voire détaillé des activités à effectuer** (comment le tester et quand le tester) **la procédure des tests** et les **Contraintes** (environnement de test particulier, installation particulière, intervention humaine spécifique) .

Plan des teste peut être décomposé :

- **Plan de Tests unitaire.**
- **Plan de Tests d'intégration.**
- **Plan de Tests de validation.**

Exemple de plan de teste : « Voir document »

Modèles de développement logiciels

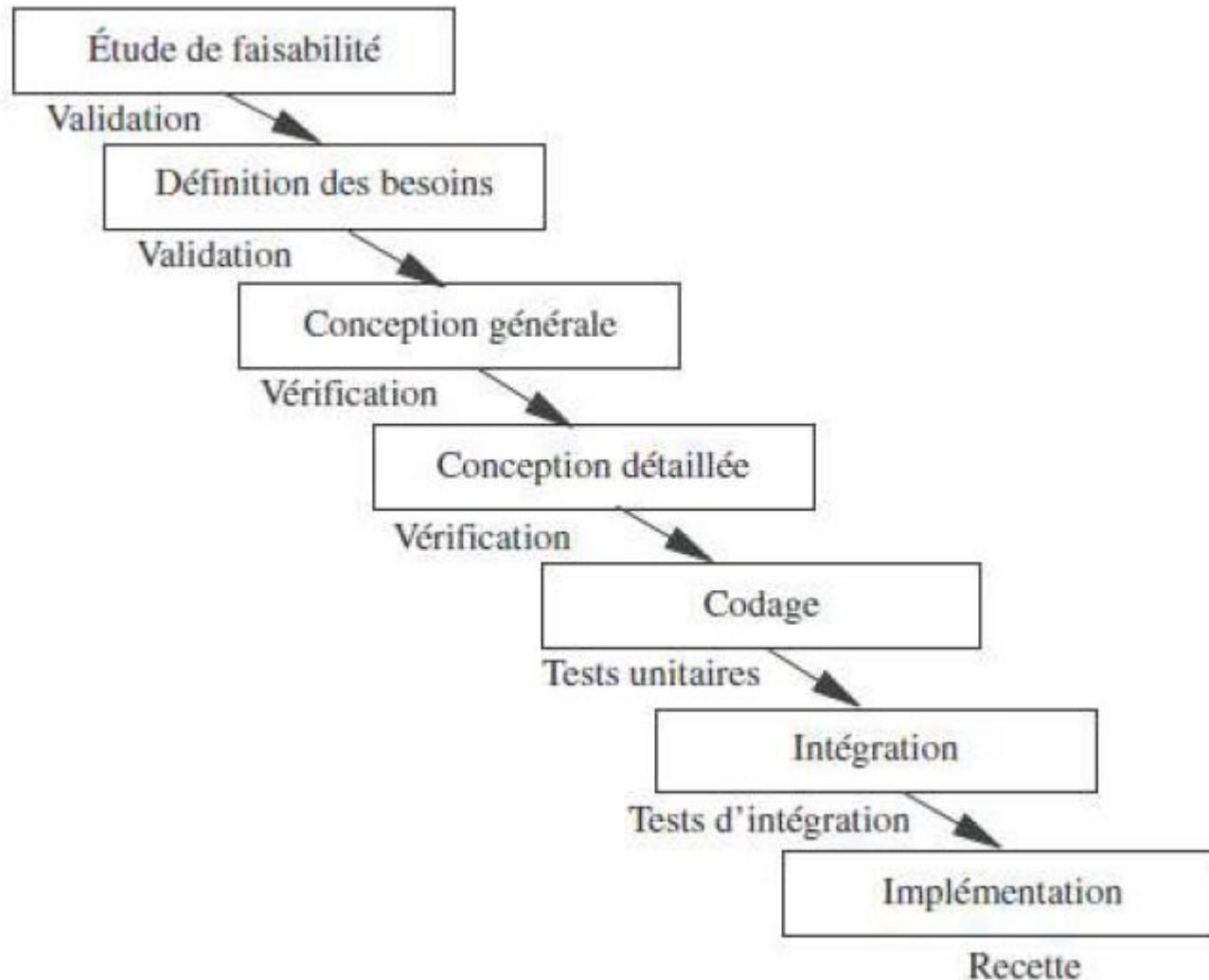
processus de développement (process models)

Modèles de développement logiciels

- Ces Modèles permettent d'exprimer clairement et définir le cycle de vie de développement du logiciel.
 - Ces Modèles contrôle le processus de développement et les documents qui accompagnent le logiciel .
 - Parmi ces modèles :
 - ❖ Le modèle de la cascade
 - ❖ Le modèle en V
 - ❖ Le modèle en Y
 - ❖ Le modèle spérale
- Les méthodes agiles**
- ❖ Scrum
 - ❖ Kanban
 - ❖ Xp

