COURS DE GENIE LOGICIEL 2 eme Partie

Raphael Grevisse Yende

SUPPORT DE COURS DE GENIE LOGICIEL 2

YENDE RAPHAEL Grevisse, Ph.D.

Docteur en Télécoms et Réseaux Inf.

Cours dispensé à l'Institut Supérieur de Commerce en Deuxième Licence CSI.

AVERTISSEMENTS

Le support de cours de *« Génie logiciel »*, demande avant tout, un certain entendement de l'informatique et des connaissances de base des réseaux informatiques et principalement une prédisposition d'analyse inéluctable et cartésienne. Vu que l'apport de ce cours, met l'accent sur le principe de conception de base des systèmes informatiques reposant sur une compréhension technique approfondie de la reproduction des applications informatiques et à l'application de l'ingénieure au développement des logiciels. Le cours de génie logiciel se veut pour objectif primordial d'initier les étudiants de première Licence en Gestion Informatique, à la conception des applications informatiques de façon *méthodique* et *rééditable*; en les incitant à rechercher et établir les fonctionnalités d'une application, et à les modéliser sous forme de cas d'utilisation et de scénarios ainsi que rechercher les classes et les acteurs nécessaires à la conception de l'application.

TABLE DES MATIERES

AVERTISSEMENTS 2
TABLE DES MATIERES 3
BIBLIOGRAPHIE 6
INTRODUCTION 7
OBJECTS DU COURS 8
DEFINITION DES CONCEPTS CLES 9
CHAPITRE I. GESTION DE PROJET LOGICIEL 10
I.1. INTRODUCTION 10
I.2. PROJET LOGICIEL 10
I.2.1. BESOIN DE GESTION DE PROJET LOGICIEL 10
I.2.2. LE CHEF DU PROJET LOGICIEL 11
I.2.3. LES ACTIVITES DE GESTION DE LOGICIELS 12
I.2.3.1. PLANIFICATION DE PROJET 12
I.2.3.2. GESTION DU CADRE DE PROJET 12
I.2.3.3.. ESTIMATION DE PROJET 13
I.2.4. TECHNIQUES D'ESTIMATION DE PROJET 14
I.3. PREVISION D'UN PROJET LOGICIEL 15
I.3.1. LA GESTION DES RESSOURCES 15
I.3.2. GESTION DU RISQUE DE PROJET 16
I.3.3. GESTION DE L'EXECUTION ET SURVEILLANCE DU PROJET 16
I.3.4. GESTION DE LA COMMUNICATION DE PROJET 17
I.3.5. GESTION DE LA CONFIGURATION 18
I.4. OUTILS DE GESTION DE PROJET 19
CHAPITRE II. LES EXIGENCES LOGICIELLES 22
II.1. LES PROCESSUS DES EXIGENCES LOGICIELLES 22
II.2. LES PROCESSUS DE DEMANDE DES EXIGENCES 24
II.3. CARACTERISTIQUES DES EXIGENCES LOGICIELLES 26
II.3.1. LES CARACTERISTIQUES DES EXIGENCES FONCTIONNELLES 26
II.3.2. LES CARACTERISTIQUES DES EXIGENCES NON-FONCTIONNELLES 27
II.4. LES EXIGENCES REQUISES DE L'INTERFACE UTILISATEUR 28
II.5. ANALYSTE DE LOGICIELS 29
II.6. METRIQUES ET MESURES LOGICIELLES 29

CHAPITRE III. LES BASES DE CONCEPTION DE LOGICIELS 31
III.1. NIVEAUX DE CONCEPTION DE LOGICIELS 31
III.2. LA MODULARISATION 32
III.3. CONCURRENCE 32
III.4. COUPLAGE ET COHESION 33
III.4.1. COHESION 33
III.4.1. COUPLAGE 34
CHAPITRE IV. OUTILS D'ANALYSE ET DE CONCEPTION DE LOGICIELS 35
IV.1. DIAGRAMME DE FLUX DE DONNEES 35
IV.1.1. TYPES DE DFD 35
IV.1.2. COMPOSANTS DE DFD 36
IV.1.3. NIVEAUX DE DFD 36
IV.2. LES ORGANIGRAMMES 37
IV.3. DIAGRAMME HIPO 39
IV.4. ANGLAIS STRUCTURE 40
IV.5. PSEUDO-CODE 41
IV.6. MODELE D'ENTITE-RELATION 42
IV.7. DICTIONNAIRE DE DONNEES 43
IV.7.1. EXIGENCE DU DICTIONNAIRE DE DONNEES 43
IV.7.2. CONTENU 43
IV.7.3. ÉLEMENTS DE DONNEES 44
IV.7.4. MAGASIN DE DONNEES 44
IV.7.5. TRAITEMENT DE L'INFORMATION 44
CHAPITRE V. STRATEGIES DE CONCEPTION DE LOGICIELS 45
V.1. CONCEPTION STRUCTUREE 45
V.2. CONCEPTION ORIENTEE FONCTION 45
V.3. CONCEPTION ORIENTEE OBJET 46
V.4. APPROCHES DE CONCEPTION LOGICIELLE 47
CHAPITRE VI. CONCEPTION DE L'INTERFACE UTILISATEUR DU LOGICIEL 49
VI.1. TYPOLOGIE DE L'INTERFACE D'UTILISATEUR 49
VI.1.1. INTERFACE EN LIGNE DE COMMANDE (CLI) 49
VI.1.2. INTERFACE UTILISATEUR GRAPHIQUE 50
VI.2. ACTIVITES DE CONCEPTION D'INTERFACE UTILISATEUR 52
VI.3. OUTILS D'IMPLEMENTATION DE L'INTERFACE GRAPHIQUE 53

VI.4. REGLES D'OR DE L'INTERFACE UTILISATEUR 54
CHAPITRE VII. COMPLEXITE DU DESIGN LOGICIEL 56
VII.1. MESURES DE COMPLEXITE DE HALSTEAD 56
VII.2. MESURE DE COMPLEXITE CYCLOMATIQUE 56
VII.3. MESURE DE POINT DE FONCTION 57
CHAPITRE VIII. MISE EN ŒUVRE LOGICIELLE 60
VIII.1. PROGRAMMATION STRUCTUREE 60
VIII.2. PROGRAMMATION FONCTIONNELLE 61
VIII.3. STYLE DE PROGRAMMATION 62
VIII.4. DOCUMENTATION DU LOGICIEL 63
VIII.4.1. DOCUMENTATION DES BESOINS 63
VIII.4.2. DOCUMENTATION DE CONCEPTION DE LOGICIEL 63
VIII.4.3. DOCUMENTATION TECHNIQUE 64
VIII.4.4. DOCUMENTATION UTILISATEUR 69
LES FORMES DE LA DOCUMENTATION UTILISATEUR 69
VIII.5. DEFIS D'IMPLEMENTATION LOGICIELLE 71
CHAPITRE IX. VUE D'ENSEMBLE DES TESTS LOGICIELS 72
IX.1. LES COMPOSANTS DU TEST LOGICIEL 72
IX.1.1. LA VALIDATION DU LOGICIEL 72
IX.1.2. LA VERIFICATION DU LOGICIEL 73
IX.2. TEST MANUEL OU AUTOMATISE 73
IX.3. APPROCHES DES TESTS LOGICIELS 74
IX.3.1. TEST DE LA BOITE NOIRE 74
IX.3.2. TEST DE LA BOITE BLANCHE 75
IX.4. NIVEAUX DES TESTS LOGICIELS 75
IX.5. DOCUMENTATION DE TEST LOGICIEL 77
IX.6. TESTS ET CONTROLE DE LA QUALITE, ASSURANCE ET AUDIT 78
CHAPITRE X. PRESENTATION DES OUTILS DE GESTION DE LOGICIELS 79
X.1. LES OUTILS « CASE » 79
X.2. COMPOSANTS DES OUTILS CASE 79
X.3. TYPES DES OUTILS DE « CASE » 80
CONCLUSION 83

BIBLIOGRAPHIE

- **ACSIOME,** « *Modélisation dans la conception des systèmes d'information* », Masson, 1989
- **Bertrand Meyer**, « *Conception et programmation orientées objet* » Editions Eyrolles, 2000
- **Boehm, B. W.**« *Software engineering economics* », Prentice-Hall, 1981,
- **Booch, Grady,** "*Object-Oriented Analysis and Design, with applications*", 3rd Ed. Addison- Wesley, 2007
- **C. TESSIER,** « *La pratique des méthodes en informatique de gestion,* » Les Editions d'Organisation, 1995
- **GALACSI,** « *Comprendre les systèmes d'information : exercices corrigés d'analyse et de conception,* » Dunod, 1985
- **I. Somerville et Franck Vallée,** « *Software Engineering* » 6th Edition, Addison-Wesley, 2001
- **I. SOMMERVILLE**, « *Le génie logiciel et ses applications,* » Inter-Éditions, 1985
- **Marie-Claude Gaudel**, « Précis de génie logiciel », Editions Dunod, 2012
- **P. ANDRÉ et A. VAILLY,** « *Conception des systèmes d'information : Panorama des méthodes et techniques* », Ellipses, collection TECHNOSUP / Génie Logiciel, 2001
- **P. ANDRÉ et A. VAILLY,** « *Spécification des logiciels – Deux exemples de pratiques récentes : Z et UML* », Ellipses, collection TECHNOSUP / Génie Logiciel, 2001.

INTRODUCTION

Le terme de « *Génie logiciel* » a été introduit à la fin des années soixante lors d'une conférence tenue pour discuter de ce que l'on appelait alors (*la crise du logiciel « software crisis »*) … Le développement de logiciel était en crise. Les coûts du matériel chutaient alors que ceux du logiciel grimpaient en flèche. Il fallait de nouvelles techniques et de nouvelles méthodes pour contrôler la complexité inhérente aux grands systèmes logiciels. La crise du logiciel était perçue à travers ces symptômes :

- *La qualité du logiciel livré était souvent déficiente* : Le produit ne satisfaisait pas les besoins de l'utilisateur, il consommait plus de ressources que prévu et il était à l'origine de pannes.
- Les performances étaient très souvent médiocres (*temps de réponse trop lents*).
- Le non-respect des délais prévus pour le développement de logiciels ne satisfaisant pas leurs cahiers des charges.
- Les coûts de développement d'un logiciel étaient presque impossible à prévoir et étaient généralement prohibitifs (*excessifs*)
- L'invisibilité du logiciel, ce qui veut dire qu'on s'apercevait souvent que le logiciel développé ne correspondait pas à la demande (*on ne pouvait l'observer qu'en l'utilisant « trop tard »).*
- La maintenance du logiciel était difficile, coûteuse et souvent à l'origine de nouvelles erreurs.

Mais en pratique, il est indispensable d'adapter les logiciels car leurs environnements d'utilisation changent et les besoins des utilisateurs évoluent. Il est rare qu'on puisse réutiliser un logiciel existant ou un de ses composants pour confectionner un nouveau système, même si celui-ci comporte des fonctions similaires. Tous ces problèmes ont alors mené à l'émergence d'une discipline appelée « *le génie logiciel* ». Ainsi, Les outils de génie logiciel et les environnements de programmation pouvaient aider à faire face à ces problèmes à condition qu'ils soient eux-mêmes utilisés dans un cadre méthodologique bien défini.

