Université Mohammed V
Faculté des Sciences, Rabat
Département d'Informatique

QUALITÉ DES LOGICIELS

Master d'Informatique Appliquée au Développement et à l'Offshore

Préparer par :
Pr. Soumia ZITI

Année universitaire 2011/2012

Introduction

Ce cours concerne **la qualité des logiciel,** nous nous intéressons, d'une part, au développement de logiciel et son suivi tout au long de son exploitation (cette étape est appelée cycle de vie). Et d'autre part, de l'assurance qualité logiciel suivant des tests de produit avant et après l'exécution du code source. La Génie Logicielle devient une discipline très importante pour plusieurs raisons per exemple la **complexité des logiciels** de plus en plus grande et nous avons besoin des techniques de GL pour l'améliorer. Cette discipline suppose des équipes de développeurs et non quelques spécialistes, une évolution des méthodes de travail et aussi un apprentissage à réutiliser l'existant (Ken Orr affirme que : 80% des développements existent déjà et que 20% seulement sont spécifiques)

1. Définition de génie logicielle:

La Génie Logicielle (GL) décrit l'ensemble de méthodes (spécification, conception, test…) , techniques et outils (Atelier de Génie logiciel) nécessaires à la production de logiciels de qualité industrielle.

Les objectifs de la GL est de :

- Satisfaire les utilisateurs, en produisant des logiciels adaptés aux besoins ;
- Satisfaire les gestionnaires, en réduisant les coûts de maintenance
- Satisfaire les chefs de projet, en rendant les logiciels productifs dans un délai raisonnable.

Le but aussi est d'éviter certains problèmes voir des catastrophes concernant les risques de logiciels utilisés massivement dans des domaines critiques comme le pilotage, les système sécuritaire, l'industrie nucléaire (réservation de ticket, pilotage aéronautique, diagnostique médicale, opération financière …)

2. Logiciel :

Un logicel est composé :

- Des documents de gestion de projet,
- Une Spécifications décrivant la liste des fonctions à remplir par le logiciel, les facteurs qualité du logiciel (portabilité, évolutivité, robustesse, . . .), les contraintes (performances temporelles et spatiales, . . .) et les interfaces avec son environnement
- Une conception décrivant la découpe de la spécification en modules (ou objets), la description les interfaces entre ces modules (ou objets) et la description des algorithmes mis en place
- Un code source et un exécutable

Par contre un produit logiciel est composé de programmes sources et machines, des procédures et des ensembles de données enregistrées. Ce produit est destiné à un client vec besoins, désir et exigence qui passe une commande de logiciel et réalisé par un fournisseur de produit ou de service qui doit donner satisfaction, éviter les réclamation, augmenter le nombre de clients...

3. Caractéristique de Logiciel

produit unique : conçu et fabriqué une seule fois, reproduit
Inusable : Défauts pas dus à l'usure mais proviennent de sa conception
Complexe :Le logiciel est fabriqué pour soulager l'humain d'un problème complexe ; il est donc

par nature complexe
Invisible :Fabrication du logiciel est une activité purement intellectuelle avec une difficulté de perception de la notion de qualité du logiciel
Techniques non matures : Encore artisanal (fait à la main) malgré les progrès

4. Plan logiciel :

Ce plan est composé de

- La description du logiciel à réaliser en différents niveaux de produits (programmes et documents)
- Les moyens matériels et/ou logiciel à mettre à disposition ou à réaliser (Méthodes, Techniques, Outils)
- Le découpage du cycle de vie en phases, la définition des tâches à effectuer dans chaque phase et l'identification des responsables associés
- Les supports de suivi de l'avancement (Planning et calendriers)
- Les moyens utilisés pour gérer le projet
- Les points clés avec ou sans intervention du client

5. Constat sur le développement logiciel

- Le coût du développement du logiciel dépasse souvent celui du matériel,
- Les coûts dans le développement du logiciel :
  - 20% pour le codage et la mise au point individuelle
  - 30% pour la conception
  - 50% pour les tests d'intégration
- Durée de vie d'un logiciel de 7 à 15 ans
- Le coût de la maintenance évolutive et corrective constitue la part prépondérante du coût total
- Plus de la moitié des erreurs découvertes en phases de tests proviennent d'erreurs introduites dans les premières étapes
- La récupération d'une erreur est d'autant plus coûteuse que sa découverte est tardive

6. Étapes de développement de logiciel

6.1. Analyse du problème

comprendre et recenser les besoins
Spécification (par exemple cahier des charges)

6.2. Conception du logiciel

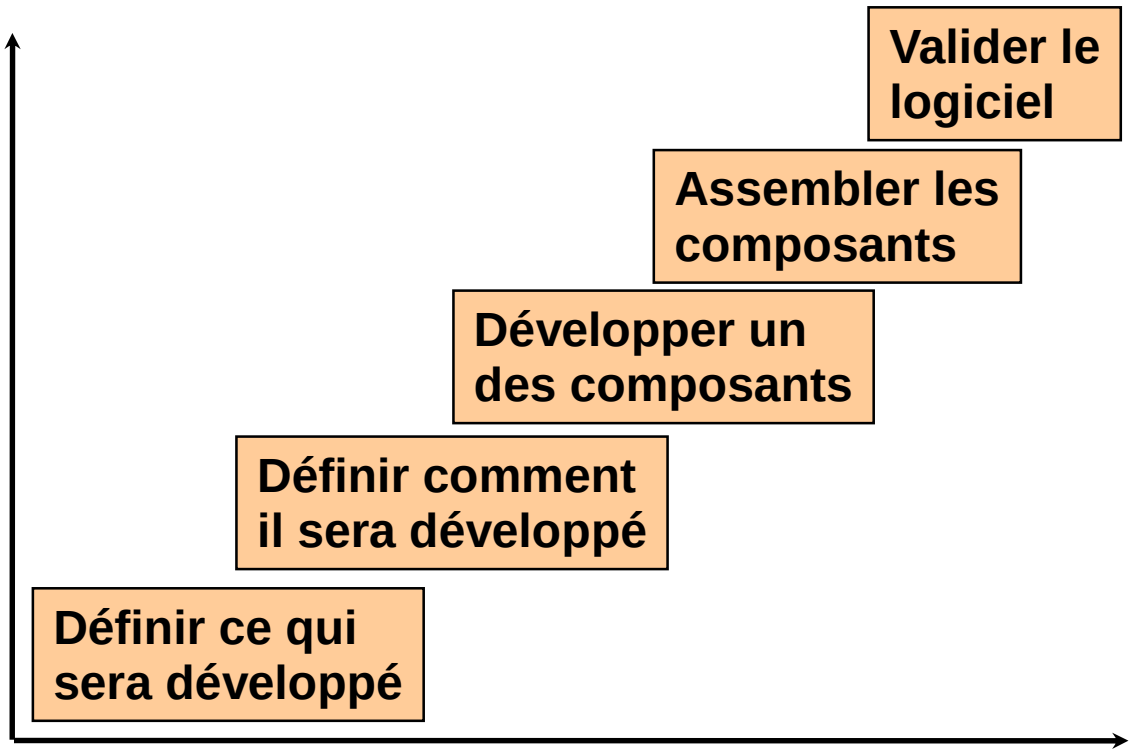
Préliminaire : il faut éclater le logiciel en sous-parties, définir les interfaces entre ces sous-parties et définir l'architecture du logiciel

Détaillée : en précisant l'architecture des sous-parties

6.3. Implantation

- codage, intégration, tests

L'organisation de ces activités et leurs enchaînements définit la cycle de développement logiciel : à partir de la formulation des besoins il faut aboutir à un produit logiciel fini en passant par des étapes intermédiaires mais se posent les questions sur la manière de procéder



7. Exemples de désastres historiques

1988 abattage d'un Airbus 320 par l'USS Vincennes : affichage cryptique et confusion du logiciel de détection

1991 échec de missile patriot : calcul imprécis de temps dû à des erreurs arithmétiques

London Ambulance Service Computer Aided Despatch System : plusieurs décès

Le 3 Juin 1980, North American Aerospace Defense Command (NORAD) rapporta que les U.S. étaient sous attaque de missiles

Echec du premier lancement opérationnel de la navette spatiale dont le logiciel d'exploitation temps réel consiste en environ 500,000 LOC : problème de synchronisation entre les ordinateurs de contrôle de vol

Réseau téléphonique longue distance d'AT&T : Panne de 9 heures provoqué par un patch de code non testé

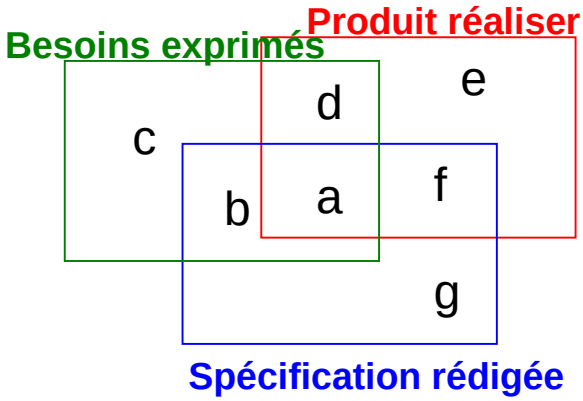
7.1. Exemple type : Ariane 5

- C'est le lanceur successeur de Ariane 4 pour le lancement de satellites en orbite terrestre.
- 37 seconde après décollage réussi, elle explose en vol
- Des information incorrectes avaient été transmises mettant le système de contrôle en défaut
- L'expertise a montré que le défaut provenait de la tentative de conversion d'un nombre flottant 64-bit en entier signé 16-bit et qu'il n' y avait pas de gestion d'exceptions implantées
- Cette partie provenait d'Ariane 4 et du fait de l'accélération et de la vitesse moins importante, le problème n'était jamais apparu ni en vol réel ni en test

Ces erreurs prouvent que la complexité des systèmes informatisés peuvent conduire à des catastrophes et que les logiciels doivent être conçus avec méthode.