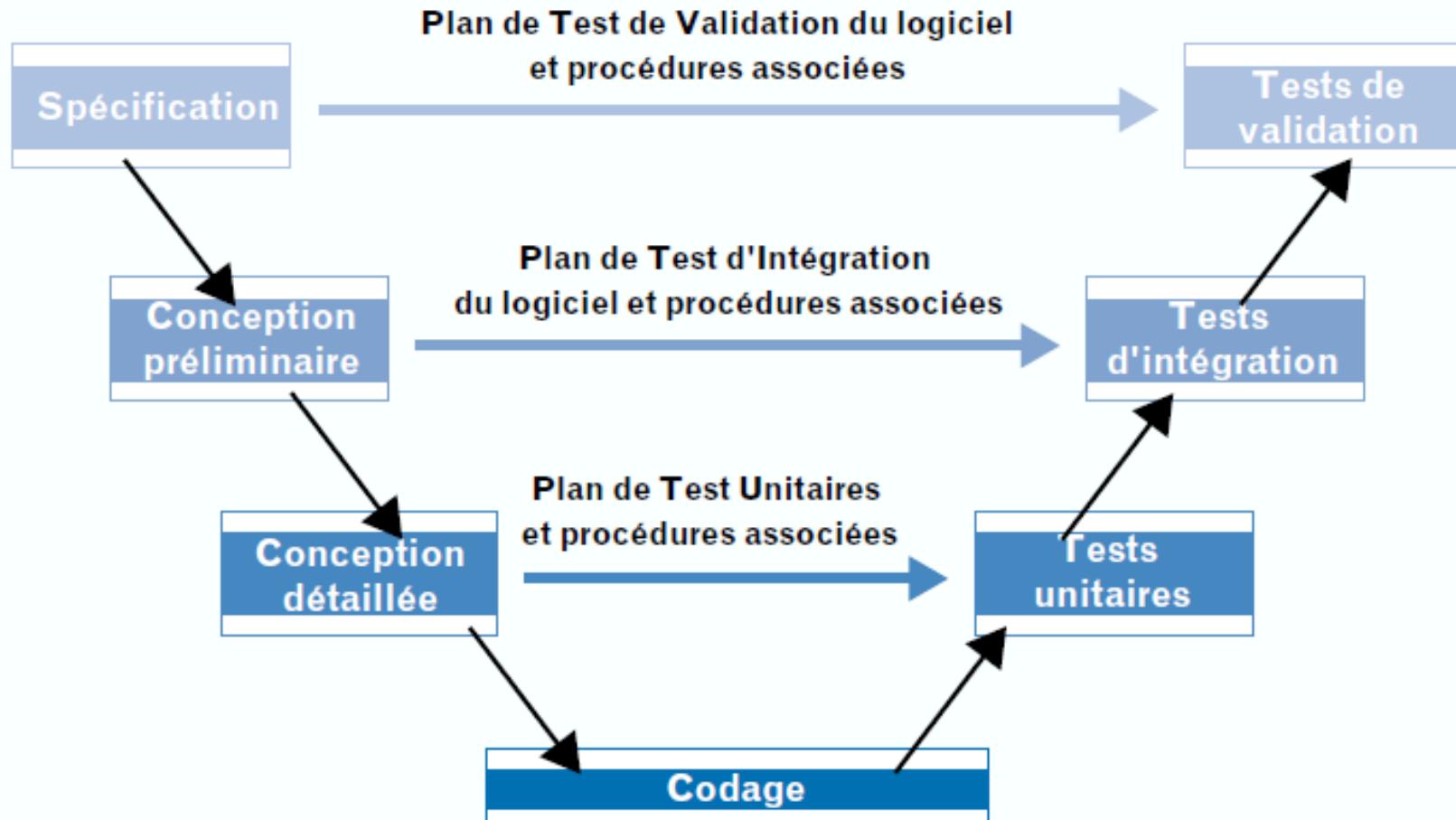
Modèles de développement logiciels

LE MODELE EN CASCADE (WATERFALL MODEL)