Nul n'ignore que les plans logiciels sont connus pour *leurs pléthores de budget*, et *leurs cahiers des charges*[1] *non respectées*. De façon générale, le développement de logiciel est une activité complexe, qui est loin de se réduire à la programmation. Le développement de logiciels, en particulier lorsque les programmes ont une certaine taille, nécessite d'adopter une méthode de développement, qui permet d'assister une ou plusieurs étapes du cycle de vie de logiciel.

[1] *Le cahier des charges est un document recensant les spécifications/exigences d'un produit*

Parmi les méthodes de développement, les approches objet, issues de la programmation objet, sont basées sur une modélisation du domaine d'application[2]. Cette modélisation a pour but de faciliter la communication avec les utilisateurs, de réduire la complexité en proposant des vues à différents niveaux d'abstraction, de guider la construction du logiciel et de documenter les différents choix de conception.

Le génie logiciel (*software engineering*) représente l'application de principes d'ingénierie au domaine de la création de logiciels. Il consiste à identifier et à utiliser des méthodes, des pratiques et des outils permettant de maximiser les chances de réussite d'un projet logiciel. Il s'agit d'une science récente dont l'origine remonte aux années 1970. A cette époque, l'augmentation de la puissance matérielle a permis de réaliser des logiciels plus complexes mais souffrant de nouveaux défauts : délais non respectés, coûts de production et d'entretien élevés, manque de fiabilité et de performances. Cette tendance se poursuit encore aujourd'hui. L'apparition du génie logiciel est une réponse aux défis posés par la complexification des logiciels et de l'activité qui vise à les produire.

OBJECTS DU COURS

Ce cours a pour objectif principal, d'initier les étudiants, à la conception des applications informatiques de façon systématique (*méthodique*) et reproductible (*rééditable*); en les incitant à rechercher et établir les fonctionnalités d'une application, et à les modéliser sous forme de cas d'utilisation et de scénarios ainsi que rechercher les classes et les acteurs nécessaires à la conception de l'application. Et, D'une façon spécifique ce cours vise à :

- Procurer à l'étudiants de la deuxième licence, les bonnes pratiques de conception, comme l'utilisation de patron de conception (*design pattern*), le découpage en couches de l'architecture, la structuration en paquetages et le maquettage ;
- Maîtriser des techniques de génie logiciel, en se focalisant sur les approches par objets et par composants ;
- Exposer les principaux courants de pensées en matière de développement logiciel.
- Proposer un ensemble de pratiques pragmatiques qui permettent de survivre à un projet de développement de logiciel.

[2] *C'est par exemple, le langage UML (Unified Modeling Language) qui est un langage de modélisation, qui permet d'élaborer des modèles objets indépendamment du langage de programmation, à l'aide de différents diagrammes.*

DEFINITION DES CONCEPTS CLES

- ***Le génie logiciel*** est un domaine des sciences de l'ingénieur dont l'objet d'étude est la *conception*, *la fabrication,* et la *maintenance* des *systèmes informatiques complexes*.

- ***Un système*** est un ensemble d'*éléments* interagissant entre eux suivant un certains nombres de principes et de *règles* dans le but de réaliser un *objectif*.

- ***Un logiciel***[3] est un ensemble d'entités nécessaires au fonctionnement d'un processus de traitement automatique de l'information. Parmi ces entités, on trouve par exemple : des programmes (en format *code source* ou exécutables);des documentations d'utilisation ;des informations de configuration.

- **Un modèle** : est une représentation schématique de la réalité.

- **Une base de Données**: ensemble des données (*de l'organisation*) structurées et liées entre elles : stocké sur support à accès direct (*disque magnétique*) ; géré par un SGBD (*Système de Gestion de Bases de Données*), et accessible par un ensemble d'applications.

- **Une analyse** : c'est un processus d'examen de l'existant

- Une **Conception** : est un processus de définition de la future application informatique.

- **Un système d'Information** : ensemble des moyens (*humains et matériels*) et des méthodes se rapportant au traitement de l'information d'une organisation.

YENDE RAPHAEL Grevisse, PhD.
Professeur associé

[3] Un logiciel est en général un sous-système d'un système englobant. Il peut interagir avec des clients, qui peuvent être : des opérateurs humains (*utilisateurs, administrateurs, etc.)* ; d'autres logiciels ; des contrôleurs matériels. Il réalise une spécification c'est-à-dire son comportement vérifie un ensemble de critères qui régissent ses interactions avec son environnement.

CHAPITRE I. GESTION DE PROJET LOGICIEL

I.1. INTRODUCTION

La structure de l'emploi d'une entreprise informatique engagée dans le développement de logiciels peut être divisée en deux parties :

- Création de logiciel
- Gestion de projet logiciel

Un projet est une tâche bien définie, c'est-à-dire une collection de plusieurs opérations effectuées pour atteindre un objectif (par exemple, le développement et la livraison de logiciels). Un projet peut être caractérisé de la manière suivante :

- Chaque projet peut avoir un objectif unique et distinct ;
- Le projet n'est pas une activité de routine ou des opérations quotidiennes ;
- Le projet est livré avec une heure de début et une heure de fin ;
- Le projet se termine lorsque son objectif est atteint, il s'agit donc d'une phase temporaire dans la vie d'une organisation ;
- Le projet a besoin de ressources adéquates en termes de temps, de personnel, de financement, de matériel et de banque de connaissances.

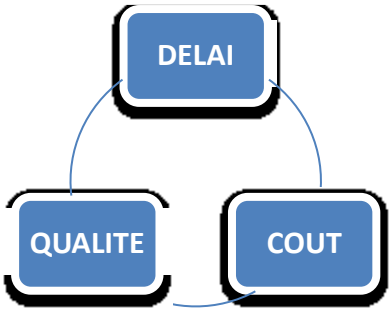
I.2. PROJET LOGICIEL

Un projet logiciel est la procédure complète de développement de logiciel allant de la collecte des exigences aux tests et à la maintenance, effectuée selon les méthodes d'exécution, dans un délai spécifié pour obtenir le produit logiciel voulu.

I.2.1. BESOIN DE GESTION DE PROJET LOGICIEL

Le logiciel est considéré comme un produit immatériel. Le développement de logiciels est une sorte de nouveau courant dans le monde des affaires et il existe très peu d'expérience dans la création de produits logiciels. La plupart des logiciels sont adaptés aux besoins du client. Le plus important est que la technologie sous-jacente change et avance si souvent et rapidement que l'expérience d'un produit ne peut pas être appliquée à l'autre.

Toutes ces contraintes commerciales et environnementales entraînent des risques dans le développement de logiciels. Il est donc essentiel de gérer efficacement les projets logiciels.



L'image ci-dessous montre les 3 grandes contraintes pour les projets logiciels. En effet, c'est une partie essentielle de l'organisation logicielle quant à la livraison d'un produit de qualité en gardant le coût constant dans le budget du client et dans un délai reparti. Tout en gardant, à l'esprit que plusieurs facteurs internes et externes pouvant impacter ces trois grandes contraintes. De même, il n'est pas exclus qu'une contraintes des trois prennent le dessus pour freiner la progression dans la production du produit final.

I.2.2. LE CHEF DU PROJET LOGICIEL

Un chef de projet logiciel, est une personne chargée de la responsabilité d'exécuter les activités du projet logiciel. Il doit cependant posséder les connaissances nécessaires quant aux phases de cycle de vie de développement logiciel qu'il pourra exécuter tout au long du projet. Le chef du projet ne doit jamais être impliqué directement dans la production du produit final plutôt il doit contrôler et gérer toutes les activités impliquées dans la production.

Un chef de projet surveille de près le processus de développement, prépare et exécute divers plans, organise les ressources nécessaires et adéquates, maintient la communication entre tous les membres de l'équipe afin de résoudre les problèmes de coûts, de budget, de ressources, de temps et de satisfaction client. Voyons peu de responsabilités qu'un chef de projet doit assumer :

- Gérer des gens :
  - → Agir comme chef de projet ;
  - → Liaison avec les parties prenantes ;
  - → Gestion des ressources humaines ;
  - → Mise en place d'une hiérarchie de rapports etc.

- Gérer du projet :
  - → Définir et configurer la portée du projet ;
  - → Gestion des activités de gestion de projet ;

→ Suivi des progrès et des performances ;
→ Analyse des risques à chaque phase ;
→ Prendre les mesures nécessaires pour éviter ou sortir des problèmes ;
→ Agir en tant que porte-parole du projet.

I.2.3. LES ACTIVITES DE GESTION DE LOGICIELS

La gestion de projets logiciels comprend un certain nombre d'activités, notamment la planification du projet, la détermination de la portée du produit logiciel, l'estimation du coût en termes divers, la planification des tâches et des événements et la gestion des ressources. Les activités de gestion de projet peuvent inclure :

- Planification de projet ;
- Gestion du cadre de projet ;
- Estimation de projet ;

I.2.3.1. PLANIFICATION DE PROJET

La planification de projet logiciel est une tâche qui est effectuée avant le démarrage effectif du logiciel. Il est là pour la production de logiciels mais n'implique aucune activité concrète ayant un lien direct avec la production de logiciels; c'est plutôt un ensemble de processus multiples, ce qui facilite la production de logiciels. La planification du projet peut inclure les éléments tels que l*a portée du projet ; l'estimation du projet en terme des délais et des coûts* ; etc.

I.2.3.2. GESTION DU CADRE DE PROJET

Elle définit la portée du projet; cela inclut toutes les activités, et le processus de dimensionnement de projet doit être fait pour créer un produit logiciel livrable. La gestion du cadre de projet est essentielle car elle crée des limites au projet en définissant clairement ce qui serait fait dans le projet et ce qui ne serait pas fait. Cela fait que le projet contient des tâches limitées et quantifiables, qui peuvent être facilement documentées et évitent à leur tour les dépassements de coûts et de temps. Au cours de la gestion du cadrage de projet, il est nécessaire de :

- Définir la portée du projet ;
- Décider de sa vérification et de son contrôle ;
- Divisez le projet en différentes parties plus petites pour faciliter la gestion ;
- Vérifier la portée du projet ;
- Contrôler la portée en intégrant les modifications apportées au cadrage du projet.

I.2.3.3.. ESTIMATION DE PROJET

Pour une gestion efficace, une estimation précise des différentes mesures est indispensable. Avec des estimations correctes, les responsables peuvent gérer et contrôler le projet de manière plus efficace. L'estimation du projet peut impliquer les éléments suivants :

- **Estimation de la taille du logiciel :** La taille du logiciel peut être estimée en termes de KLOC (Kilo Line of Code) ou en calculant le nombre de points de fonction dans le logiciel. Les lignes de code dépendent des pratiques de codage et les points de fonction varient en fonction des exigences de l'utilisateur ou du logiciel.

- **Estimation de l'effort** : Les gestionnaires estiment les efforts en termes de personnel et d'heures de main-d'œuvre requis pour produire le logiciel. Pour une estimation de l'effort, la taille du logiciel doit être connue. Cela peut être obtenu par l'expérience des gestionnaires, les données historiques de l'organisation ou la taille du logiciel peuvent être converties en efforts en utilisant certaines formules standards.