Problème de conformité aux besoins

a) Satisfaction
b) Non-conformité
c) Insatisfaction client
d) Satisfaction hasardeuse
e) Prestations inutiles hors spécification
f) Spécifications et prestations inutiles
g) Spécification inutiles



Cycle de vie de développement logiciel

1. Définition

- Le « **cycle de vie d'un logiciel** » (*software lifecycle*), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition.
- Ce découpage permet de définir des jalons intermédiaires permettant la **validation** du développement logiciel (conformité du logiciel avec les besoins exprimés), et la **vérification** du processus de développement (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.

2. Activités du cycle de vie

2.1. Définition des objectifs :

consistant à définir la finalité du projet et son inscription dans une stratégie globale.

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.
Conception générale : Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.

Conception détaillée : consistant à définir précisément chaque sous- ensemble du logiciel.
Codage (Implémentation ou programmation) : soit la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.
Tests unitaires : permettant de vérifier individuellement que chaque sous-ensemble du logiciel est implémentée conformément aux spécifications.
Intégration : dont 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 : visant à produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs.
Mise en production : déploiement sur site du logiciel.
Maintenance : comprenant toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

2.2. Phase : définition des besoins (Pourquoi?)

Activités (du client, externe ou interne) : il faut effectuer des consultations d'experts et d'utilisateurs potentiels et réaliser des questionnaire d'observation de l'usager dans ses tâches :

- Pourquoi faut-il réaliser un logiciel?
- Y a-t-il de meilleur alternatives?
- Le logiciel sera-t-il satisfaisant pour les utilisateur?
- Y a-t-il un marché pour le logiciel?
- A-t-on le budget, le personnel, le matériel nécessaire?

Productions :

- Cahier des charges (les « exigences »)
- Fonctionnalités attendues
- Contraintes non fonctionnelles
- Qualité, précision, optimalité, ...
- temps de réponse, contraintes de taille, de volume de traitement, ...

2.3. Phase : Analyse des besoins / spécification (Quoi?)

Activités : Définir précisément les fonctions que le logiciel doit réaliser ou fournir en définissant les spécifications générales : Objectifs, contraintes (utilisation de matériels et outils existants) et généralités à respecter ; les spécifications fonctionnelles : description détaillé des fonctionnalités du logiciel et les spécifications d'interface : description des interfaces du logiciel avec le monde extérieure (hommes, autres logiciels, matériels...)

Productions :

- Dossier d'analyse
- Ébauche du manuel utilisateur
- Première version du glossaire

2.4. Phase : Codage et tests unitaires (Comment ?)

Activités : Définir la structure ou l'architecture du logiciel (décomposition en modules) et la spécification des interfaces des modules, ensuite réaliser la conception détaillée du logicielen implémentant les algorithmes de chacune des procédures des modules et les tester indépendamment les uns des autres

Productions :

- Modules codés et testés
- Documentation de chaque module
- Résultats des tests unitaires
- Planning mis à jour

2.5. Phase : Intégration

Activités : Les modules doivent être testés et intégrés suivant un plan d'intégration, de plus le test de l'ensemble (modules intégrés) doit être réalisé conformément au plan de tests. Il faut intégrer les différents modules avec validation / vérification de l'adéquation de l'implantation, de la conception et de l'architecture avec le cahier de charge (acceptation)

Productions :

- Logiciel testé
- Jeu de tests de non-régression
- Manuel d'installation : guide dans les choix d'installation
- Version finale du manuel utilisateur
- Manuel de référence : liste exhaustive de chaque fonctionnalité

2.6. Phase : Qualification

Activités :Déploiement du logiciel chez le client et tests avec des sous ensemble d'usager choisi. Dans cette étape il faut réaliser les tests en vraie grandeur dans les conditions normales d'utilisation et intégrer le logiciel dans l'environnement extérieur (autres logiciels, utilisateurs), ensuite vérifier que le logiciel développé répond aux besoins exprimés dans la phase de spécifications des besoins.

Productions :

- diagnostic OK / KO

2.7. Phase : Maintenance

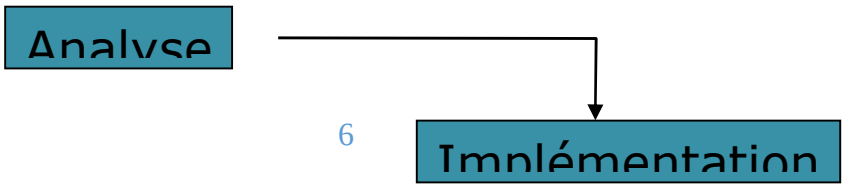
Activités : Utilisation en situation réelle, retour d'information des usagers, des administrateurs, des gestionnaires…Il faut organiser une formation de l'utilisateur avec une assistance technique et effectuer si nécessaire des maintenance correctives en cas de non conformité aux spécifications, avec détection et correction des erreurs résiduelles ; maintenance adaptative concernant la modification de l'environnement (matériel, fournitures logicielles, outil, …) ou maintenance évolutive à cause de changement des spécifications fonctionnelles du logiciel

Productions :

- Logiciel corrigé
- Mises à jour, patch
- Documents corrigés.