Modèles de développement logiciels

LE MODELE EN V



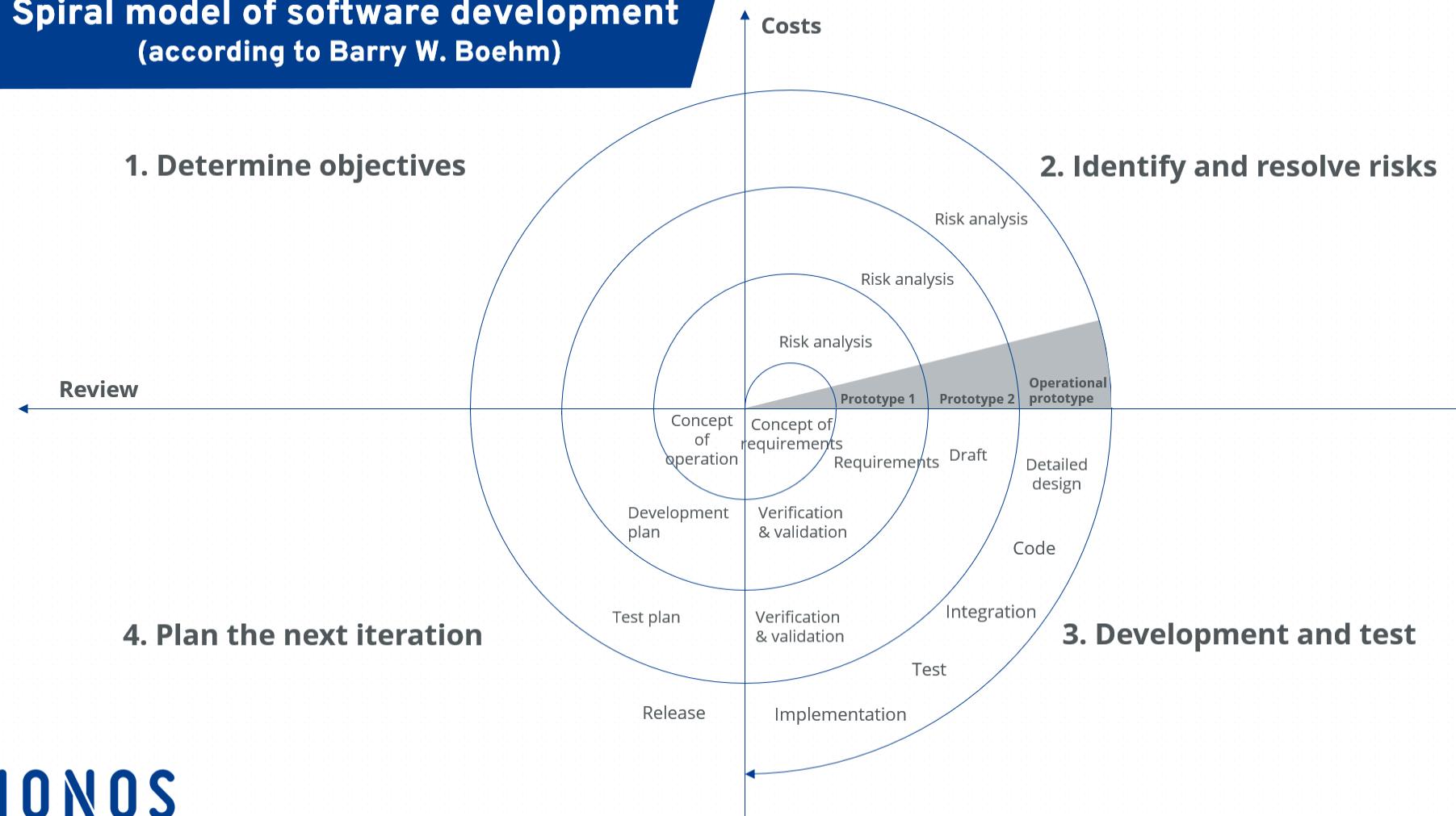
Modèles de développement logiciels

Étapes du cycle de vie	Principaux documents	Contrôles qualité
0 Préliminaires	<ul style="list-style-type: none"> - Cahier des charges - Appel d'offres - Contrat 	Revue de contrat
1 Planification	<ul style="list-style-type: none"> - Note de lancement - Plan d'assurance qualité - Plan de développement 	Revue de planification
2 Spécifications	<ul style="list-style-type: none"> - Dossier de définition des besoins/ Cahier des charges - Spécifications d'interface - Dossier tests de validation 	Revue de spécifications
3 Conception générale	<ul style="list-style-type: none"> - Dossier de conception générale - Dossier tests d'intégration 	Revue de conception générale
4 Conception détaillée	<ul style="list-style-type: none"> - Dossier de conception détaillée - Dossier tests unitaires 	Revue de conception détaillée
5 Codage	<ul style="list-style-type: none"> - Listings de programme - Documentation programme 	Revue de code
6 Tests unitaires	<ul style="list-style-type: none"> - Cahier de tests unitaires - Bilan des tests unitaires 	Revue de tests unitaires
7 Intégration	<ul style="list-style-type: none"> - Cahier de tests d'intégration - Bilan de l'intégration - Dossier d'installation - Dossier d'exploitation 	Revue d'intégration
8 Recette	<ul style="list-style-type: none"> - Cahier de tests de validation - Bilan de la recette 	Revue de recette
9 Installation/diffusion	<ul style="list-style-type: none"> - Cahier de l'installation 	Revue d'installation
Exploitation Maintenance (hors projet)	<ul style="list-style-type: none"> - Dossier de maintenance 	

Modèles de développement logiciels

LE MODELE EN SPÉRALE

Spiral model of software development
(according to Barry W. Boehm)