- **Estimation du temps :** Une fois la taille et les efforts estimés, le temps nécessaire pour produire le logiciel peut être estimé. Les efforts requis sont séparés en sous-catégories conformément aux spécifications requises et à l'interdépendance des divers composants du logiciel. Les tâches logicielles sont divisées en tâches, activités ou événements plus petits par structure WBS (Work Breakthrough Structure). Les tâches sont planifiées au jour le jour ou en mois calendaire. La somme du temps requis pour effectuer toutes les tâches en heures ou en jours correspond au temps total consacré à la réalisation du projet.

- **Estimation du coût :** Cela peut être considéré comme le plus difficile car il dépend de plus d'éléments que les précédents. Pour estimer le coût du projet, il est nécessaire de prendre en compte :

  → Taille du logiciel ;
  → Qualité du logiciel ;
  → Matériel ;
  → Logiciels ou outils supplémentaires, licences, etc.
  → Personnel qualifié ayant des compétences spécifiques à une tâche ;
  → Voyage impliqué ;
  → la communication ;
  → Formation et support.

I.2.4. TECHNIQUES D'ESTIMATION DE PROJET

Nous avons discuté de divers paramètres impliquant l'estimation de projet tels que la taille, l'effort, le temps et le coût. Le chef de projet peut estimer les facteurs énumérés en utilisant deux techniques largement reconnues :

- **Technique de décomposition** : Cette technique suppose le logiciel en tant que produit de différentes compositions. Il est subdivisé deux modèles principaux :

  → **L'estimation de la ligne de code** est effectuée au nom du nombre de lignes de codes du produit logiciel.

  → **L'estimation Points de fonction** est effectuée pour le compte du nombre de points de fonction du produit logiciel.

- **Technique d'estimation empirique :** Cette technique utilise des formules dérivées empiriquement pour faire des estimations. Ces formules sont basées sur la LOC ou les PF. C'est par exemple :

  → **Modèle Putnam :** Ce modèle est réalisé par Lawrence H. Putnam, basé sur la distribution de fréquence de Norden (courbe de Rayleigh). Le modèle Putnam cartographie le temps et les efforts requis avec la taille du logiciel.

  → **COCOMO** : signifie COnstructive COst MOdel, développé par Barry W. Boehm. Il divise le produit logiciel en trois catégories de logiciels: organique, semi-détaché et intégré.

I.3. PREVISION D'UN PROJET LOGICIEL

La prévision de projet dans un projet fait référence à la feuille de route de toutes les activités à effectuer avec un ordre spécifié et dans un créneau horaire attribué à chaque activité. Les chefs de projet ont tendance à définir différentes tâches et les étapes du projet et à les organiser en tenant compte de divers facteurs. Ils recherchent des tâches se trouvant dans le chemin critique du calendrier, qui doivent être accomplies de manière spécifique (*en raison de l'interdépendance des tâches*) et strictement dans les délais impartis. La disposition des tâches qui sont hors du chemin critique est moins susceptible d'avoir un impact sur tout le calendrier du projet.

Pour faire la prévision d'un projet, il est nécessaire de :

- Décomposer les tâches du projet en une forme plus petite et gérable ;
- Découvrez diverses tâches et corrélez-les ;
- Estimer le délai requis pour chaque tâche ;
- Diviser le temps en unités de travail ;
- Assigner un nombre suffisant d'unités de travail pour chaque tâche ;
- Calculer le temps total requis pour le projet du début à la fin.

I.3.1. LA GESTION DES RESSOURCES

Tous les éléments utilisés pour développer un produit logiciel peuvent être considérés comme des ressources pour ce projet. Cela peut inclure des ressources humaines, des outils de production et des bibliothèques de logiciels. Les ressources sont disponibles en quantité limitée et restent dans l'organisation en tant que pool d'actifs. Le manque de ressources entrave le développement du projet et peut être en retard sur le calendrier.

L'allocation de ressources supplémentaires augmente les coûts de développement. Il est donc nécessaire d'estimer et d'allouer des ressources suffisantes pour le projet. La gestion des ressources comprend :

- Définition du projet d'organisation approprié en créant une équipe de projet et en attribuant des responsabilités à chaque membre de l'équipe ;
- Déterminer les ressources requises à un stade particulier et leur disponibilité ;
- Gérez les ressources en générant des demandes de ressources lorsqu'elles sont requises et en les désallouant lorsqu'elles ne sont plus nécessaires.

I.3.2. GESTION DU RISQUE DE PROJET

La gestion des risques comprend toutes les activités liées à l'identification, à l'analyse et à la prévision des risques prévisibles et non prévisibles dans le projet. Le risque peut inclure les éléments suivants:

- Personnel expérimenté quittant le projet et nouveau personnel entrant ;
- Changement dans la gestion organisationnelle ;
- Exigence de modification ou d'interprétation erronée ;
- Sous-estimation du temps et des ressources requis ;
- Changements technologiques, changements environnementaux, concurrence des entreprises.

Ainsi, Les activités suivantes sont impliquées dans le processus de gestion de risque:

1. **Identification** - Prenez note de tous les risques possibles pouvant survenir dans le projet.
2. **Catégoriser** - Classer les risques connus en intensité de risque élevée, moyenne et faible en fonction de leur impact possible sur le projet.
3. **Gérer** - Analyser la probabilité d'occurrence des risques à différentes phases. Faites un plan pour éviter ou faire face à des risques. Essayez de minimiser leurs effets secondaires.
4. **Surveiller** - Surveiller étroitement les risques potentiels et leurs premiers symptômes. Surveillez également les effets des mesures prises pour les atténuer ou les éviter.

I.3.3. GESTION DE L'EXECUTION ET SURVEILLANCE DU PROJET

Dans cette phase, les tâches décrites dans les plans de projet sont exécutées conformément à leurs calendriers. L'exécution doit être surveillée afin de vérifier si tout se déroule conformément au plan. La surveillance permet de vérifier la probabilité du risque et de prendre des mesures pour faire face au risque ou signaler l'état des différentes tâches. Ces mesures comprennent :

- **Surveillance de l'activité** - Toutes les activités planifiées dans une tâche peuvent être surveillées au quotidien. Lorsque toutes les activités d'une tâche sont terminées, elles sont considérées comme complètes.

- **Rapports de statut** - Les rapports contiennent l'état des activités et des tâches exécutées dans un délai donné, généralement une semaine. Le statut peut être marqué comme terminé, en attente ou en cours, etc.

- **Liste de vérification des jalons** - Chaque projet est divisé en plusieurs phases où les tâches principales sont exécutées (jalons) en fonction des phases du processus SDLC. Cette liste de vérification des étapes est préparée une fois toutes les quelques semaines et indique l'état des jalons.

I.3.4. GESTION DE LA COMMUNICATION DE PROJET

Une communication efficace joue un rôle essentiel dans la réussite d'un projet. Il comble les écarts entre le client et l'organisation, entre les membres de l'équipe et les autres parties prenantes du projet, tels que les fournisseurs de matériel. La communication peut être orale ou écrite. Le processus de gestion de la communication peut comporter les étapes suivantes:

- **Planning** - Cette étape comprend les identifications de tous les acteurs du projet et le mode de communication entre eux. Il considère également si des moyens de communication supplémentaires sont nécessaires.

- **Partage** - Après avoir déterminé divers aspects de la planification, le gestionnaire se concentre sur le partage d'informations correctes avec la bonne personne au moment opportun. Cela permet à chacun d'être impliqué dans le projet de l'avancement du projet et de son statut.

- **Commentaires** - Les chefs de projet utilisent diverses mesures et mécanismes de rétroaction et créent des rapports d'état et de performance. Ce mécanisme garantit que les contributions des différentes parties prenantes parviennent au chef de projet en tant que commentaires.

- **Fermeture** - A la fin de chaque événement majeur, fin d'une phase de SDLC ou fin du projet lui-même, la clôture administrative est officiellement annoncée pour mettre à jour chaque partie en envoyant un courrier électronique, en distribuant une copie papier ou autre moyen de communication efficace.

Après la fermeture, l'équipe passe à la phase suivante ou au projet suivant.

I.3.5. GESTION DE LA CONFIGURATION

La gestion de la configuration est un processus de suivi et de contrôle des modifications logicielles en termes d'exigences, de conception, de fonctions et de développement du produit.

IEEE le définit comme « le processus d'identification et de définition des éléments du système, contrôlant le changement de ces éléments tout au long de leur cycle de vie, enregistrant et signalant l'état des articles et des demandes de modification, et vérifiant l'exhaustivité et l'exactitude des éléments ».

En règle générale, une fois que le SRS est finalisé, il y a moins de chance que des changements soient requis de la part de l'utilisateur. Si elles se produisent, les modifications ne sont traitées qu'avec l'approbation préalable de la haute direction, car il existe une possibilité de dépassement des coûts et des délais.

1. **Ligne de base :** Une phase de SDLC est présumée si elle est basée sur la base, c'est-à-dire que la ligne de base est une mesure qui définit la complétude d'une phase. Une phase est définie lorsque toutes les activités qui s'y rapportent sont terminées et bien documentées. Si ce n'était pas la phase finale, son résultat serait utilisé dans la prochaine phase immédiate. La gestion de la configuration est une discipline de l'administration de l'organisation qui prend en charge l'occurrence de tout changement (processus, exigence, technologie, stratégie, etc.) après la phase de base, CM vérifie les modifications apportées au logiciel.

2. **Le contrôle des changements :** Le contrôle des modifications est une fonction de la gestion de la configuration, qui garantit que toutes les modifications apportées au système logiciel sont cohérentes et effectuées conformément aux règles et réglementations organisationnelles. Un changement dans la configuration du produit passe par les étapes suivantes :

   → **Identification** - Une demande de modification arrive de la source interne ou externe. Lorsque la demande de changement est identifiée officiellement, elle est correctement documentée.

   → **Validation** - La validité de la demande de modification est vérifiée et sa procédure de traitement est confirmée.

   → **Analyse** - L'impact de la demande de modification est analysé en termes de calendrier, de coûts et d'efforts requis. L'impact global du changement potentiel sur le système est analysé.

→ **Contrôle** - Si le changement potentiel a un impact sur un trop grand nombre d'entités dans le système ou s'il est inévitable, il est obligatoire de prendre l'approbation des autorités supérieures avant d'intégrer le changement dans le système. Il est décidé si le changement vaut ou non l'incorporation. Si ce n'est pas le cas, la demande de changement est refusée formellement.