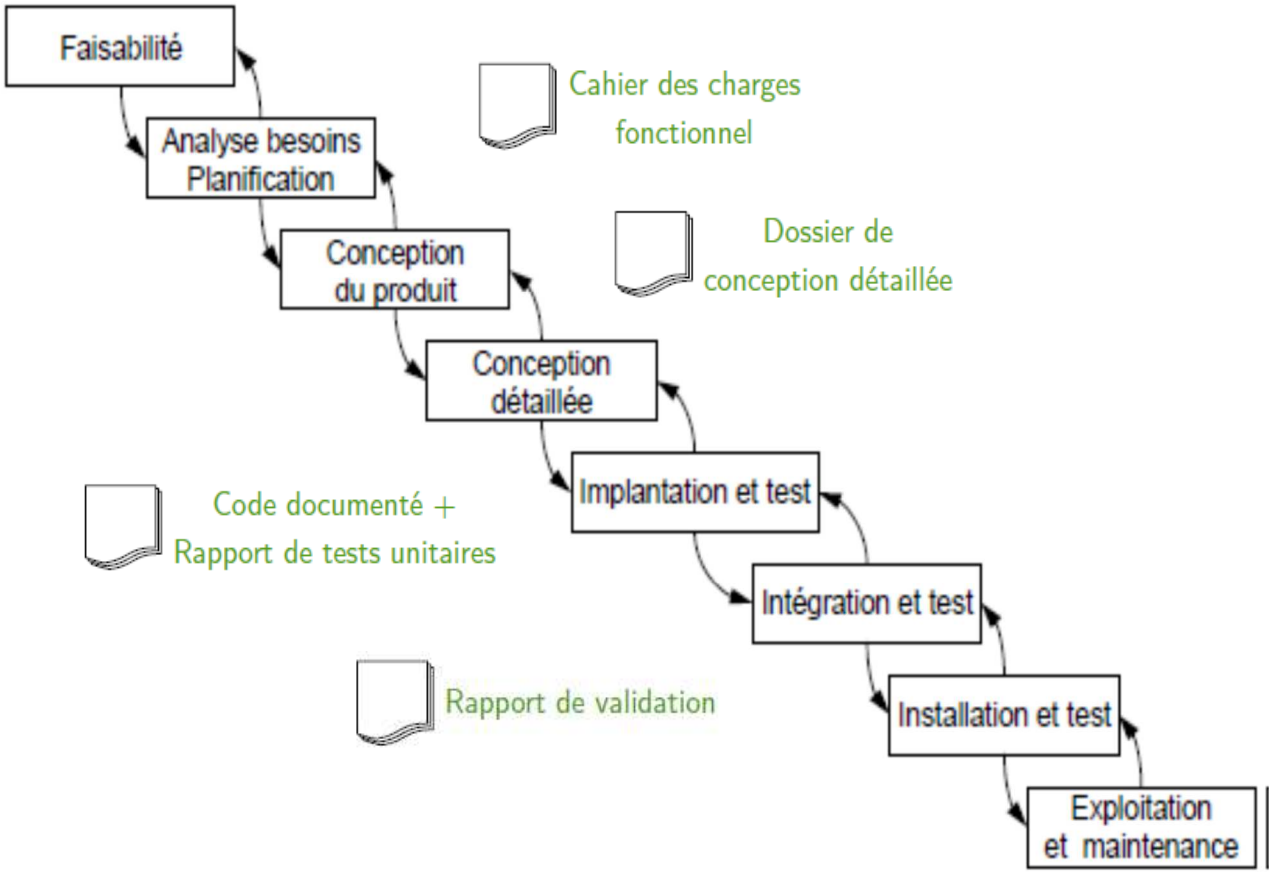
3. Le modèle linéaire de cycle de vie

Historiquement, la première tentative pour mettre de la rigueur dans le 'développement sauvage' (coder et corriger ou 'code and fix') a consisté à distinguer une phase d'*analyse* avant la phase d'*implémentation* (séparation des questions).



3.1. Le modèle en cascade

Le modèle en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970. Il défini des étapes (ou phases) durant lesquelles les activités de développement se déroulent Une étape doit se terminer à une date donnée par la production de certains documents ou logiciels Les résultats de l'étape sont soumis à une étude approfondie. L'étape suivante n'est abordée que si les résultats sont jugés satisfaisants



Inconvénients

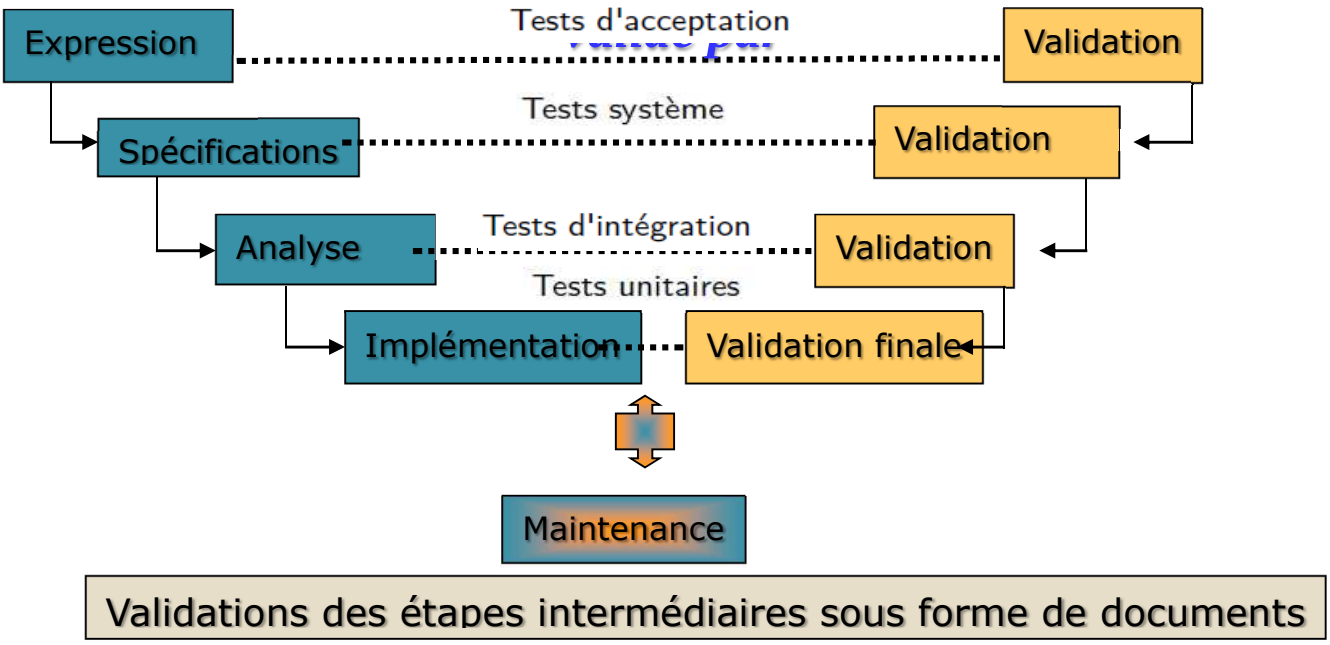
- Entraîne des relations conflictuelles avec les parties prenantes en raison :
  - Du manque de clarté de la définition des exigences
  - D'engagements importants dans un contexte de profonde incertitude
  - D'un désir inévitable de procéder à des changements
- Entraîne une identification tardive de la conception, et un démarrage tardif du codage
- Retarde la résolution des facteurs de risque (intégration tardive dans le cycle de vie)
- Exige d'accorder une attention très importante aux documents

3.2. Le modèle en V

Modèle de cascade amélioré dans lequel le développement des tests et du logiciels 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.

Problème de vérification trop tardive du bon fonctionnement du système.



Objectifs :

Validations intermédiaires pour prévenir les erreurs tardives : meilleure planification et gestion

Principes :

Validation et préparation des protocoles de validation finaux à chaque étape descendante puis validation finale montante et confirmation de la validation descendant

Conséquences :

Obligation de concevoir les jeux de test et leurs résultats, réflexion et retour sur la description en cours et eilleure préparation de la branche droite du V

Intérêts :

Bon suivi du projet avec avancement éclairé et limitation des risques en cascade d'erreurs, une décomposition fonctionnelle de l'activité et génération de documents et outils supports

Limitations :

- Maintenance non intégrée : logiciels à vocation temporaire
- Validations intermédiaires non formelles : aucune garantie sur la non transmission d'erreurs
- Problème de vérification trop tardive du bon fonctionnement du système.
- Un modèle séquentiel (linéaire)

-Manque d'adaptabilité

-Maintenance non intégrée : logiciels à vocation temporaire

-Validations intermédiaires non formelles : aucune garantie sur la non transmission d'erreurs

-Problème de vérification trop tardive du bon fonctionnement du système.

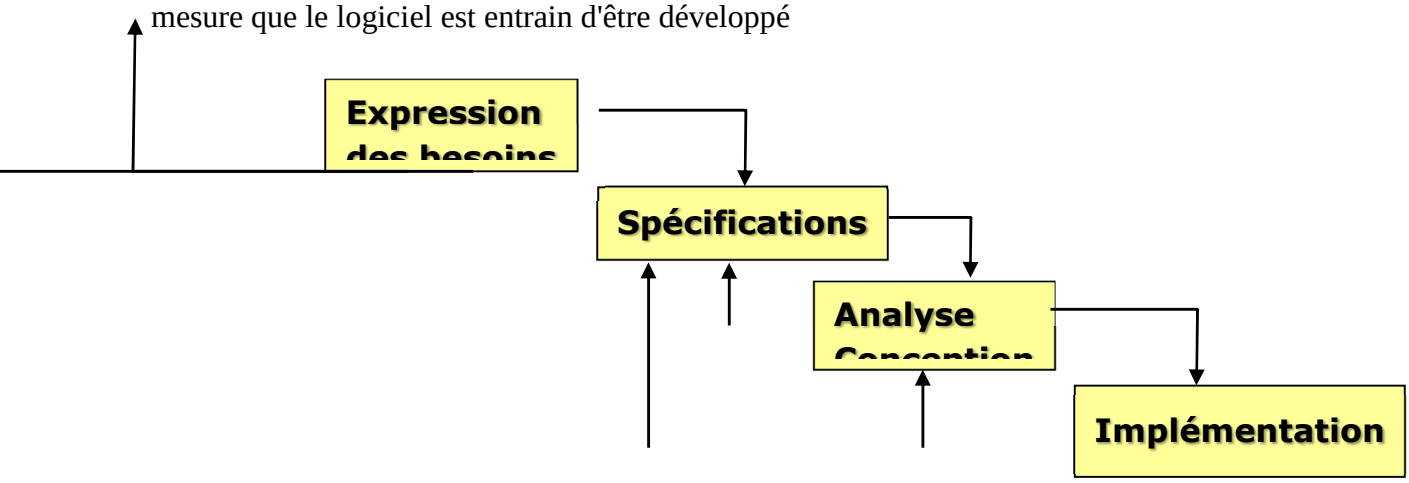
Amélioration :

Retours de correction des phases précédentes pour casser la linéarité : cycle de vie itératif