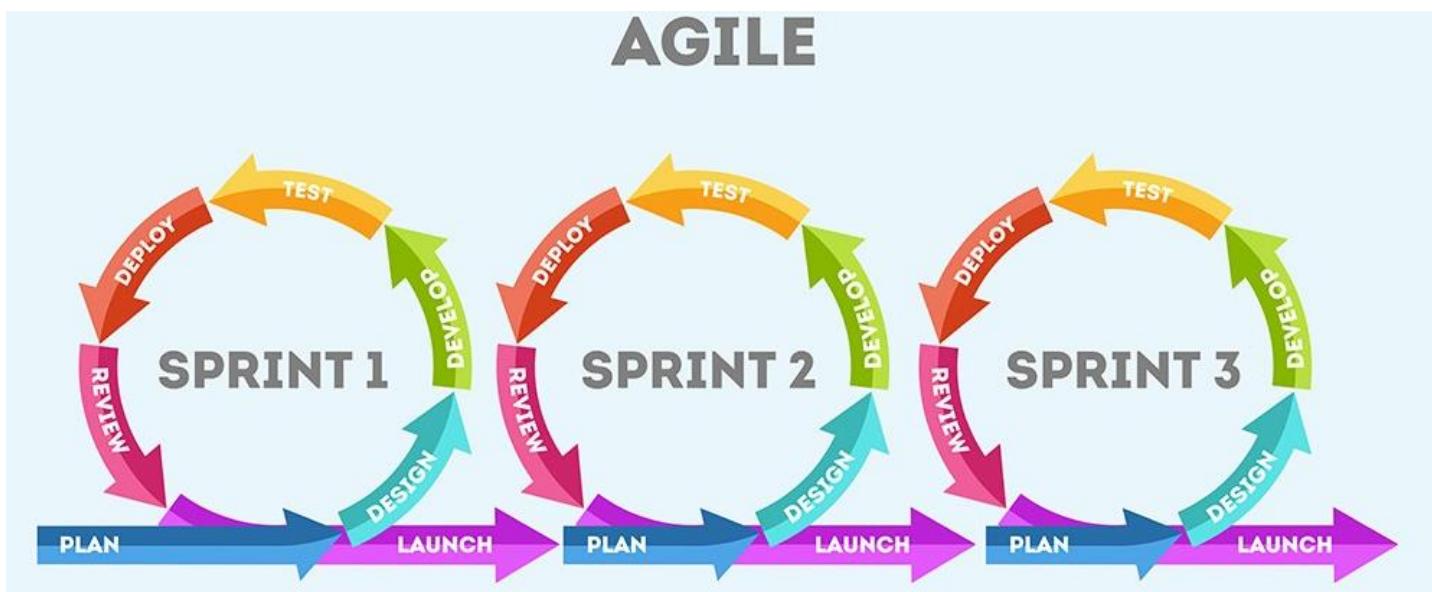
Modèles de développement logiciels

Approches Agiles

Approches Agiles

- Définition:

- **Découper en sous-parties** (ou sous-projets) autonomes (on parle également de **développement itératif**).
- Les parties (itérations) forment le projet dans sa globalité.
- Le projet est donc divisé en plusieurs sous-projets. Une fois l'objectif atteint, on passe au suivant, et ce jusqu'à l'accomplissement de l'objectif final. Cette approche est plus flexible. Puisqu'il est impossible de tout prévoir et de tout anticiper, elle laisse la place aux imprévus et aux changements.



Approches Agiles

- ***Bases (Principes de fondement)***

- Le Manifeste Agile est une déclaration rédigée par des experts en 2001 pour améliorer le développement de logiciels.
- Le Manifeste Agile propose **quatre valeurs** :
 - 1) *Les individus et leurs interactions* plus que les processus et les outils
 - 2) *Des logiciels opérationnels* plus qu'une documentation exhaustive
 - 3) *La collaboration avec les clients* plus que la négociation contractuelle
 - 4) *L'adaptation au changement* plus que le suivi d'un plan

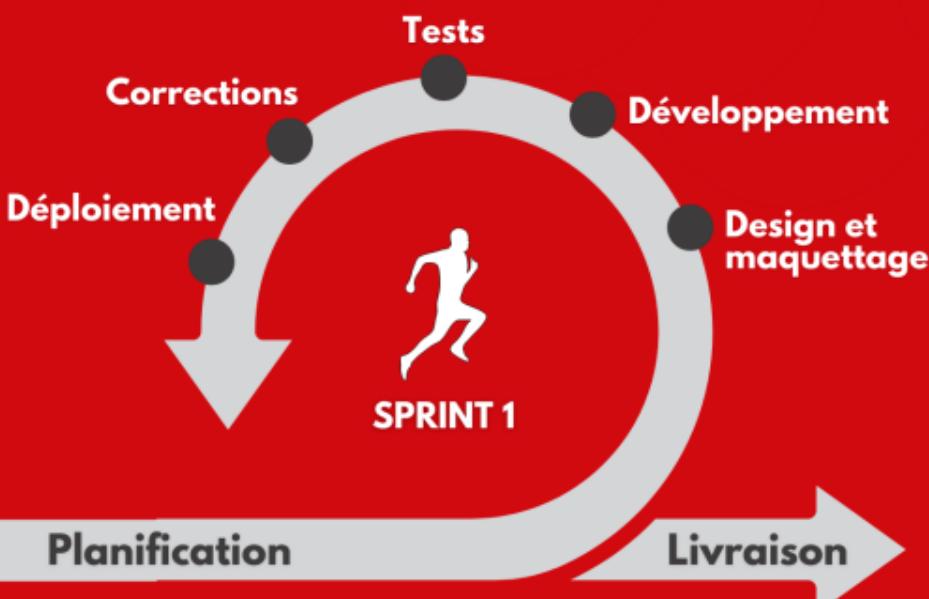
Approches Agiles

- *Principes sous-jacents au manifeste*

1. Privilégier la satisfaction client, c'est le principe le plus important.
2. Consentir au changement, même tardivement au cours du développement.
3. Fournir fréquemment des logiciels opérationnels.
4. Garantir une coopération quotidienne tout au long du projet entre les utilisateurs et les développeurs.
5. Construire les projets avec des équipes motivées.
6. Privilégier le dialogue, c'est le plus efficace pour transmettre l'information.
7. Produire un logiciel fonctionnel est la principale mesure du progrès.
8. Avancer le projet à un rythme soutenable et constant pour les sponsors, développeurs et utilisateurs.
9. Prêter une attention continue à l'excellence technique et au design.
10. Accorder la priorité à la simplicité, l'art d'exploiter au maximum le travail qui reste à faire.
11. Laisser les équipes s'auto-organiser afin d'obtenir les meilleures architectures, spécifications et conceptions.
12. Réfléchir à intervalles réguliers aux moyens de devenir plus efficace et ajuster en conséquence.

Approches Agiles vs Cascade

MÉTHODE AGILE



Les sprints se répètent jusqu'à livraison de l'application complète.