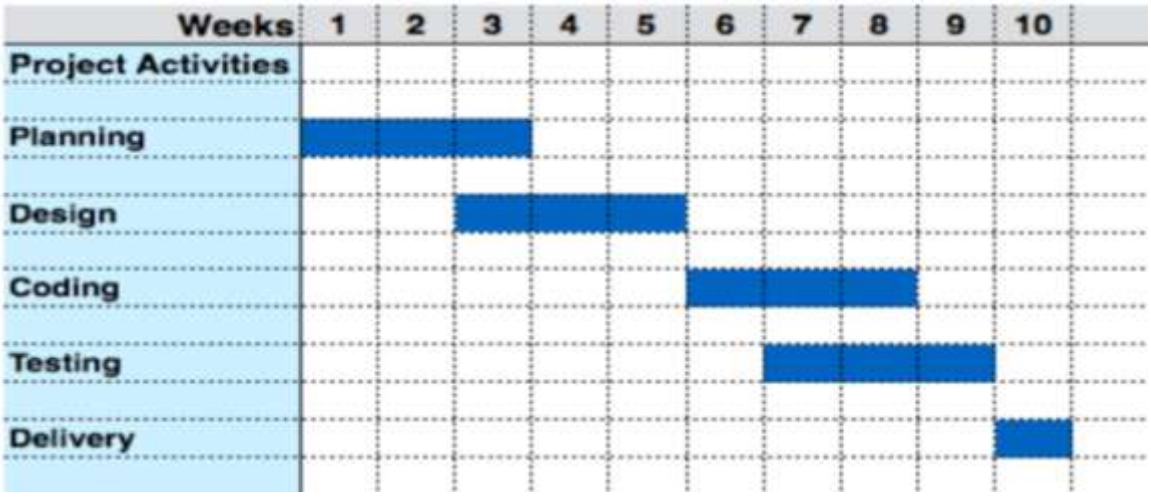
→ **Exécution** - Si la phase précédente décide d'exécuter la demande de modification, cette phase prend les mesures appropriées pour exécuter la modification, effectue une révision approfondie si nécessaire.

→ **Close request** - La modification est vérifiée pour une implémentation correcte et une fusion avec le reste du système. Ce changement nouvellement incorporé dans le logiciel est documenté correctement et la demande est officiellement fermée.

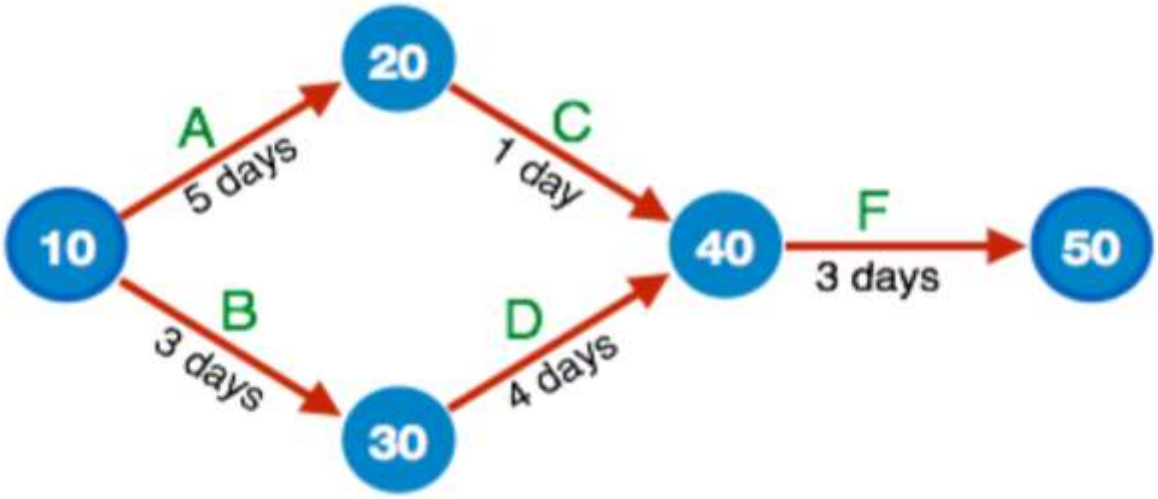
I.4. OUTILS DE GESTION DE PROJET

Le risque et l'incertitude s'accroissent en fonction de la taille du projet, même lorsque le projet est développé selon des méthodologies définies. Des outils sont disponibles pour faciliter une gestion efficace des projets. Quelques-uns sont décrits en l'occurrence :

1. **Diagramme de Gantt :** Les diagrammes de Gantt ont été conçus par Henry Gantt (1917). Il représente le calendrier du projet par rapport aux périodes. Il s'agit d'un graphique à barres horizontales avec des barres représentant les activités et l'heure prévues pour les activités du projet.



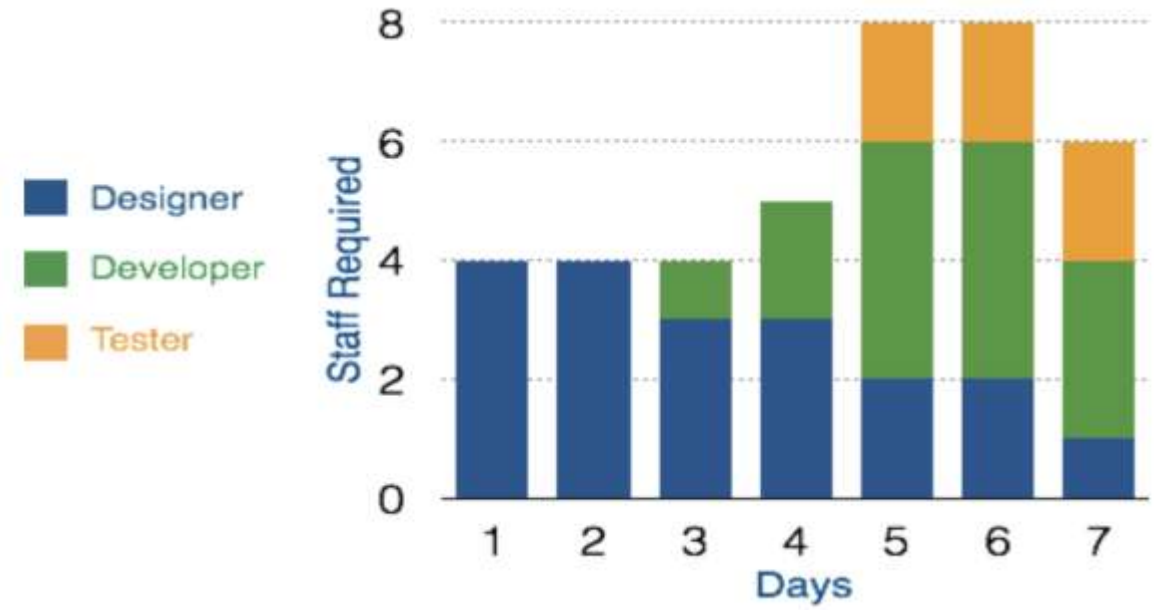
2. **Graphique PERT** : Le diagramme PERT (*Program Evaluation & Review Technique*) est un outil qui décrit le projet en tant que diagramme de réseau. Il est capable de représenter graphiquement les principaux événements du projet de manière parallèle et consécutive. Les événements qui se produisent l'un après l'autre montrent la dépendance de l'événement ultérieur par rapport à l'événement précédent.



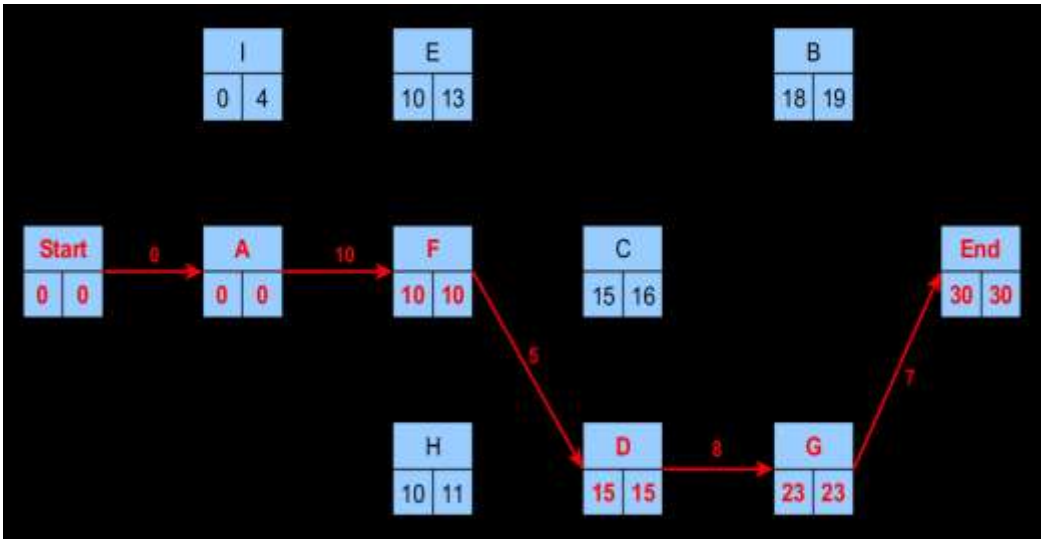
Les événements sont affichés sous forme de nœuds numérotés. Ils sont reliés par des flèches étiquetées décrivant la séquence des tâches du projet.

3. **Histogramme de ressource :** Il s'agit d'un outil graphique contenant des barres ou des diagrammes représentant le nombre de ressources (généralement du personnel qualifié) nécessaires au fil du temps pour un événement (ou une phase) du projet. L'histogramme des ressources est un outil efficace pour la planification et la coordination du personnel.

Staff	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Designer	4	4	3	3	2	2	1
Developer	0	0	1	2	4	4	3
Tester	0	0	0	0	2	2	2
Total	4	4	4	5	8	8	6



4. **Analyse du chemin critique :** Cet outil est utile pour reconnaître les tâches interdépendantes dans le projet. Cela aide également à trouver le chemin le plus court ou le chemin critique pour mener à bien le projet. Comme pour le diagramme PERT, chaque événement se voit attribuer une période spécifique. Cet outil montre la dépendance de l'événement en supposant qu'un événement peut passer à la suivante uniquement si la précédente est terminée.



Les événements sont organisés en fonction de leur heure de début la plus précoce possible. Le chemin entre le nœud de début et le nœud final est un chemin critique qui ne peut être réduit davantage et tous les événements doivent être exécutés dans le même ordre.

CHAPITRE II. LES EXIGENCES LOGICIELLES

Les exigences logicielles sont la description des fonctionnalités et des fonctionnalités du système cible. Les exigences traduisent les attentes des utilisateurs du produit logiciel. Les exigences peuvent être évidentes ou cachées, connues ou inconnues, attendues ou inattendues du point de vue du client.

II.1. LES PROCESSUS DES EXIGENCES LOGICIELLES

Le processus pour rassembler les exigences logicielles du client, les analyser et les documenter est connu sous le nom d'ingénierie des exigences. L'objectif de l'ingénierie des exigences est de développer et de maintenir un document sophistiqué et descriptif de « *spécification des exigences du système* ». C'est un processus en quatre étapes, qui comprend :

- Étude de faisabilité ;
- Rassemblement d'exigences ;
- Spécification des besoins logiciels ;
- Validation des besoins logiciels.

- **Étude de faisabilité**

Lorsque le client contacte l'organisation pour obtenir le produit souhaité, il se fait une idée approximative de toutes les fonctions que le logiciel doit exécuter et de toutes les fonctionnalités attendues du logiciel. Se référant à ces informations, les analystes effectuent une étude détaillée pour déterminer si le système souhaité et ses fonctionnalités sont réalisables.