4. Le modèle itératif

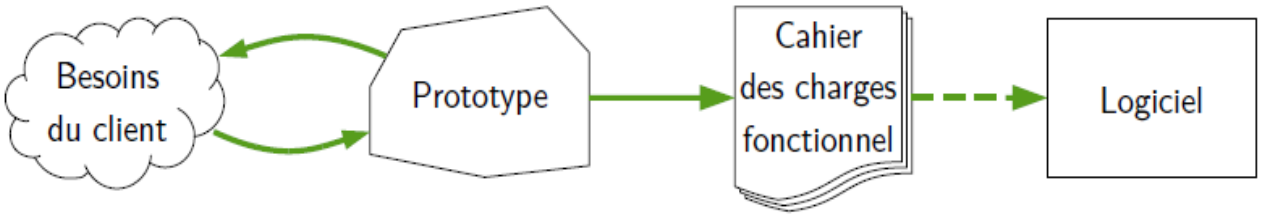
Principe : A chaque étape, on rajoute de nouvelles fonctionnalités pour augmenter le logiciel

Caractéristiques : Chaque étape est relativement simple qui doit être tester et essayer au fur et à mesure que le logiciel est entrain d'être développé



4.1. Le modèle par prototype

Modèle valable dans le cas où les besoins ne sont pas clairement définis ou ils changent au cours de temps. Il permet le développement rapide d'une ébauche du futur logiciel avec le client afin de fournir rapidement un prototype exécutable pour permettre une validation concrète (ici expérimentale) et non sur papier (document) avec moins de risque de spécifications. Des versions successives du prototype (itérations) sont réalisées avec écriture de la spécification à partir du prototype puis processus de développement linéaire

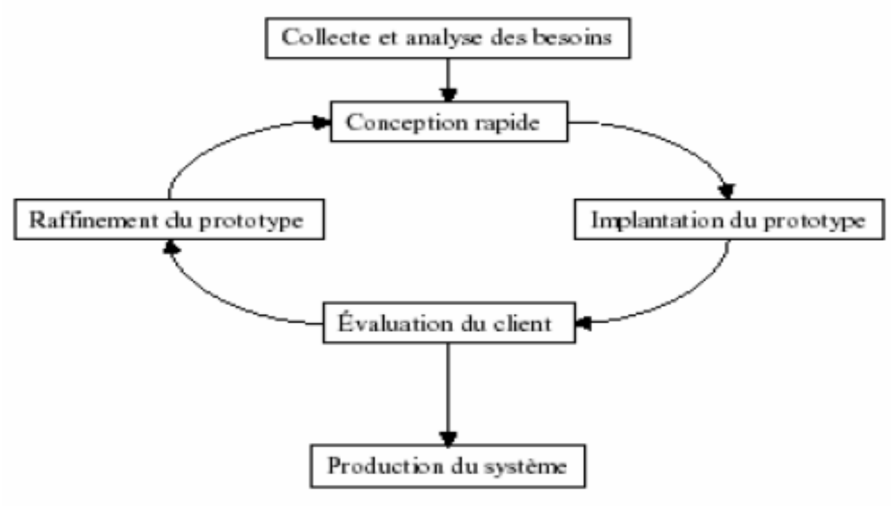


Prototype jetable :

Modèle gérable d'un point de vue changement des spécifications qui sont générées par prototypage qui est construit et utilisé lors de l' analyse des besoins et la spécifications fonctionnelles avec validation des spécifications par l'expérimentation

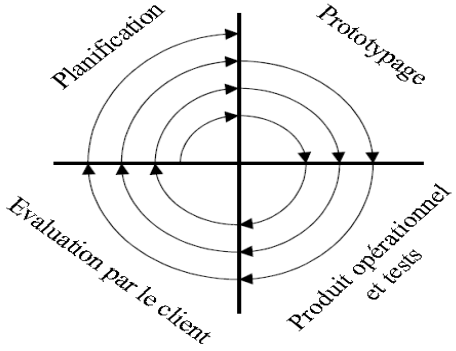
Prototype évolutif :

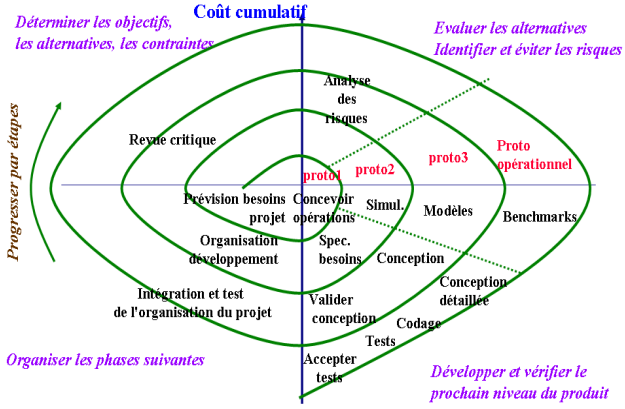
La première version du prototype construit l'embryon du produit final, plusieurs itération sont réalisé jusqu'au produit final cependant il existe des difficulté à mettre en œuvre des procédures de validation et de vérification. Exemple modèle en spirale, développement incrémental



4.2. Le modèle en spirale

Proposé par B. Boëhm en 1988, ce modèle général met l'accent sur l'évaluation des risques.
A chaque étape, après avoir défini les objectifs et les alternatives, celles-ci sont évaluées par différentes techniques (prototypage, simulation, ...). L'étape est réalisée et la suite est planifiée,
Le nombre de cycles est variable selon que le développement est classique ou incrémental





Risques majeurs :

- Défaillance du personnel
- Calendrier et budget irréalistes
- Développement de fonction inapproprié
- Développement d'interfaces utilisateurs inappropriées
- Produit « plaqué or »
- Validité des besoins
- Composants externes manquants
- Tâches externes défaillantes
- Problème de performance

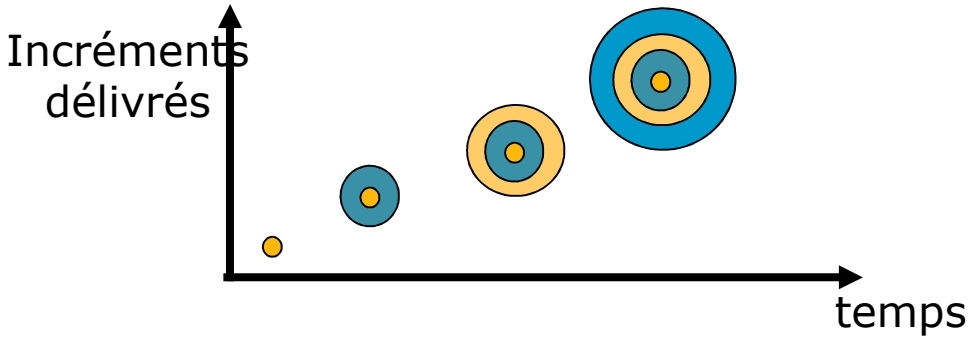
- Exigences démesurées par rapport à la technologie

Risques majeurs :

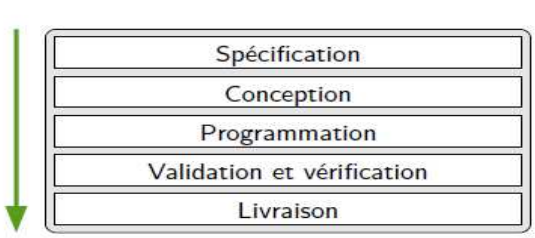
- Défaillance du personnel
- Calendrier et budget irréalistes
- Développement de fonction inapproprié
- Développement d'interfaces utilisateurs inappropriées
- Produit « plaqué or »
- Validité des besoins
- Composants externes manquants
- Tâches externes défaillantes
- Problème de performance
- Exigences démesurées par rapport à la technologie

5. Le modèle incrémental

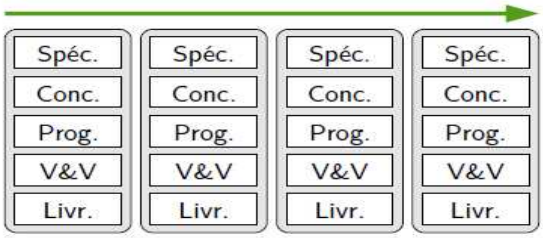
Face aux dérives bureaucratiques de certains gros développements, et à l'impossibilité de procéder de manière aussi linéaire, le *modèle incrémental* a été proposé dans les années 80.



Le développement est réaliser par suite d'incréments ou composantes, qui correspondent à des parties des spécification, et qui viennent intégrer le noyau de logiciel. L'objectif est d'assurer la meilleur adéquation aux besoins possibles. Le choix d'incrément est un compromis entre la durée de développement et le niveau de prise en compte des spécification



Développement en cascade



Développement incrémental

5.1. Avantages du modèle incrémental

- Mise en production plus rapidement du système, avec la diffusion sous forme de version
- Les premiers incréments renforcent l'expression de besoin et autorisent la définition des incrément suivant
- Diminution du risque global d'échec du projet
- Les fonctions les plus utilisées seront les plus testées et donc les plus robustes

5.2. Risques du modèle incrémental

- 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 être spécifiés globalement, au début du projet. Les incréments doivent être aussi indépendants que possibles, fonctionnellement mais aussi sur le plan du calendrier du développement.