MÉTHODE EN CASCADE

Définition des spécifications fonctionnelles et techniques

Design et maquettage de l'application

Développement de l'application

Réalisation des tests unitaires et d'acceptance

Approches Agiles

Exemples d'approches Agiles :

- ❑ **XP eXtreme Programming** est une méthode de développement agile, orientée projet informatique et dont les ressources sont régulièrement actualisées.
- ❑ la méthode **Scrum** est une méthode agile de gestion de projets informatiques privilégiant la communication, et facilitant les réorientations opportunes.
- ❑ L'approche **Kanban** trouve son origine dans le mot japonais pour “panneau”. Elle nous vient des procédures de production de Toyota, appliquées à l'univers de la programmation logiciel. Cette approche consiste à croiser des tâches avec leurs états d'avancement, au sein d'une matrice en colonnes.
- ❑ **Agile Unified Process, Lean Software Development, SAFe.**

Approches Agiles

Exemples d'approches Agiles :



Approches Agiles : SCRUM

La **méthode Scrum** est une méthodologie de développement agile, utilisée dans le développement de logiciel, basée sur des **processus itératifs et incrémentaux**.

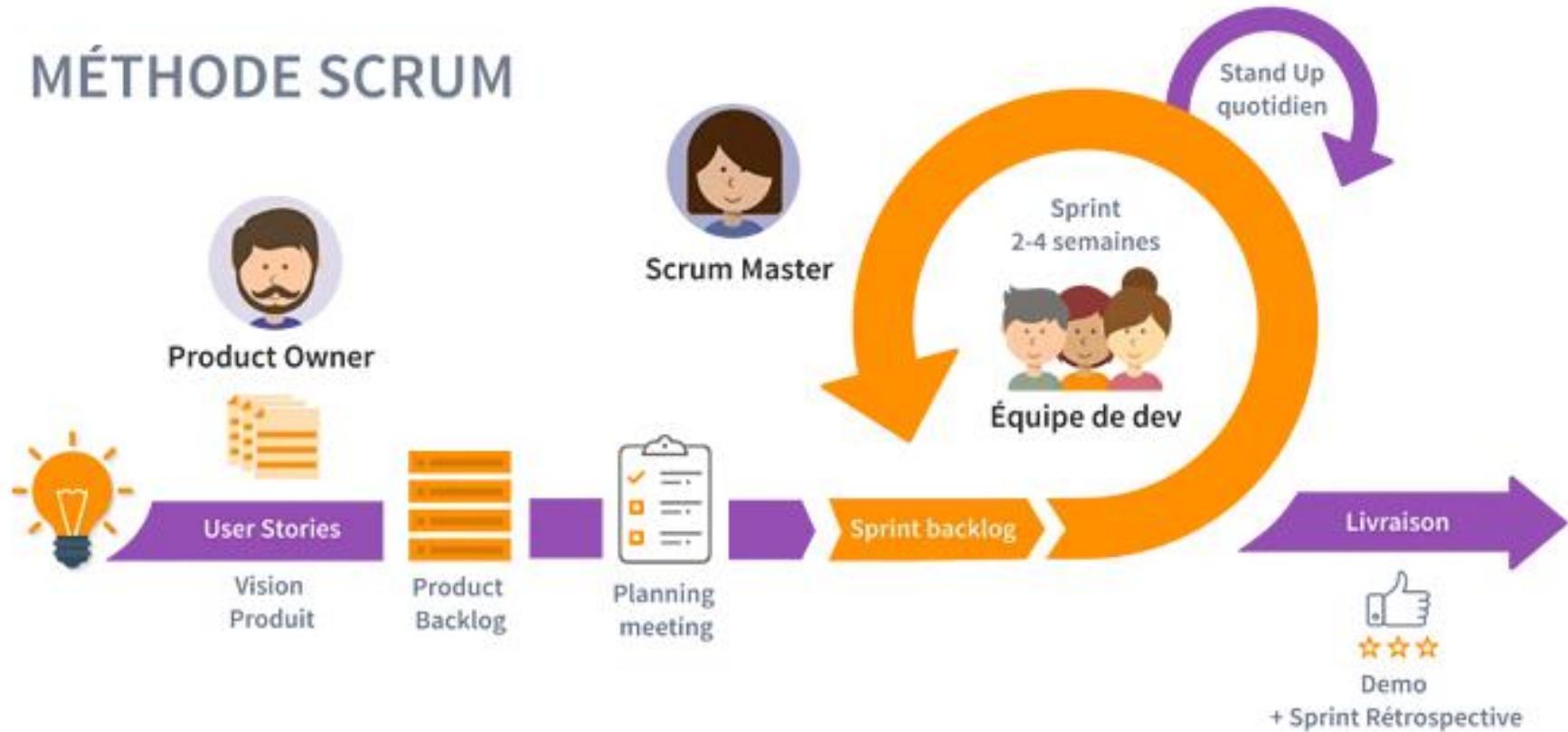
La méthode **Scrum est un framework (cadre) agile** adaptable, rapide, flexible et efficace qui est conçu pour apporter de la valeur au client tout au long du développement du projet.

L'objectif principal de Scrum est de **satisfaire les besoins du client** à travers un environnement de **transparence dans la communication, de responsabilité collective et de progrès continu**.

Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

MÉTHODE SCRUM

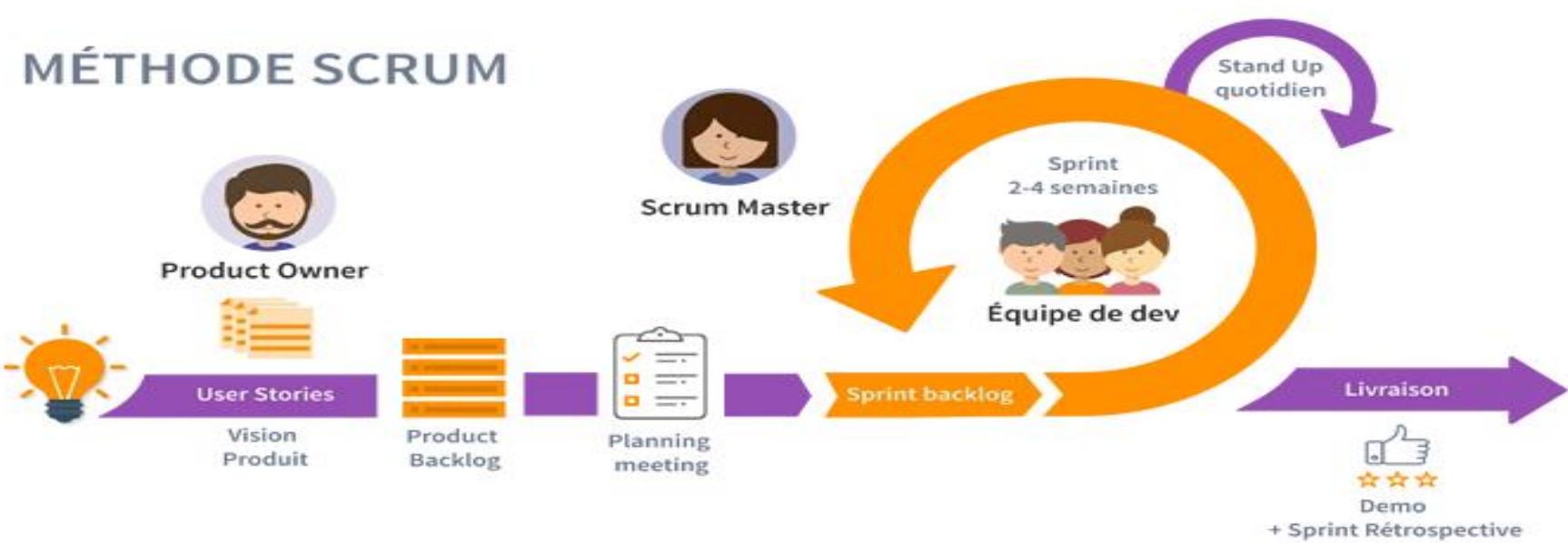


Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

La méthode Scrum est basé sur 3 phases :

- 1) Pre-Game(planification + architecture)
- 2) Game(Sprint + réunion Scrum)
- 3) Post-Game(démo + clôture)

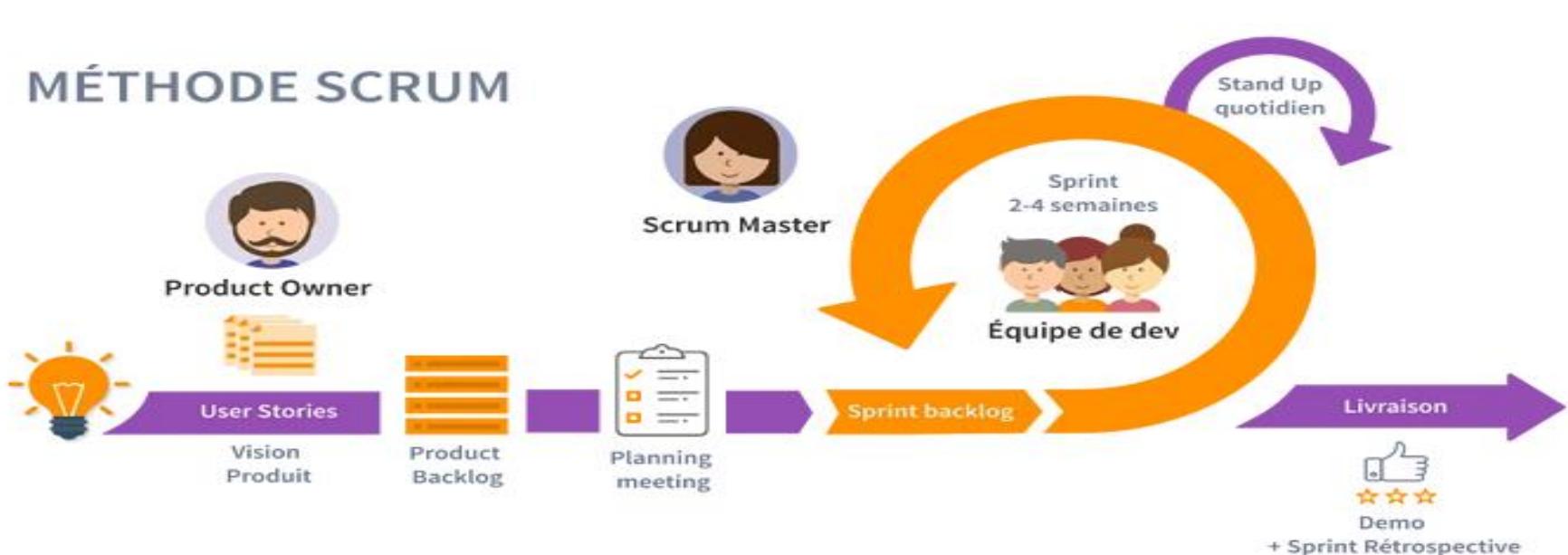


Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

Etape 1 : Le Product Backlog

- **Le Product Owner** rencontre le client et analyse son besoin. Il identifie toutes les fonctionnalités dont le produit devra être composé (**les user stories**) dans ce qui s'appelle le **Product Backlog**.
- Ce “ *cahier des charges* ” n'est pas fixé pour toujours, et pourra évoluer en fonction des besoins du client et l'avancement du projet.
- **L'équipe** décide de ce qu'elle peut faire et dans quel ordre le faire



Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

- **Product Owner :**

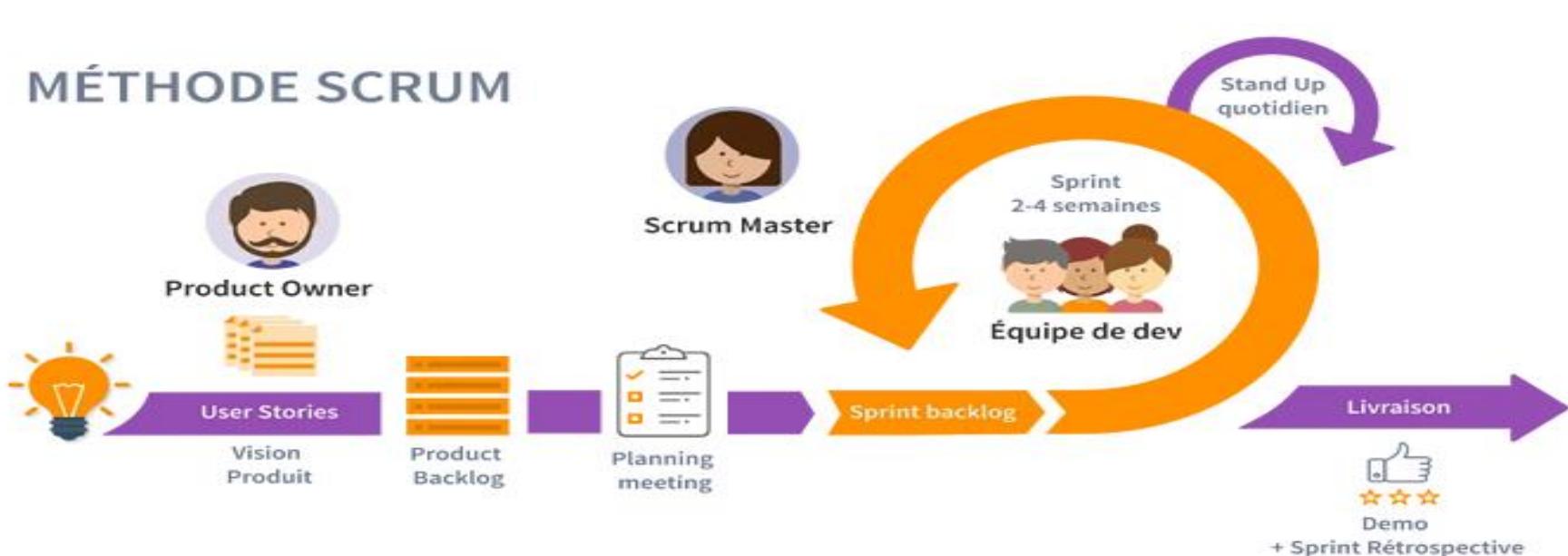
Lui c'est l'expert qui collabore avec le client.

Il définit à la suite des feedbacks clients les spécificités fonctionnelles du produit.

Il les priorise ensuite avec l'équipe.

Il valide les fonctionnalités développées.

Il endosse le rôle du client auprès de l'équipe.



Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

- **Étape 2 : Le sprint**

La méthode SCRUM se caractérise par une répartition de chacune des tâches à faire.

L'équipe trie les fonctionnalités et tâches qu'elle répartit dans des Sprint (durée de cycle de deux semaines).

Pendant ce cycle, l'équipe s'occupera par exemple uniquement de coder une fonctionnalité du produit qu'elle devra livrer à la fin de cette phase.



Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

- **Étape 2 : Le sprint - Planning meeting**

On organise avant chaque sprint une réunion de planification. C'est une sorte de négociation entre le product owner et l'équipe technique : le sprint planning meeting. Cette **réunion** permet de sélectionner dans **le product backlog** les exigences les plus prioritaires pour le client. Le **product owner** propose des **user stories**.

L'**équipe de développement** analyse les **user stories**, les traduit en **tâches techniques**, et regarde en combien de temps chacune des user stories pourra être réalisée. Après négociation et décision concernant les choses à faire, les tâches constituent **le sprint backlog**.

MÉTHODE SCRUM



Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

- **Equipe de développement:**

Dans un projet de développement agile, la mission principale de l'équipe de développement au sein d'une équipe Scrum est de fournir un livrable de manière incrémentale et itérative.

Dans un premier temps, elle fournit un MVP (produit minimum viable).

La taille conseillée pour une équipe Scrum se situe **entre 3 et 9 membres**, hors *Product Owner et Scrum Master*, d'après le Scrum Guide. En dessous de 3 membres, l'équipe n'est pas assez diversifiée et peut rencontrer des obstacles. Au-dessus de 9 membres, la coordination et la communication se complexifient et l'équipe risque de perdre en agilité.

Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

- **Étape 2 : Le sprint - Le Daily SCRUM (stand up quotidien)**
 - **Cette réunion** a lieu tous les jours. Sa **durée maximale est de 15 minutes**. Son objectif est de permettre à l'équipe Scrum de synchroniser et d'ajuster son action pour les 24 heures et de répartir les tâches à traiter.

Dans un Daily Scrum on parle de trois choses :

- Ce qu'on a fait hier ?
- Quels problèmes on a rencontrés ?
- Que va-t-on faire aujourd'hui ?

Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

- **Étape 2 : Le sprint -comment on s'organise dans le sprint ?**

Utiliser un tableau et le diviser en trois colonnes : *à faire, en cours, réalisé.*

- **En début de sprint,** on recense chacune des petites tâches qui seront nécessaires pour réaliser la fonctionnalité en question du produit.
- On représente chaque tâche par un post-it de couleur. Elles se trouvent dans la première colonne : à faire.
- Au fur et à mesure qu'elles seront réalisées, elles seront bougées vers la droite.
- L'équipe a établi combien de temps prendrait chaque tâche. Il est ainsi facile de voir pendant un sprint si l'équipe tiendra ses timings et ces objectifs.
- Cela force à tendre vers l'efficacité plus que la perfection.

Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

- **Étape 3 : Sprint Review**

La sprint review, ou revue de sprint, est une **réunion** qui a lieu en **fin de sprint**, où **l'équipe** Scrum présente aux parties prenantes l'incrément de sprint et le travail réalisé en cours de sprint. Cela permet de recueillir le feedback des utilisateurs en vue d'améliorer le produit et de maximiser sa valeur.

Cette réunion est un moment d'échange, de partage et de collaboration entre l'équipe Scrum d'un côté et **les parties prenantes du projet** de l'autre.

C'est le moment parfait pour vérifier la progression vers l'objectif de produit, et discuter des items à prioriser dans le product backlog, en fonction du retour des parties prenantes.

Participants : Product Owner + Scrum Master + Équipe de développement + Parties prenantes (Stakeholders)

Approches Agiles : SCRUM

Le fonctionnement de la méthode Scrum

- **Étape 4 : Sprint Retrospective (Rétrospective de Sprint)**

Une rétrospective de sprint est une réunion qui se tient après une revue de sprint et avant le début du sprint suivant. Il sert principalement d'occasion pour les membres de l'équipe de réfléchir à leurs efforts de collaboration. C'est également le moment pour les équipes d'identifier comment appliquer les connaissances passées au sprint à venir.

Participants :

Toute l'équipe - les développeurs, le Scrum master et le Product Owner - se réunit pour revenir sur et analyser le sprint qu'ils viennent de terminer. C'est le moment idéal pour l'équipe de réfléchir à ses pratiques, d'échanger sur ce qu'elle pense avoir fonctionné et sur ce qui doit être amélioré dans le travail de chacun.

Approches Agiles : SCRUM

Autres concepts SCRUM ...

Approches Agiles : SCRUM

L'ÉQUIPE SCRUM



Scrum Master (SM)



Product Owner (PO)



Équipe de développement

Approches Agiles : SCRUM

Le Scrum Master (SM) :

- est le facilitateur de l'équipe Scrum. Il est chargé de s'assurer que l'équipe adhère au processus et aux principes Scrum et supprime tout obstacle pouvant survenir dans le quotidien des membres.
- Le Scrum Master n'est pas un manager ou un chef de projet, mais plutôt un leader-serviteur qui aide l'équipe à s'auto-organiser et à atteindre ses objectifs de sprint.
- Le guide Scrum évoque ce rôle de Scrum Master sous la casquette du "garant de la méthodologie Scrum". C'est la raison pour laquelle il est recommandé qu'il soit formé avant de prendre ses fonctions et n'aura de cesse de continuer d'apprendre et d'approfondir des concepts liés à l'Humain.
- Il devra accompagner la bonne application du cadre Scrum par l'ensemble de l'équipe.

Approches Agiles : SCRUM

User Story:

Une user story typique pourrait ressembler à ceci :

"En tant que <rôle d'utilisateur>, je souhaite <fonctionnalité> afin de pouvoir <recevoir des avantages>."

Exemple :

"Je suis Brad, un utilisateur du portail de réservation de vols en ligne, et je veux savoir si mon billet est réservé pour ne pas avoir à réessayer de réserver."

Approches Agiles : SCRUM

EPICS

Les EPICS « épopées » sont utilisées dans le développement logiciel agile **pour classer vos user stories et donner une structure à votre backlog.**

Les user stories requises pour parvenir à accomplir un EPIC pourraient être réparties sur plusieurs sprints.

Un EPIC est une grande user story qui est trop grosse pour tenir dans un sprint. Cette histoire de haut niveau est généralement divisée en plus petites, chacune pouvant être complétée en un sprint. En ce sens, un EPIC est une collection de User Story avec un objectif uniifié.

Approches Agiles : SCRUM

Product Backlog :

- Liste des fonctionnalités du produit.
- Au démarrage du développement d'un produit agile, le MVP va être découpé en petites fonctionnalités ou tâches à réaliser pour faciliter sa construction.
- Le Product Backlog est une sorte de réservoir regroupant l'ensemble des fonctionnalités du produit. Les tâches doivent y être ordonnées avec discernement en fonction de la priorité dans laquelle elles doivent être réalisées.

Sprint Backlog :

- Planification des éléments du Product Backlog à mettre en oeuvre lors du Sprint pour livrer l'incrément de produit doté des fonctionnalités requises pour cette étape.
- Le cadre méthodologique SCRUM divise le calendrier d'une équipe en cycles qui se répètent, nommés Sprint.
- Le Sprint Backlog est une vue en temps-réel, très visible du travail que l'Équipe planifie d'accomplir durant le Sprint et il appartient uniquement à l'Équipe de Développement.

L'incrément de produit :

- Durant chaque Sprint, l'équipe de développement réalise un incrément de produit.
- Un Sprint démarre lorsque le précédent est terminé. Il s'agit d'un processus incrémental.