Cette étude de faisabilité est axée sur l'objectif de l'organisation. Cette étude analyse si le produit logiciel peut être matérialisé en termes d'implémentation, de contribution du projet à l'organisation, de contraintes de coûts et de valeurs et d'objectifs de l'organisation. Il explore les aspects techniques du projet et du produit tels que la facilité d'utilisation, la maintenabilité, la productivité et la capacité d'intégration.

Le résultat de cette phase devrait être un « *rapport d'étude de faisabilité* » qui devrait contenir des commentaires et des recommandations adéquats à l'intention de la direction sur la question de savoir si le projet doit être entrepris ou non.

- **Rassemblement d'exigences**

Si le rapport de faisabilité est favorable à la réalisation du projet, la phase suivante commence par la collecte des exigences de l'utilisateur. Les analystes et les ingénieurs communiquent avec le client et les utilisateurs finaux pour connaître leurs idées sur ce que le logiciel doit fournir et sur les fonctionnalités à inclure dans le logiciel.

- **Spécification des besoins logiciels**

Le SRS (*software requirement specification*) est un document créé par l'analyste du système après que les exigences ont été recueillies auprès de diverses parties prenantes. Le SRS définit la manière dont le logiciel prévu va interagir avec le matériel ; les interfaces externes ; la rapidité de fonctionnement ; le temps de réponse du système ; la portabilité des logiciels sur diverses plates-formes ; facilité de restauration après plantage ; sécurité ; qualité ; limitations ; etc. … Les exigences reçues du client sont écrites en langage naturel. Il incombe à l'analyste du système de documenter les exigences dans un langage technique afin qu'elles puissent être comprises et utiles par l'équipe de développement de logiciels. Le SRS devrait proposer les fonctionnalités suivantes :

- Les exigences de l'utilisateur sont exprimées en langage naturel ;
- Les exigences techniques sont exprimées dans un langage structuré, qui est utilisé dans l'organisation ;
- La description de la conception doit être écrite en pseudo-code ;
- Format d'impression de formulaires et d'interface graphique.
- Notations conditionnelles et mathématiques pour les DFD, etc.

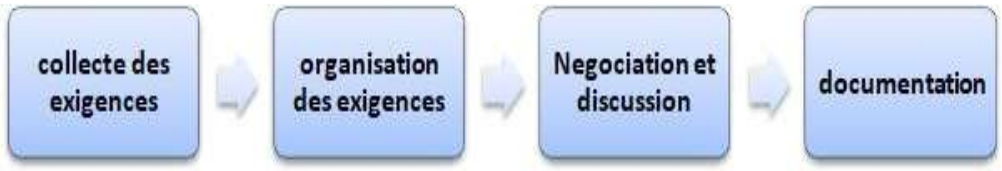
- **Validation des besoins logiciels**

C'est la confirmation de régularité par un client compétent à propos de l'adéquation de produit logiciel. Une fois les spécifications requises développées, les exigences mentionnées dans ce document sont validées. L'utilisateur peut demander une solution illégale, peu pratique ou les experts peuvent interpréter les exigences de manière incorrecte. Cela entraîne une énorme augmentation des coûts si elle n'est pas étouffée dans l'œuf. Les exigences peuvent être vérifiées par rapport aux conditions suivantes :

- S'ils peuvent être mis en pratique ;
- S'ils sont valides et selon la fonctionnalité et le domaine du logiciel ;
- S'il y a des ambiguïtés ;
- S'ils sont complets ;
- S'ils peuvent être démontrés

II.2. LES PROCESSUS DE DEMANDE DES EXIGENCES

Le processus de demande des exigences permet de déterminer les questions relatives à un système logiciel en communiquant avec le client, les utilisateurs finaux, les utilisateurs du système et les autres parties prenantes du développement du système logiciel. Ce processus peut être décrit à l'aide du diagramme suivant :



- **Collecte des exigences -** Les développeurs discutent avec le client et les utilisateurs finaux et connaissent leurs attentes par rapport au logiciel.
- **Exigences d'organisation** - Les développeurs hiérarchisent et ordonnancent les exigences par ordre d'importance, d'urgence et de commodité.
- **Négociation & discussion** - Si les exigences sont ambiguës ou s'il existe des conflits dans les exigences des différentes parties prenantes, si elles le sont, elles sont ensuite négociées et discutées avec les parties prenantes. Les exigences peuvent alors être hiérarchisées et raisonnablement compromises. Les exigences proviennent de différentes parties prenantes. Pour lever l'ambiguïté et les conflits, ils sont discutés pour plus de clarté et d'exactitude. Les exigences irréalistes sont compromises raisonnablement.
- **Documentation** - Toutes les exigences formelles et informelles, fonctionnelles et non fonctionnelles sont documentées et mises à disposition pour le traitement de la phase suivante.

Il existe différentes manières de découvrir les exigences logicielles :

- **Les interviews -** sont un moyen fort pour recueillir les besoins du client. L'organisation peut mener plusieurs types d'entretiens tels que:

  → **Entretiens structurés (fermés),** où chaque information à rassembler est décidée à l'avance, ils suivent fermement le modèle et la question de la discussion.

  → **Entretiens non structurés (ouverts),** où les informations à rassembler ne sont pas décidées à l'avance, plus flexibles et moins biaisées.

  → **Entretiens oraux ;**

  → **Entretiens écrits ;**

→ **Entrevues individuelles** entre deux personnes de l'autre côté de la table.

→ **Entretiens de groupe** qui ont lieu entre des groupes de participants. Ils aident à découvrir toute exigence manquante car de nombreuses personnes sont impliquées.

- **Les Enquêtes -** L'organisation peut mener des enquêtes auprès de diverses parties prenantes en interrogeant leurs attentes et les exigences du système à venir.

- **Les Questionnaires** - Un document avec un ensemble prédéfini de questions objectives et d'options respectives est remis à toutes les parties prenantes pour répondre, qui sont collectées et compilées. Un inconvénient de cette technique est que, si une option pour certains problèmes n'est pas mentionnée dans le questionnaire, le problème pourrait être laissé sans surveillance.

- **Analyse des tâches** - Une équipe d'ingénieurs et de développeurs peut analyser l'opération pour laquelle le nouveau système est requis. Si le client dispose déjà d'un logiciel pour effectuer certaines opérations, il est étudié et les exigences du système proposé sont collectées.

- **Analyse de domaine** - Chaque logiciel appartient à une catégorie de domaine. Les experts du domaine peuvent être d'une grande aide pour analyser les besoins généraux et spécifiques.

- **Brainstorming -** Un débat informel a lieu entre les différentes parties prenantes et toutes leurs contributions sont enregistrées pour une analyse plus approfondie des besoins.

- **Prototypage -** Le prototypage consiste à créer une interface utilisateur sans ajouter de fonctionnalités détaillées permettant à l'utilisateur d'interpréter les fonctionnalités du produit logiciel prévu. Cela aide à donner une meilleure idée des exigences. Si aucun logiciel n'est installé à la fin du client pour la référence du développeur et que le client n'a pas connaissance de ses propres exigences, le développeur crée un prototype basé sur les exigences initialement mentionnées. Le prototype est montré au client et le retour est noté. La rétroaction du client sert d'intrant pour la collecte des exigences.

- **Observation -** Une équipe d'experts visite l'organisation ou le lieu de travail du client. Ils observent le fonctionnement réel des systèmes installés existants. Ils observent le « *workflow* » à la fin du client et comment les problèmes d'exécution sont traités. L'équipe elle-même tire certaines conclusions qui aident à former les exigences attendues du logiciel.

II.3. CARACTERISTIQUES DES EXIGENCES LOGICIELLES

La collecte des logiciels requis constitue la base de tout le projet de développement logiciel. Une spécification de logiciel complète doit être :

- Clair ;
- Correct ;
- Cohérent ;
- Cohérent ;
- Compréhensible ;
- Modifiable ;
- Vérifiable ;
- Priorisé ;
- Non ambigu ;
- Traçable ;
- Source crédible

Lors de la phase de la demande des exigences ; les caractéristiques des exigences logicielles sont de deux grandes catégories :

- Les caractéristiques des exigences fonctionnelles ;
- Les caractéristiques des exigences non-fonctionnelles.

II.3.1. LES CARACTERISTIQUES DES EXIGENCES FONCTIONNELLES

Ce sont des exigences liées à l'aspect fonctionnel (logique et rationnel) du logiciel entrent dans cette catégorie. Ils définissent des fonctions et des fonctionnalités au sein et à partir du système logiciel.

Exemples :

- Option de recherche donnée à l'utilisateur pour rechercher à partir de différentes factures ;
- L'utilisateur doit pouvoir envoyer tout rapport à la direction ;
- Les utilisateurs peuvent être divisés en groupes et les groupes peuvent recevoir des droits distincts ;
- Devrait respecter les règles métier et les fonctions administratives.
- Le logiciel est développé en conservant la compatibilité descendante.

II.3.2. LES CARACTERISTIQUES DES EXIGENCES NON-FONCTIONNELLES

Ce sont des exigences, qui ne sont pas liées à l'aspect fonctionnel du logiciel, toutefois qui entrent dans cette catégorie. Ce sont des caractéristiques implicites (sous-entendues et conventionnelles) ou attendues du logiciel, dont les utilisateurs font des suppositions. Les caractéristiques des exigences non fonctionnelles incluent :

- Sécurité ;
- Enregistrement ;
- Espace de rangement ;
- Configuration ;
- Performance ;
- Coût ;
- Interopérabilité ;
- La flexibilité ;
- Rétroaction du logiciel;
- Accessibilité.

Mise à part, les deux grandes catégories des caractéristiques des exigences logicielles ; les logiciels sont également classés logiquement selon 4 principes :

- **Ce qu'il doit avoir** : Le logiciel ne peut pas être dit opérationnel sans eux ;
- **Ce qu'il devrait avoir** : Améliorer la fonctionnalité du logiciel ;
- **Ce qu'il pourrait avoir** : Le logiciel peut toujours fonctionner correctement avec ces exigences ;
- **Ce qu'est la Liste de souhaits**: ces exigences ne correspondent à aucun objectif du logiciel.

Ainsi, Tout en développant un logiciel, il faut mettre en œuvre *« ce qu'il doit avoir »,* et « *ce qu'il devrait être* » sont un sujet de débat avec les parties prenantes et la négation, alors que «*ce qu'il pourrait avoir* » et « *liste de souhaits* » peuvent être conservé pour les mises à jour logicielles.

II.4. LES EXIGENCES REQUISES DE L'INTERFACE UTILISATEUR

L'interface utilisateur est une partie importante de tout logiciel, matériel ou système hybride. Un logiciel est largement accepté si son interface de l'utilisateur est :

- facile à utiliser ;
- rapide en réponse ;
- gérer efficacement les erreurs opérationnelles ;
- simple mais cohérente.