6. Extreme Programming (XP)

Nouvelle approche de développement incrémental basée sur la réalisation rapide, en équipe, d'incréments fonctionnels

Caractéristiques :

Ce modèle est en emphase sur la satisfaction du client avec réalisation du produit dont il a besoin avec des livraisons aux moments où il en a besoin et un processus robuste dans le cas où les exigences sont modifiées. Il est en emphase aussi sur le travail d'équipe contenant managers, clients, développeurs qui effectuent un travail d'ensemble

6.1. Valeurs de l'Extreme Programming (XP)

Communication: Au sein de l'équipe de développeurs et avec le client
Feedback : Itérations rapides permettant la réactivité, test du code dès le début
Simplicité : Code simple livrable ne contenant que les exigences du client avec un design propre et simple
Courage : Dans le cas des choix difficiles, peut être facilité par les valeurs précédentes

6.2. Pratique de l'Extreme Programming (Equipe)

Responsabilité collective du code : polyvalence e des développeurs, contribution collective aux nouvelles fonctionnalités, à la correction de bogues et à la restructurations (pas de droits exclusifs)

Programmation en binôme : partage des compétence, prévention des erreurs, motivation mutuelle

Langage commun \ Métaphore : Les mots utilisés pour désigner les entités techniques doivent être choisis parmi les métaphores du domaine de projet

Rythme durable : maintenir un niveau optimal de travail entre énergie nécessaire pour le développement et le repos réparateur

6.3. Pratique de l'Extreme Programming (Code)

Refactoring : Investissement pour le futur, réduction de la dette technique
Conception simple : facilité des reprises du code, facilité de l'ajout de fonctionnalité
Tests unitaires: Test en premier pour une programmation ultérieure plus facile et plus rapide, fixe la spécification (ambiguïtés) avant le codage, feedback immédiat lors du développement, itération du processus pour tout tester
Intégration continue : Ajout de valeurs chaque jour, évite la divergences, efforts fragmentés et permet la diffusion rapide pour la réutilisation
Règles de codage : cohérence globale, code lisible / modifiable par toute l'équipe

6.4. Pratique de l'Extreme Programming (Projet)

Client sur site : orienter le développement en fonction des réactions des usagers et des points critiques du domaine d'utilisation
Tests d'acceptation : conformité par rapport aux attentes du client
Planification itérative et livraison fréquente : Les fiches sont rédigés tout au long du projet, chaque fiche correspond à une fonctionnalité (scénario) où figure la description du scénario, l'ordre de priorité, la durée de réalisation, les noms des développeurs…

6.5. Le cycle standard XP

1. Le client écrit ses besoins sous forme de scénario
2. Les développeurs évaluent le coût de chaque scénario, si le coût est trop élevé pour un scénario ou difficile à estimer, le client le découpe en plusieurs scénario et les soumet aux développeurs
3. Le client choisi, en accord avec les développeur, les scénario à intégrer à la prochaine livraison
4. Chaque binôme des développeur prend la responsabilité d'une tâche
5. Le binôme écrit les tests unitaires correspondant au scénario à implémenter
6. Le binôme procède à l'implémentation du programme
7. Le binôme réorganise le code (refactoring)
8. Le binôme intègre ses développement à la version d'intégration, en s'assurant que tous les tests de non régression passent

7. Le modèle avec réutilisation de composants

Développer un logiciel à l'aide d'une base de composants génériques pré-existants. L'élaboration de la spécification est dirigée par cette base : une fonctionnalité est proposée à l'utilisateur en fonction des composants existants. Ce modèle permet d'obtenir rapidement des produits de bonne qualité (utilisation des composants prouvés). Le travail d'intégration peut s'appuyer sur des outils dirigés par des descriptions de haut niveau du système pour générer le code, cette situation est typique chez les sociétés de services (hébergement de serveurs, déploiement automatique de site Web, . . .).

7.1. Etapes des processus

- Analyse des composants

- Besoins en modification
- Conception avec réutilisation
- Développement et intégration

Cette approche est très **répondu** car elle peut permettre d'important réduction de **coût**

8. Conclusion

Il n'y a *pas de modèle idéal car tout dépend des circonstances*. Le modèle en cascade ou en V est risqué pour les développements innovants car les spécifications et la conception risquent d'être inadéquats et souvent remis en cause. Le modèle incrémental est risqué car il ne donne pas beaucoup de visibilité sur le processus complet. Le modèle en spirale est un canevas plus général qui inclut l'évaluation des risques. Souvent, un même projet peut mêler *différentes approches* , comme le prototypage pour les sous-systèmes à haut risque et la cascade pour les sous systèmes bien connus et à faible risque.

Éléments de qualité logiciel

1. Qualité du logiciel

1.1. Définition Intuitive :

La qualité est la conformité avec les besoins, l'adéquation avec l'usage attendu, le degré d'excellence ou , tout simplement, la valeur de quelque chose pour quelqu'un

1.2. Définition ISO :

Ensemble des traits et des caractéristiques d'un produit logiciel portant sur son aptitude à satisfaire des besoins exprimés ou implicites

1.3. Définition IEEE :

La qualité du logiciel correspond au degré selon lequel un logiciel possède une combinaison d'attributs désirés.

1.4. Définition de Crosby :

La qualité du logiciel correspond au degré selon lequel un client perçoit qu'un logiciel réponde aux multiples attentes.

1.5. Définition de Pressman :

Conformité aux exigences explicites à la fois fonctionnelles et de performances, aux standards de développements explicitement documentés et aux caractéristiques implicites qui sont attendues de tous logiciels professionnellement développés

1.6. Importance de qualité de logiciel

Le logiciel est une composante majeure des systèmes informatiques (environ 80% du coût) utilisés pour : communication (ex. syst. téléphone, syst. Email), santé (monitoring), transport (ex. automobile, aéronautique), échanges économiques (ex. commerce), Entertainment...

Les défauts du logiciel sont extrêmement coûteux en terme d'argent, de réputation et de perte de

vie

1.7. Facteurs de non qualité de logiciel

Mauvaise spécification : Les spécifications des clients qui ne sont pas des informaticiens Vague, incomplète, instables…
Mauvaise estimation : Fausse, oublis, précisions insuffisantes…
Mauvaise répartition des tâches : Organisation inadaptée, contraintes omises ou écart non détecté à temps
Mauvaise réalisation technique : Codage, tests, documentation
Problèmes humains : Mauvaise distribution des travaux, conflits ou rétention d''information
Manque d'expérience du métier de chef de projet : compétence inadaptés

2. Critères de qualité logicielle

ISO/IEC 9126 propose 6 caractéristiques de qualité du produit logiciel. Pour chaque facteur ou critère de qualité, il faut connaître sa définition son importance et comment l'évaluer du point de vue du fournisseur (évaluation en amont) et du point de vue du client (évaluation en aval) et savoir comment l'améliorer ou l'optimiser

2.1. Capacité fonctionnelle *(Functionality)*

Définition : Ensemble d'attributs portant sur l'existence d'un ensemble de fonctions et leurs propriétés. Les fonctions sont celles qui satisfont aux besoins exprimés ou implicites.
Ce critère possède les sous caractéristiques :

- **Aptitude :** présence et adéquation d'une série de fonctions pour des tâches données
- **Exactitude :** fourniture de résultats ou d'effets justes ou convenus
- **Interopérabilité :** capacité à interagir avec des systèmes donnés
- **Sécurité :** aptitude à empêcher tout accès non autorisé (accidentel ou délibéré) aux programmes et données

Ce critère est plus important des critères de qualité on souhaite développer des logiciels qui répondent aux besoin de l'utilisateur tout en faisant un bon usage de ses ressources. Il est capitale pour l'utilisateur final et le client.
La détermination de la capacité
fonctionnelle du système sera fortement influencée par le rédacteur du texte original. En ce qui concerne le client, c'est le lecteur qui déterminera l'importance de la capacité fonctionnelle. Le rédacteur et le lecteur ne sont concernés que par cette caractéristique.

2.2. Fiabilité *(Reliability)*

Définition : Ensemble d'attributs portant sur l'aptitude du logiciel à maintenir son niveau de service dans des conditions précises et pendant une période déterminée.
Ce critère possède les sous caractéristiques :

- **Maturité :** fréquence des défaillances dues à des défauts du logiciel
- **Tolérance aux fautes :** aptitude à maintenir un niveau de service donné en cas de défaut du logiciel ou de violation de son interface
- **Possibilité de récupération :** capacité à rétablir son niveau de service et de restaurer les informations directement affectées en cas de défaillance ; temps et effort nécessaire pour le faire

Ce critère correspond à l'aptitude d'un logiciel à fournir des résultats voulus dans les conditions

normales d'utilisation selon deux point de vue : qualité de la conception et qualité de la réalisation. Un logiciel est fiable doit répondre aux spécifications, ne jamais produire de résultats faux ou être dans un état incohérent, il doit aussi réagir utilement dans une situation inattendue et ne jamais être en défaut qu'en cas extrême

La fiabilité est subjective : elle dépend de l'utilisateur et du contexte d'utilisation. Elle donne une mesure du degré de confiance et des conséquences d'une faute. Le but est de fournir les services attendus par les utilisateurs sachant que certains services sont extrêmement importants (une seule erreur peut être fatale ex: avionique) tout en réduisant la probabilité des pannes sur une durée fixée et pour un contexte donné.

2.3. Facilité d'utilisation (Usability)

Définition : Ensemble d'attributs portant sur l'effort nécessaire pour l'utilisation et l'évaluation individuelle de cette utilisation par un ensemble défini ou implicite d'utilisateurs.
Ce critère possède les sous caractéristiques :

- **Facilité de compréhension :** effort que doit faire l'utilisateur pour reconnaître la logique et sa mise en œuvre
- **Facilité d'apprentissage :** effort que doit faire l'utilisateur pour apprendre son application, il doit comprendre ce que l'on peut faire avec le logiciel, et savoir comment le faire
- **Facilité d'exploitation :** effort que doit faire l'utilisateur pour exploiter et contrôler l'exploitation de son application

Ce critère concerne l'effectivité, l'efficacité et la satisfaction avec lesquelles des utilisateurs spécifiés accomplissent des objectifs bien définis dans un environnement particulier. Avant de réaliser un logiciel il faut analyser le mode opératoire des utilisateurs, ensuite adapter l'ergonomie des logiciels aux utilisateurs. Ce critère comprend la façon dont l'information et les suggestions sont présentées à l'utilisateur final, et aussi la documentation disponible, et l'effort nécessaire pour apprendre à se servir du système.

2.4. Rendement (Efficiency)

Définition : Ensemble d'attributs portant sur le rapport existant entre le niveau de service d'un logiciel et la quantité de ressources utilisées, dans des conditions déterminées.
Ce critère possède les sous caractéristiques :

- **Comportement vis-à-vis du temps :** temps de réponses et de traitement ; débits lors de l'exécution de sa fonction
- **Comportement vis-à-vis des ressources :** quantité de ressources utilisées ; durée de leur utilisation lorsqu'il exécute sa fonction

Les logiciels doivent satisfaire aux contraintes de temps d'exécution en développant des produits plus simples tout en veillant à la complexité des algorithmes et en utilisant des machines plus performantes.

2.5. Maintenabilité (Maintainability)

Définition : Ensemble d'attributs portant sur l'effort nécessaire pour faire des modifications données.
Ce critère possède les sous caractéristiques :

- **Facilité d'analyse :** effort nécessaire pour diagnostiquer les déficiences et causes de défaillance ou pour identifier les parties à modifier
- **Facilité de modification :** effort nécessaire pour modifier, remédier aux défauts ou changer d'environnement

- **Stabilité :** risque des effets inattendus des modifications
- **Facilité de test :** effort nécessaire pour valider le logiciel modifié

Ce critère concerne la gestion du processus de modification pour éviter que des corrections partielles soient faites en dehors du processus d'itérations.
Il existe trois type de maintenance

- **Maintenance perfective (évolutive)** qui consiste à maintenir les fonctionnalités antérieures tout en ajoutant des nouvelles fonctionnalités qui modifient profondément l'architecture comme le changement de SGBD…
- **Maintenance adaptative** qui correspond à l'ajout de petites fonctionnalités qui ne modifie pas l'architecture comme passage de données par fichiers…
- **Maintenance corrective** qui permet la correction des anomalies ou des erreurs mises à jour par le client et non pas lo rs des tests de vérification et de validation.

Le coût de la maintenance est influencé par l'age du logiciel, l'importance des modification, la stabilité de l'équipe, la qualité de la documentation technique ou de document de maintenance

2.6. Portabilité (Portability)

Définition : Ensemble d'attributs portant sur l'aptitude du logiciel à être transféré d'un environnement à l'autre.
Ce critère possède les sous caractéristiques :

- **Facilité d'adaptation :** possibilité d'adaptation à différents environnements donnés sans que l'on ait recours à d'autres actions ou moyens que ceux prévus à cet effet pour le logiciel considéré
- **Facilité d'installation :** effort nécessaire pour installer le logiciel dans un environnement donné
- **Conformité aux règles de portabilité :** conformité aux normes et aux conventions ayant trait à la portabilité
- **Interchangeabilité :** possibilité et effort d'utilisation du logiciel à la place d'un autre logiciel donné dans le même environnement

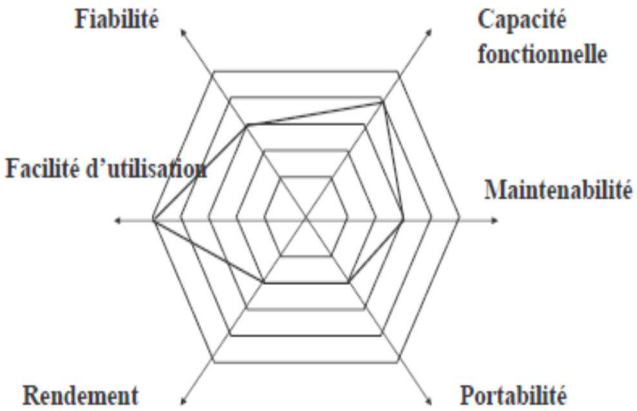
Ce critère est aussi très important car il faut toujours essayer de développer un logiciel portable d'une plate-forme à une autre, d'un site à un autre ; sachant que les architectures machines diffèrent, les systèmes d'exploitation, les performances aussi… de plus il faut prendre en considération qu'un logiciel vit et dure dans le temps et que l'environnement humain et matériel évolue.

2.7. Compromis de qualité

Les critère de qualité peuvent être contradictoires, pour chaque, le choix des compromis doit s'effectuer en fonction du contexte.

- Facilité d'emploi **vs** maintenabilité
- Fiabilité **vs** portabilité
- Capacité **vs** rendement

La qualité s'obtient par la mise en place de procédure et des méthodes de phase de spécification tout en prenant en compte que la démarche qualité ne consiste pas à corriger les défauts mais à les prévenir



3. Assurance qualité logicielle

3.1. Définition

C'est un modèle planifié et systématique de toutes les actions nécessaires pour fournir une confiance adéquate qu'un article ou un produit est conforme à ses exigences techniques établies afin d'obtenir de la qualité requise. Il comprend un ensemble d'activités conçu pour évaluer le processus par lequel les produits sont développés ou fabriqués.

➔Mise en œuvre d'une approche préventive de la qualité : L'AQ consiste en un ensemble d'actions de prévention des défauts qui accompagnent le processus de développement des artefacts logiciels.

Ce modèle passe par l'élaboration d'un MANUEL QUALITE

3.2. Objectifs de l'AQL :

- Assurer un niveau de confiance acceptable que le logiciel sera conforme aux exigences fonctionnelles techniques.
- Assurer un niveau de confiance acceptable que le logiciel sera conforme aux exigences de gestion concernant l'échéancier et le budget.
- Initier et gérer des activités visant à l'amélioration et la plus grande efficience des activités de développement et d'assurance de qualité logicielle.
- Initier et gérer des activités visant à l'amélioration et à l'augmentation de l'efficience des activités de maintenance et d'assurance de qualité logicielle

3.3. Contrôle de la qualité

Ensemble des processus et méthodes pour contrôler le travail et observer si les exigences sont satisfaites tout en réalisant une série de sélection des caractéristiques d'évaluation ainsi que la spécification des techniques d'évaluation utilisée et en effectuant des revues et des lecture et aussi des suppressions des défectuosités avant la livraison du produit.