L'acceptation de l'interface de l'utilisateur dépend principalement de la manière dont l'utilisateur peut utiliser le logiciel. L'interface utilisateur est le seul moyen pour les utilisateurs de percevoir le système. Un système logiciel performant doit également être équipé d'une interface utilisateur ***attrayante, claire, cohérente et réactive***. Sinon, les fonctionnalités du système logiciel ne peuvent pas être utilisées de manière pratique. Un système est dit bon, s'il fournit des moyens de l'utiliser efficacement. Les exigences de l'interface utilisateur sont brièvement mentionnées ci-dessous :

- Présentation du contenu ;
- Navigation facile ;
- Interface simple ;
- Sensible ;
- Éléments d'interface utilisateur cohérents ;
- Mécanisme de rétroaction ;
- Paramètres par défaut ;
- Mise en page utile ;
- Utilisation stratégique de la couleur et de la texture ;
- Fournir des informations d'aide ;
- Approche centrée sur l'utilisateur ;
- Paramètres de vue basés sur un groupe.

II.5. ANALYSTE DE LOGICIELS

L'analyste système dans une organisation informatique est une personne qui analyse les exigences du système proposé et s'assure que les exigences sont conçues et documentées raisonnablement et correctement. Le rôle d'un analyste commence lors de la phase d'analyse logicielle de Cycle de vie de développement logiciel. Il incombe à l'analyste de s'assurer que le logiciel développé répond aux exigences du client.

Les analystes système ont les responsabilités suivantes :

- Analyser et comprendre les exigences du logiciel prévu ;
- Comprendre comment le projet contribuera aux objectifs de l'organisation ;
- Identifier les sources d'exigence ;
- Validation de l'exigence ;
- Élaborer et mettre en œuvre un plan de gestion des exigences ;
- Documentation des exigences métier, techniques, processus et produits ;
- Coordination avec les clients pour hiérarchiser les exigences et supprimer les ambiguïtés ;
- Finalisation des critères d'acceptation avec le client et les autres parties prenantes

II.6. METRIQUES ET MESURES LOGICIELLES

Les mesures logicielles peuvent être comprises comme un processus de quantification et de symbolisation des divers attributs et aspects du logiciel. **Les métriques logicielles** fournissent des mesures pour divers aspects du processus logiciel et du produit logiciel.

Les mesures logicielles sont une exigence fondamentale du génie logiciel. Ils aident non seulement à contrôler le processus de développement logiciel, mais aident également à maintenir la qualité du produit final excellente. Selon **Tom DeMarco**, un (ingénieur logiciel), « *vous ne pouvez pas contrôler ce que vous ne pouvez pas mesurer* ». Selon lui, l'importance des mesures logicielles est très claire. Voyons quelques métriques logicielles :

- **Mesures de taille - LOC (lignes de code),** principalement calculées en milliers de lignes de code source, dénommées KLOC. Le nombre de points de fonction est une mesure de la fonctionnalité fournie par le logiciel. Le nombre de points de fonction définit la taille de l'aspect fonctionnel du logiciel.

- **Métriques de complexité** - La complexité cyclomatique de McCabe quantifie la limite supérieure du nombre de chemins indépendants dans un programme, ce qui est perçu comme une complexité du programme ou de ses modules. Il est représenté en termes de concepts de théorie des graphes en utilisant un graphe de flux de contrôle.

- **Paramètres de qualité** - Les défauts, leurs types et leurs causes, leurs conséquences, leur intensité et leurs implications définissent la qualité du produit. Le nombre de défauts détectés dans le processus de développement et le nombre de défauts signalés par le client après l'installation ou la livraison du produit chez le client définissent la qualité du produit.

- **Métriques de processus** - Au cours des différentes phases de SDLC, les méthodes et les outils utilisés, les normes de l'entreprise et les performances du développement sont des paramètres de processus logiciels.

- **Métriques de ressource** - Effort, temps et diverses ressources utilisées, représentent des mesures pour la mesure des ressources.

CHAPITRE III. LES BASES DE CONCEPTION DE LOGICIELS

La conception de logiciels est un processus permettant de transformer les besoins des utilisateurs en une forme appropriée, ce qui aide le programmeur à coder et à mettre en œuvre les logiciels.

Pour évaluer les besoins des utilisateurs, un document SRS (*Software Requirement Specification*) est créé alors que pour le codage et la mise en œuvre, des exigences plus spécifiques et détaillées en termes de logiciels sont nécessaires. La sortie de ce processus peut être directement utilisée dans l'implémentation dans les langages de programmation. La conception logicielle est la première étape de SDLC (*Software Design Life Cycle*), qui déplace la concentration du domaine problématique au domaine solution. Il essaie de spécifier comment satisfaire aux exigences mentionnées dans SRS.

III.1. NIVEAUX DE CONCEPTION DE LOGICIELS

La conception du logiciel donne trois niveaux de résultats :

- **Conception architecturale** - La conception architecturale est la version abstraite la plus élevée du système. Il identifie le logiciel en tant que système avec de nombreux composants interagissant les uns avec les autres. A ce niveau, les concepteurs ont l'idée du domaine de solution proposé.

- **Conception de haut niveau** - La conception de haut niveau divise le concept de conception architecturale «mono-entité-multiple» en une vue moins abstraite des sous-systèmes et des modules et décrit leur interaction. La conception de haut niveau se concentre sur la manière dont le système et tous ses composants peuvent être implémentés dans des formes de modules. Il reconnaît la structure modulaire de chaque sous-système et leur relation et interaction entre eux.

- **Conception détaillée-** La conception détaillée traite de la partie mise en œuvre de ce qui est considéré comme un système et de ses sous-systèmes dans les deux conceptions précédentes. Il est plus détaillé vers les modules et leurs implémentations. Il définit la structure logique de chaque module et de leurs interfaces pour communiquer avec les autres modules.

III.2. LA MODULARISATION

La modularisation est une technique permettant de diviser un système logiciel en plusieurs modules discrets et indépendants, susceptibles de réaliser des tâches de manière indépendante. Ces modules peuvent fonctionner comme des constructions de base pour l'ensemble du logiciel.

Les concepteurs ont tendance à concevoir des modules de manière à ce qu'ils puissent être exécutés et / ou compilés séparément et indépendamment. La conception modulaire suit involontairement les règles de la stratégie de résolution de problèmes « **diviser pour régner** », car la conception modulaire d'un logiciel présente de nombreux autres avantages en l'occurrence :

- Les plus petits composants sont plus faciles à entretenir ;
- Le programme peut être divisé en fonction des aspects fonctionnels ;
- Le niveau d'abstraction souhaité peut être introduit dans le programme ;
- Les composants à haute cohésion peuvent être réutilisés ;
- L'exécution simultanée peut être rendue possible ;
- Souhaité de l'aspect sécurité.

III.3. CONCURRENCE

Dans le passé, tous les logiciels étaient destinés à être exécutés séquentiellement. Par exécution séquentielle, nous entendons que l'instruction codée sera exécutée l'une après l'autre, ce qui implique qu'une partie seulement du programme était activée à un moment donné. Disons qu'un logiciel possède plusieurs modules, alors qu'un seul de tous les modules peut être trouvé actif à tout moment de l'exécution.

Dans la conception de logiciels, la concurrence est la mise en œuvre en divisant le logiciel en plusieurs unités d'exécution indépendantes, telles que des modules et en les exécutants en parallèle. En d'autres termes, la concurrence fournit au logiciel la possibilité d'exécuter plus d'une partie du code en parallèle. Les programmeurs et les concepteurs doivent reconnaître ces modules qui peuvent être exécutés en parallèle. C'est par Exemple : La fonction de vérification orthographique du traitement de texte est un module de logiciel qui se trouve à côté du traitement de texte lui-même.

III.4. COUPLAGE ET COHESION

Lorsqu'un logiciel est modularisé, ses tâches sont divisées en plusieurs modules basés sur certaines caractéristiques. Comme nous le savons, les modules sont un ensemble d'instructions réunies pour accomplir certaines tâches. Ils sont cependant considérés comme une entité unique mais peuvent se référer les uns aux autres pour travailler ensemble. Il existe des mesures permettant de mesurer la qualité de la conception des modules et leur interaction. Ces mesures sont appelées couplage et cohésion.

III.4.1. COHESION

La cohésion est une mesure qui définit le degré de fiabilité interne des éléments d'un module. Plus la cohésion est grande, meilleure est la conception du programme. Il existe sept types de cohésion, à savoir :

- **Cohésion coïncidente** - Il s'agit d'une cohésion non planifiée et aléatoire, qui pourrait résulter de la division du programme en modules plus petits pour la modularisation. Parce qu'il n'est pas planifié, il peut être source de confusion pour les programmeurs et n'est généralement pas accepté.
- **Cohésion logique** - Lorsque des éléments logiquement classés sont regroupés dans un module, on parle de cohésion logique.
- **Cohésion temporelle** - Lorsque des éléments de module sont organisés de manière à être traités à un moment similaire, on parle de cohésion temporelle.
- **Cohésion procédurale** - Lorsque des éléments de module sont regroupés, qui sont exécutés séquentiellement pour effectuer une tâche, on parle de cohésion procédurale.
- **Cohésion communicationnelle** - Lorsque des éléments de module sont regroupés, exécutés de manière séquentielle et œuvrant sur les mêmes données (informations), on parle de cohésion communicationnelle.
- **Cohésion séquentielle** - Lorsque des éléments d'un module sont regroupés parce que la sortie d'un élément sert d'entrée à un autre et ainsi de suite, on parle de cohésion séquentielle.
- **Cohésion fonctionnelle** - Il est considéré comme le plus haut degré de cohésion, et il est fortement attendu. Les éléments de module en cohésion fonctionnelle sont regroupés car ils contribuent tous à une seule fonction bien définie. Il peut également être réutilisé.

III.4.1. COUPLAGE

Le couplage est une mesure qui définit le niveau d'inter-fiabilité entre les modules d'un programme. Il indique à quel niveau les modules interfèrent et interagissent les uns avec les autres. Plus le couplage est faible, meilleur est le programme. Il existe cinq niveaux de couplage, à savoir -

- **Couplage de contenu** - Lorsqu'un module peut accéder directement au contenu d'un autre module, le modifier ou s'y référer, il est appelé couplage au niveau du contenu.
- **Couplage commun** - Lorsque plusieurs modules ont un accès en lecture et en écriture à certaines données globales, on parle de couplage commun ou global.
- **Couplage de contrôle** - Deux modules sont appelés couplés au contrôle si l'un d'eux décide de la fonction de l'autre module ou modifie son flux d'exécution.
- **Couplage de timbres** - Lorsque plusieurs modules partagent une structure de données commune et travaillent sur des parties différentes, cela s'appelle le couplage de timbres.
- **Couplage de données** - Le couplage de données se produit lorsque deux modules interagissent les uns avec les autres en passant des données (en tant que paramètre). Si un module transmet la structure de données en paramètre, le module récepteur doit utiliser tous ses composants.

Idéalement, aucun couplage n'est considéré comme le meilleur.

III.5. VERIFICATION DE LA CONCEPTION

Le résultat du processus de conception logicielle est la documentation de conception, les pseudo-codes, les schémas logiques détaillés, les diagrammes de processus et une description détaillée de toutes les exigences fonctionnelles ou non fonctionnelles. La phase suivante, qui est la mise en œuvre du logiciel, dépend de toutes les sorties mentionnées ci-dessus.