Le CQ correspond à la **validation** : effectuée à la fin du processus de développement pour assurer

la conformité aux exigences du produit (répondre à la question "sommes nous en train de faire le bon produit?) et à la **vérification** : effectuée à la fin d'une phase pour s'assurer que des besoins établis pendant la phase précédente ont été répondus (répondre à la question "est ce que nous faisons le produit correctement")

3.4. Directives complémentaires

Les directives d'identification, d'implantation et d'analyse des métriques nécessaires au processus d'évaluation du produit final,

Les directives de définition des indicateurs qui permettent des évaluations partielles pendant le cycle de développement

Les buts de ces directives sont :

- Les informations générales sur les indicateurs de qualité des logiciels ;
- Les critères de sélection de ces indicateurs
- Les directions pour l'évaluation des données de mesurage
- Les directions pour l'amélioration du processus de mesurage
- Les exemples de types de graphes d'indicateurs

3.5. Manuel qualité :

C'est un document décrivant les dispositions générales prises par l'entreprise pour obtenir la qualité de ses produits ou de ses services destiné à un usage interne pour évaluer les produits développé par l'entreprise et externe pour évaluer les produits développé par les autres

Organisé en 6 parties:

- Organisation de l'entreprise
- Activités de production et de contrôle technique
- Activité de gestion
- Activité de contrôle de la qualité
- Plan type du PLAN QUALITE
- Lignes directrices permettant d'établir le plan qualit

Utilisation du manuel qualité :

- Communiquer la politique, les procédures et les exigences
- Mettre en œuvre un système effectif;
- Fournir une maîtrise améliorée des pratiques et faciliter les activités relatives à l'assurance;
- Fournir les bases documentaires pour auditer les systèmes qualité;
- Assurer la continuité du système qualité et de ses exigences en cas de modification des circonstances;
- Former le personnel aux exigences de système qualité et aux méthodes relatives à la conformité
- Présenter leur système qualité pour des usages externes, tel que la démonstration de leur conformité aux normes
- Démontrer la conformité de leurs systèmes qualité avec les normes relatives à la qualité

dans des situations contractuelles.

3.6. Plan qualité logiciel :

C'est un document décrivant les dispositions spécifiques prises par une entreprise pour obtenir la qualité du produit ou du service considéré. Il comprend les techniques et les activités à caractère opérationnel qui ont pour but à la fois de piloter un processus et d'éliminer les causes de fonctionnement non satisfaisant à toutes les phases de la boucle de la qualité en vue d'atteindre la meilleure efficacité économique

Il contient : le produit auquel il est prévu d'être appliqué, les objectifs relatifs à la qualité du produit (exprimer ces objectifs en termes mesurables chaque fois que cela est possible), les exclusions spécifiques ainsi que la période de validité

Plan type de qualité logiciel :

1)But, Domaine d'application et responsabilité: Portée du plan qualité et dispositions pour en assurer son application

2) Documents applicables et documents de références: Documents appelés dans le plan qualité

3) Terminologie : glossaire et termes utilisés

4) Organisation: Personnes intervenant dans le projet, pour chaque personne il faut connaître sa place dans la structure de l'entreprise, son rôle et ses responsabilités dans le projet ainsi que les liens hiérarchiques et fonctionnels entre les intervenants

5)Démarche de développement : contenant la liste des phases de développement et pour chaque phase définir le contenu de ses activités, les documents ou produits en entrée ou réalisés ainsi que les conditions de passage à la phase suivante et points clés

6)Documentation

Liste des documents produits dans chaque phase avec des références aux plans types de chaque document, son statut (livrable, consultable, privée...) les documents classés en: documents de gestion de projet, en documents techniques de réalisation ou en manuels d'utilisation et d'exploitation

7) Gestion des configurations : Éléments de configuration y compris les moyens de développement et de tests avec des conventions d'identification (nomenclatures)

8) Gestion des modifications : citant le responsable de leur mise en œuvre ainsi que les règles d'évolution de l'identification des éléments modifiés et de la nomenclature

9) Méthodes, outils et règles

10) Contrôle des fournisseurs

11) Reproduction, protection, livraison

12) Suivi de l'application du plan qualité (plan de contrôle): Comprenant les interventions du responsable qualité sur la démarche de développement, les interventions du responsable qualité dans les procédures de gestion des configurations, de gestion des modifications, la vérification des exigences de qualité envers les fournisseurs ainsi que les modalités de recette et qualification

3.7. Principes généraux de l'AQL

La définition des exigences de qualité : interpréter le contexte afin d'avoir des exigences et

spécifications explicites et aussi un processus de développement logiciel avec l'analyse des exigences, les tests d'acceptabilité, et des feedbacks fréquents des usagers

La préparation des processus d'évaluation : Planifier les activités de contrôle qualité initier et mettre au point les bases de l'évaluation . Ceci est fait en trois sous-étapes :

- **La sélection des métriques de qualité :** Ces dernières doivent correspondre aux caractéristiques énumérées plus haut.
- **La définition des taux de satisfaction :** Les échelles de valeurs doivent être divisées en portions correspondant aux niveaux de satisfaction des exigences.
- **La définition des critères d'appréciation :** Ceci inclut la préparation de la procédure de compilation des résultats par caractéristique. Il est possible aussi de prendre en compte dans cette procédure des aspects de gestion tels que le temps ou les coûts.

L'exécution de la procédure d'évaluation : réaliser l'évaluation du logiciel avant et après l'exécution ce qui permet de donner un cadre rigoureux pour guider le développement du logiciel, de sa conception à sa livraison, de contrôler les coûts d'évaluer les risques et respecter les délais.
Il existe quatre méthodes complémentaire pour faire cette évaluation : les méthodes formelle, les métriques, les revues/inspections et les tests

3.8. Méthodes d'évaluation de qualité logicielle

Méthodes formelles : Elles consistent à vérifier mathématiquement des propriétés spécifiées pour prouver la correction (expression formelle des spécifications), la terminaison de l'exécution (production de la preuve d'une conception et d'une implémentation sans erreurs) et des propriétés spécifiques (explicitation précise des propriétés d'architecture de développement). Pour atteindre cet objectif il faut utiliser des assistants de preuve (theorem prover) comme les systèmes Coq, PVS, Isabelle / HOL, ...Cependant ces méthodes sont encore difficiles d'usage pour des non spécialistes

Métriques : Elle consistent à mesurer un ensemble connu de propriétés liées à la qualité
Mesures des processus (les séries d'activités reliées au développement du logiciel), des produits (les objets produits, livrables ou documents qui résultent d'une activité d'un processus), des ressources (les entités exigées par une activité d'un processus)

- **Mesure des caractéristiques « internes » :** mesures objectives de taille, de complexité du flot de contrôle, de cohésion modulaire / couplage entre modules, ...
- **Mesure des caractéristiques « externes » :** évaluations stochastiques (statistiques) de délai moyen de réponse à une requête, de nombre de requêtes simultanées sans « écrouler » un serveur, ...

Ce sont des mesures **postérieurs** qui peuvent arriver parfois « trop tard »

Revues et Inspections : Relecture et examen par humain des exigences, design, code, ... basés sur des checklists. Cela consiste à effectuer soit un examen détaillé d'une spécification, d'une conception ou d'une implémentation par une personne ou un groupe de personnes, afin de déceler des fautes, des violations de normes de développement ou d'autres problèmes ; soit un examen systématique de certains points pour la recherche de potentiels défauts et la vérification de l'application de certaines règles menée par un spécialiste du domaine inspecté. Il existe plusieurs types de revues ou d'inspections :
Auto-correction (desk-checking) : c'est une relecture personnelle avec une efficacité quasi nulle pour les documents amont, faible pour le code

Lecture croisée (author-reader cycle) : c'est un collègue recherche des ambiguïtés, oublis,

imprécisions. L'efficacité de cette méthode est faible pour les documents amont mais plus adapté pour la relecture du code

Revue (walkthrough) : c'est un lecteur qui résume paragraphe par paragraphe en animant une discussion informelle au sein d'un groupe. Ceci a une contribution moyenne à l'assurance qualité (évaluation très liée à la prestation du lecteur)

Revue structurée : c'est la constitution d'une liste de défauts, pendant un débat dérigé par un secrétaire, en utilisant une liste de défauts typiques (checklist) cela a une bonne contribution à l'assurance qualité

Inspection : c'est un cadre de relecture plus formel avec recherche des défauts avant les débats et un bon suivi des décisions et corrections cela a une excellente contribution à l'assurance qualité

3.9. Défauts typiques

Référence aux données :

- Variables non initialisées
- Indices de tableaux hors bornes
- Accès à des structures/records à champs variables ou à des unions
- Confusion entre donnée et pointeur vers la donnée
- Déréférence de pointeurs nuls
- Pointeurs sur des données désallouées ou pas encore allouées
- Pointeurs sur des données devenues inutiles mais non libérées

Calculs :

- Conversions de type (implicites et explicites)
- Underflow/overflow (dépassement de capacité du type)
- Division par zéro
- Précédence des opérateurs

Comparaison :

- incohérence des types : mélanges d'entiers et de booléens
- inclusion ou non des bornes incorrecte :
  < au lieu de <=, >= au lieu de >, …
- inversion du test : == au lieu de !=, > au lieu de <, ...
- confusion en égalité (==) et affectation (=) :
  if (x = y) {...} au lieu de if (x == y) {...}:
- confusion entre opérateurs binaires (bits) et logiques :
  et (&, &&, and), ou (|, ||, or), négation (~, !, not)
- négation incorrecte d'une condition logique
  !(x ==1 && (y < 2 || !z)) équiv. x !=1 || (y >= 2 && z)
- précédence des opérateurs booléens
  x || y && z équiv. x || (y && z)

Contrôle :

- Switch : ensemble de case incomplet, cas default manquant, break oublié
- Rattachement du else au if
- Terminaison du programme : boucles et récusions sans fin
- Boucles : conditions initiales (indices, ...) incorrectes, itérations en plus ou en moins, incohérences après une sortie de boucle anticipée, incohérences après une sortie de boucles emboîtées
- Procédures et fonctions : incohérences après une sortie anticipée
- Exceptions non rattrapées

Tests : Données explicites pour exécuter le logiciel et vérifier si les résultats correspondent aux

attentes. Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus, ils existent deux catégories de test :

Test de vérification (conformité aux spécifications) qui consiste aux tests unitaires de chaque module et les tests d'intégration des modules les un aux autres

Test de qualification soit de la validation par rapport aux contraintes non fonctionnelles avec tests de performance ou des tests de capacité de charge soit de la validation par rapport aux besoins avec les bêta-tests réalisés chez l'utilisateur fina

4. Méthode SCOPE (Software CertificatiOn Programmein Europe)

C'est un projet européen ESPRIT qui a comme objectifs :

Définir les procédures d'attribution d'un label de qualité à un logiciel quand celui-ci satisfait un certain ensemble d'attributs de qualité

Développer des technologies nouvelles et efficaces d'évaluation, à des coûts raisonnables, permettant l'attribution de ce label

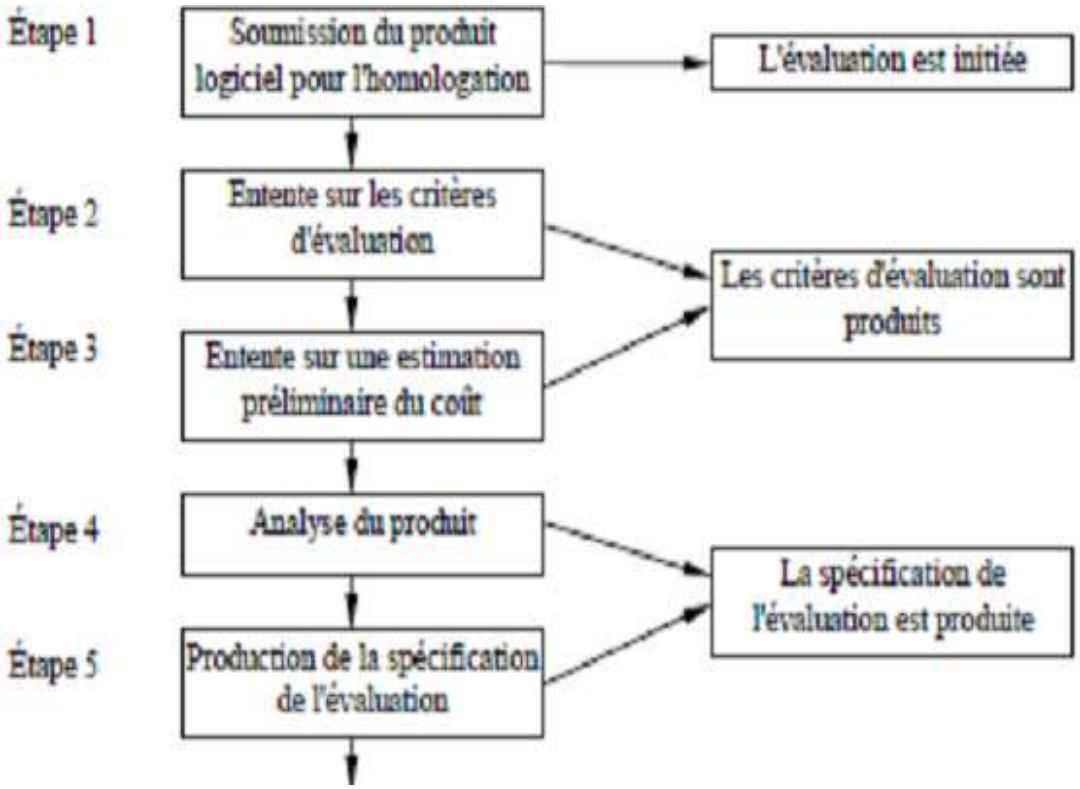
Promouvoir l'utilisation des technologies modernes de l'ingénierie des logiciels. Celles-ci, étant utilisées durant le développement des logiciels, contribuent à l'attribution de ce label

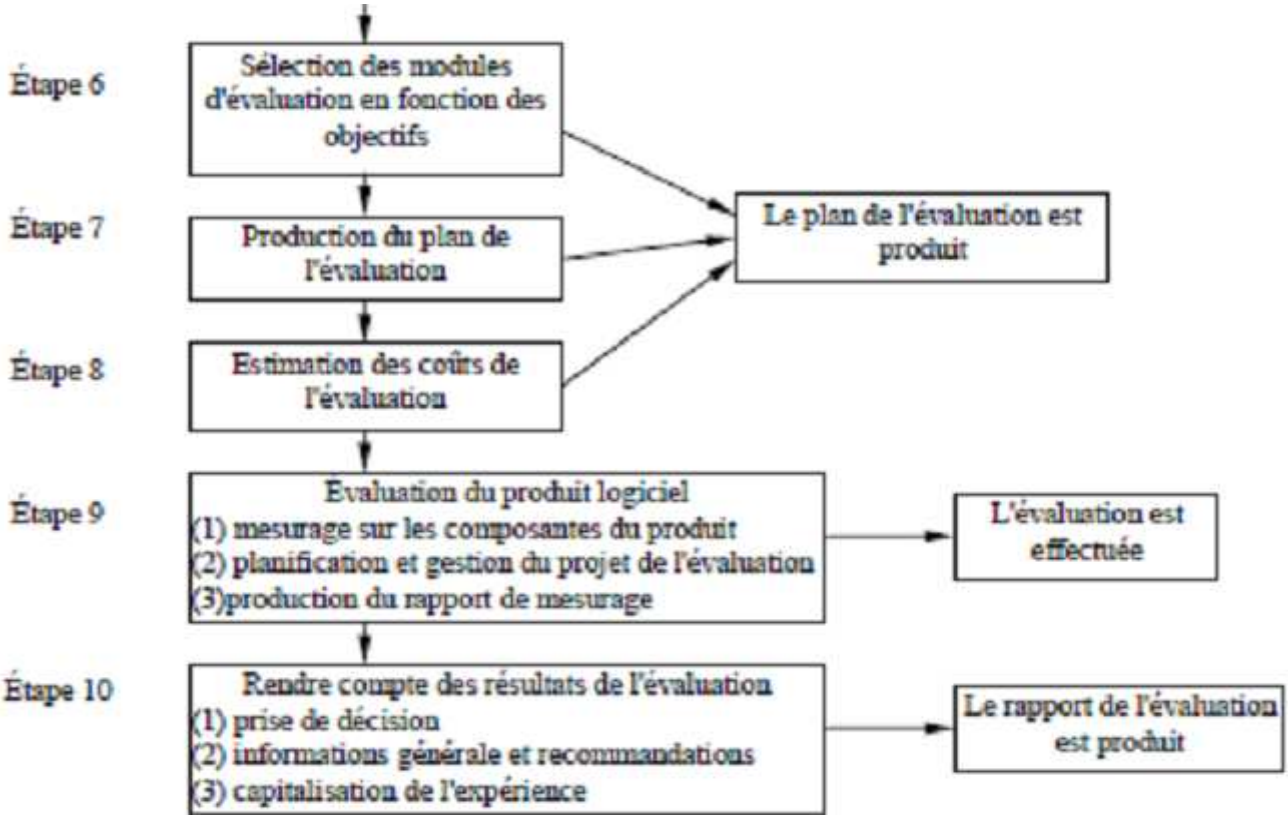
La méthode d'évaluation s'appuie sur trois types d'analyse techniques

L'analyse statique qui consiste à examiner le code pour **évaluer** les caractéristiques de qualité.

L'analyse dynamique qui consiste entre autres à **simuler** le déroulement de l'application pour effectuer des mesures.

L'inspection qui concerne particulièrement les **interfaces "utilisateur".**




Étape 6
Sélection des modules d'évaluation en fonction des objectifs
Étape 7
Production du plan de l'évaluation
Étape 8
Estimation des coûts de l'évaluation
Le plan de l'évaluation est produit
Étape 9
Evaluation du produit logiciel
(1) mesurage sur les composantes du produit
(2) planification et gestion du projet de l'évaluation
(3)production du rapport de mesurage
L'évaluation est effectuée
Étape 10
Rendre compte des résultats de l'évaluation
(1) prise de décision
(2) informations générale et recommandations
(3) capitalisation de l'expérience
Le rapport de l'évaluation est produit