Il est alors nécessaire de vérifier la sortie avant de passer à la phase suivante. Plus tôt une erreur est détectée, plus elle est efficace ou peut ne pas être détectée avant le test du produit. Si les résultats de la phase de conception sont sous forme de notation formelle, leurs outils de vérification associés doivent alors être utilisés, sinon une revue de conception approfondie peut être utilisée pour la vérification et la validation. Par une approche de vérification structurée, les examinateurs peuvent détecter les défauts qui pourraient être causés par la négligence de certaines conditions. Une bonne revue de conception est importante pour une bonne conception, précision et qualité du logiciel.

CHAPITRE IV. OUTILS D'ANALYSE ET DE CONCEPTION DE LOGICIELS

L'analyse et la conception logicielles incluent toutes les activités qui facilitent la transformation de la spécification des exigences en implémentation. Les spécifications des exigences spécifient toutes les attentes fonctionnelles et non fonctionnelles du logiciel. Ces spécifications se présentent sous la forme de documents lisibles et compréhensibles par l'homme, auxquels un ordinateur n'a rien à voir. L'analyse et la conception logicielles constituent l'étape intermédiaire qui permet de transformer les exigences lisibles par l'homme en code réel. Voyons quelques outils d'analyse et de conception utilisés par les concepteurs de logiciels :

IV.1. DIAGRAMME DE FLUX DE DONNEES

Le diagramme de flux de données est une représentation graphique du flux de données dans un système d'information. Il est capable de décrire le flux de données entrant, le flux de données sortant et les données stockées. Le DFD ne mentionne rien sur la manière dont les données circulent dans le système. Il y a une différence importante entre DFD et Organigramme. L'organigramme décrit le flux de contrôle dans les modules de programme. Les DFD décrivent le flux de données dans le système à différents niveaux. DFD ne contient aucun élément de contrôle ou de branche.

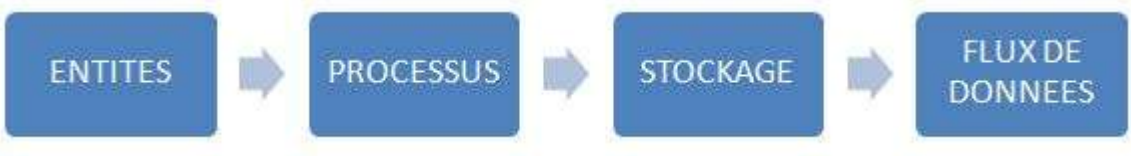
IV.1.1. TYPES DE DFD

Les diagrammes de flux de données sont **logiques** ou **physiques.**

- **Le DFD logique** - Ce type de DFD se concentre sur le processus système et le flux de données dans le système. Par exemple, dans un système de logiciel bancaire, comment les données sont déplacées entre différentes entités.
- **Le DFD physique** - Ce type de DFD montre comment le flux de données est réellement implémenté dans le système. Il est plus spécifique et proche de la mise en œuvre.

IV.1.2. COMPOSANTS DE DFD

Le DFD peut représenter la source, la destination, le stockage et le flux de données en utilisant l'ensemble de composants suivant :

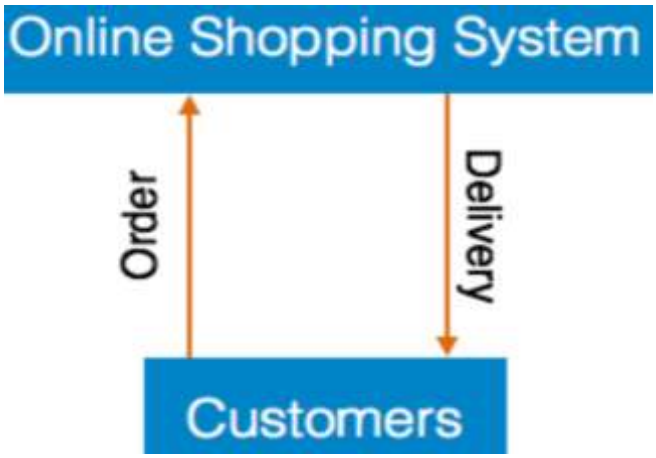


- **Entités** - Les entités sont la source et la destination des données d'information. Les entités sont représentées par des rectangles avec leurs noms respectifs.
- **Processus** - Les activités et les actions effectuées sur les données sont représentées par des rectangles à bords arrondis.
- **Stockage de données** - Il existe deux variantes de stockage de données: il peut être représenté sous la forme d'un rectangle dépourvu de petits côtés ou d'un rectangle ouvert sur lequel n'existe qu'un seul côté.
- **Flux de données (Data flow)** - Le mouvement des données est indiqué par des flèches pointues. Le mouvement des données est indiqué à partir de la base de la flèche comme source vers la tête de la flèche en tant que destination.

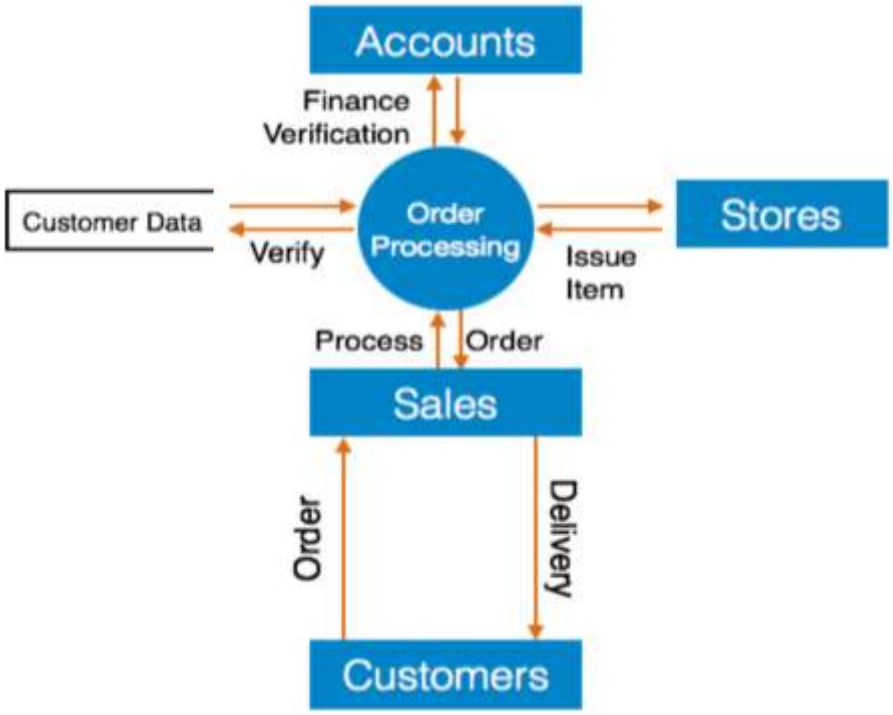
IV.1.3. NIVEAUX DE DFD

Il existe trois niveaux du diagramme de flux des données :

- **Niveau 0** - Niveau d'abstraction le plus élevé Le DFD est connu sous le nom de DFD de niveau 0, qui décrit l'ensemble du système d'information sous la forme d'un diagramme masquant tous les détails sous-jacents. Les DFD de niveau 0 sont également appelés DFD au niveau du contexte.



- **Niveau 1** - Le DFD de niveau 0 est divisé en DFD de niveau 1 plus spécifique. Le DFD de niveau 1 décrit les modules de base du système et le flux de données entre différents modules. Niveau 1 DFD mentionne également les processus de base et les sources d'information.



- **Niveau 2** - À ce niveau, le DFD montre comment les données circulent dans les modules mentionnés au niveau 1.

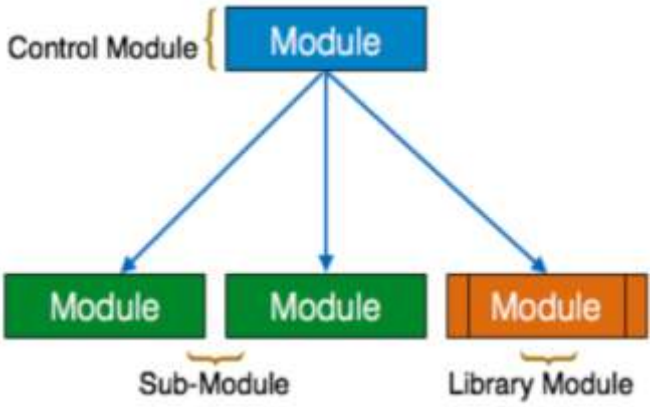
Les DFD de niveau supérieur peuvent être transformés en DFD de niveau inférieur plus spécifique avec un niveau de compréhension plus profond à moins que le niveau de spécification souhaité ne soit atteint.

IV.2. LES ORGANIGRAMMES

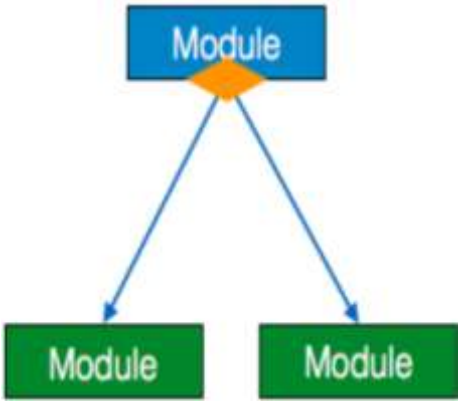
Les organigrammes ou les diagrammes de structures sont des graphiques dérivé du diagramme de flux de données. Il représente le système plus en détail que DFD. Il décompose le système entier en modules fonctionnels les plus bas, décrit plus en détail les fonctions et sous-fonctions de chaque module du système que DFD.

Le diagramme de structure représente la structure hiérarchique des modules. A chaque couche, une tâche spécifique est effectuée. Voici les symboles utilisés dans la construction des diagrammes de structure :

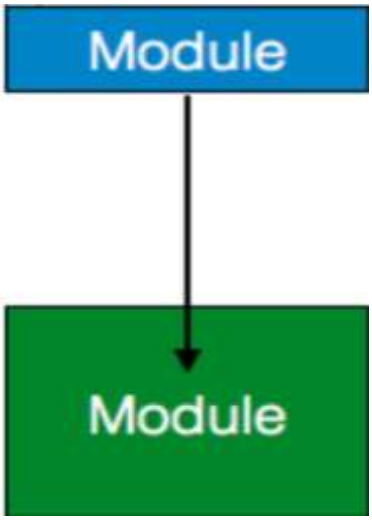
- **Module** - Représente un processus ou une sous-routine ou une tâche. Un module de contrôle se branche sur plusieurs sous-modules. Les modules de bibliothèque sont réutilisables et invocables à partir de n'importe quel module.



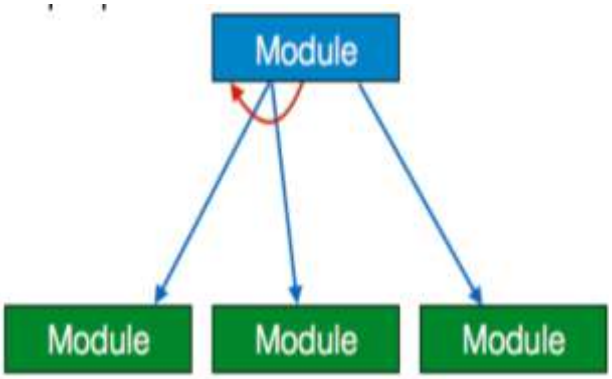
- **Condition** - Il est représenté par un petit diamant à la base du module. Il montre que le module de contrôle peut sélectionner n'importe quelle sous-routine en fonction de certaines conditions.



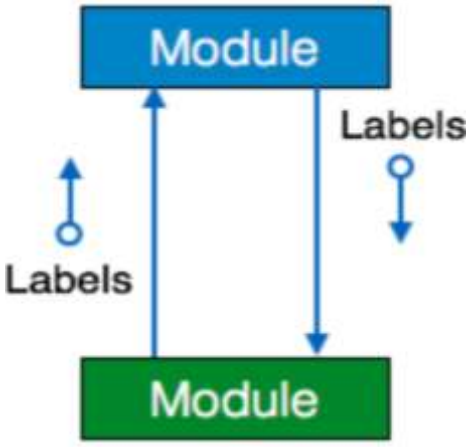
- **Jump** - Une flèche apparaît dans le module pour indiquer que le contrôle sautera au milieu du sous-module.



- **Loop** - Une flèche courbe représente une boucle dans le module. Tous les sous-modules couverts par l'exécution de répétition de boucle du module.



- **Flux de données** - Une flèche dirigée avec un cercle vide à la fin représente le flux de données.



- **Flux de contrôle** - Une flèche dirigée avec un cercle rempli à la fin représente le flux de contrôle.

IV.3. DIAGRAMME HIPO

Le diagramme HIPO (***Hierarchical Input Process Output***) est une combinaison de deux méthodes organisées pour analyser le système et fournir les moyens de documentation. Le modèle HIPO a été développé par IBM en 1970. Le diagramme HIPO représente la hiérarchie des modules dans le système logiciel. Analyst utilise le diagramme HIPO pour obtenir une vue de haut niveau des fonctions du système. Il décompose les fonctions en sous-fonctions de manière hiérarchique. Il décrit les fonctions exécutées par le système.

Les diagrammes HIPO sont bons à des fins de documentation. Leur représentation graphique permet aux concepteurs et aux gestionnaires d'obtenir plus facilement l'idée graphique de la structure du système.

Contrairement au diagramme IPO (*Input Process Output*), qui décrit le flux de contrôle et de données dans un module, HIPO ne fournit aucune information sur le flux de données ou le flux de contrôle.

Exemple : Les deux parties du diagramme HIPO, la présentation hiérarchique et le graphique IPO sont utilisées pour la conception de la structure du logiciel ainsi que pour la documentation de celui-ci.

IV.4. ANGLAIS STRUCTURE

La plupart des programmeurs ne sont pas au courant de la grande image des logiciels, ils ne comptent donc que sur ce que leurs gestionnaires leur ont demandé de faire. Il est de la responsabilité de la gestion logicielle supérieure de fournir des informations précises aux programmeurs afin de développer un code précis et rapide. D'autres formes de méthodes, utilisant des graphiques ou des diagrammes, peuvent parfois être interprétées différemment par des personnes différentes. Par conséquent, les analystes et les concepteurs du logiciel proposent des outils tels que l'anglais structuré. Ce n'est rien d'autre que la description de ce qu'il faut coder et comment le coder. Anglais structuré aide le programmeur à écrire un code sans erreur.

D'autres formes de méthodes, utilisant des graphiques ou des diagrammes, peuvent parfois être interprétées différemment par différentes personnes. Ici, l'anglais structuré et le pseudo-code tentent à la fois d'atténuer ce manque de compréhension. L'anglais structuré est le fait qu'il utilise des mots anglais simples dans le paradigme de la programmation structurée. Ce n'est pas le code ultime mais une sorte de description de ce qui est requis pour coder et comment le coder. Voici quelques jetons de programmation structurée :

- IF-THEN-ELSE,
- DO-WHILE-UNTIL

L'analyste utilise la même variable et le même nom de données, stockés dans le dictionnaire de données, ce qui simplifie considérablement l'écriture et la compréhension du code. **Exemple** : Nous prenons le même exemple d'authentification client dans l'environnement de magasinage en ligne. Cette procédure d'authentification du client peut être écrite en anglais structuré comme suit :

```
Entrez le nom du client
SEEK Customer_Name dans le fichier Customer_Name_DB
SI Customer_Name trouvé
Procédure d'appel USER_PASSWORD_AUTHENTICATE ()
AUTRE
  Message d'erreur d'impression
  Procédure d'appel NEW_CUSTOMER_REQUEST ()
FIN SI
```

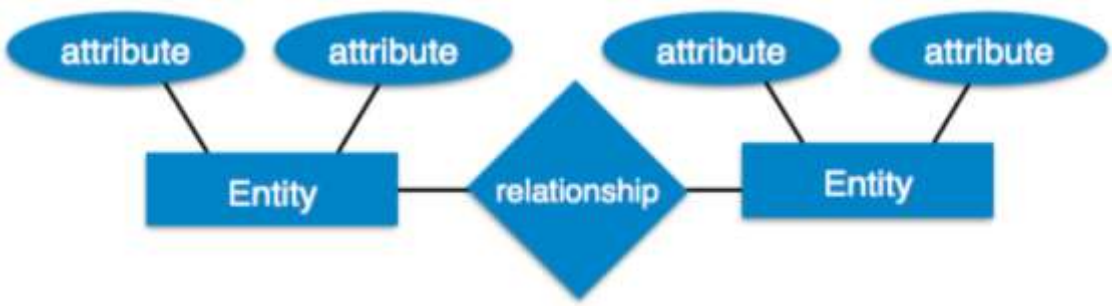
Le code écrit en anglais structuré ressemble plus à l'anglais parlé au quotidien. Il ne peut pas être implémenté directement en tant que code de logiciel. L'anglais structuré est indépendant du langage de programmation.

IV.5. PSEUDO-CODE

Le pseudo-code est écrit plus près du langage de programmation. Il peut être considéré comme un langage de programmation augmenté, plein de commentaires et de descriptions. Le pseudo-code évite la déclaration de variable mais ils sont écrits en utilisant certaines constructions du langage de programmation, comme C, Fortran, Pascal, etc. ... Le pseudo-code contient plus de détails de programmation que l'anglais structuré. Il fournit une méthode pour effectuer la tâche, comme si un ordinateur exécutait le code.

IV.6. MODELE D'ENTITE-RELATION

Le modèle Entité-Relation est un type de modèle de base de données basé sur la notion d'entités du monde réel et la relation entre elles. Nous pouvons cartographier le scénario du monde réel sur le modèle de base de données ER. Le Modèle ER crée un ensemble d'entités avec leurs attributs, un ensemble de contraintes et de relations entre eux.



Le modèle ER est utilisé au mieux pour la conception conceptuelle de la base de données. Le modèle ER peut être représenté comme suit :

- **Entity** - Une entité dans ER Model est un monde réel, qui possède des propriétés appelées attributs. Chaque attribut est défini par son ensemble de valeurs correspondant, appelé domaine. Par exemple, considérons une base de données scolaire. Ici, un étudiant est une entité. L'élève a différents attributs comme le nom, l'identifiant, l'âge et la classe, etc.

- **Relation** - L'association logique entre entités est appelée relation. Les relations sont cartographiées avec des entités de différentes manières. Les cardinalités de mappage définissent le nombre d'associations entre deux entités.

Cardinalités de cartographie :

- Un par un ;
- un à plusieurs ;
- plusieurs à un ;
- plusieurs à plusieurs.

IV.7. DICTIONNAIRE DE DONNEES

Le dictionnaire de données est la collecte centralisée d'informations sur les données. Il stocke la signification et l'origine des données, leur relation avec d'autres données, le format des données à utiliser, etc. Le dictionnaire de données a des définitions rigoureuses de tous les noms afin de faciliter la conception des utilisateurs et des logiciels. Le dictionnaire de données est souvent appelé référentiel de métadonnées (données sur les données). Il est créé avec le modèle DFD (Diagramme de flux de données) du logiciel et doit être mis à jour chaque fois que DFD est modifié ou mis à jour.

IV.7.1. EXIGENCE DU DICTIONNAIRE DE DONNEES

Les données sont référencées via un dictionnaire de données lors de la conception et de la mise en œuvre de logiciels. Le dictionnaire de données supprime toute possibilité d'ambiguïté. Il aide à garder le travail des programmeurs et des concepteurs synchronisé tout en utilisant la même référence d'objet partout dans le programme. Le dictionnaire de données fournit un moyen de documentation pour le système de base de données complet en un seul endroit. La validation du DFD est effectuée à l'aide du dictionnaire de données.

IV.7.2. CONTENU

Le dictionnaire de données doit contenir des informations sur les éléments suivants :

- Flux de données
- Structure de données
- Éléments de données
- Magasins de données
- Traitement de l'information

Le flux de données est décrit au moyen de DFD étudiés précédemment et représentés sous forme algébrique comme décrit.

= Composé de
{} Répétition
() Optionnel
+ Et
[/] Ou

Exemple : Adresse = Maison No + (Rue / Zone) + Ville + État

ID du cours = Numéro du cours + Nom du cours + Niveau du cours + Notes du cours

IV.7.3. ÉLEMENTS DE DONNEES

Les éléments de données se composent du nom et des descriptions des données et des éléments de contrôle, des magasins de données internes ou externes, etc. avec les détails suivants :

- Nom primaire
- Nom secondaire (alias)
- Cas d'utilisation (comment et où utiliser)
- Description du contenu (notation, etc.)
- Informations complémentaires (valeurs prédéfinies, contraintes, etc.)

IV.7.4. MAGASIN DE DONNEES

Il stocke les informations à partir desquelles les données entrent dans le système et existe en dehors du système. Le magasin de données peut inclure :

- **Des dossiers**
  - Interne au logiciel.
  - Externe au logiciel mais sur la même machine.
  - Externe au logiciel et au système, situé sur une machine différente.

- Les tables
  - Convention de nommage
  - Propriété d'indexation

IV.7.5. TRAITEMENT DE L'INFORMATION

Il existe deux types de traitement de données :

- Logique: comme l'utilisateur le voit ;
- Physique: comme le logiciel le voit.

CHAPITRE V. STRATEGIES DE CONCEPTION DE LOGICIELS

La conception logicielle est un processus visant à conceptualiser les exigences logicielles en implémentation logicielle. La conception logicielle prend en compte les exigences des utilisateurs en tant que défis et tente de trouver une solution optimale. Pendant que le logiciel est en cours de conceptualisation, un plan est élaboré pour trouver la meilleure conception possible pour la mise en œuvre de la solution prévue. Il existe plusieurs variantes de conception de logiciel. Laissez-nous les étudier brièvement :

V.1. CONCEPTION STRUCTUREE

La conception structurée est une conceptualisation du problème en plusieurs éléments de solution bien organisés. Il est essentiellement concerné par la conception de la solution. L'avantage de la conception structurée est qu'elle permet de mieux comprendre comment le problème est résolu. La conception structurée permet également au concepteur de se concentrer plus précisément sur le problème. La conception structurée est principalement basée sur la stratégie de «diviser pour régner», où un problème est divisé en plusieurs petits problèmes et chaque petit problème est résolu individuellement jusqu'à ce que le problème soit résolu.

Les petits problèmes sont résolus au moyen de modules de solution. Accent sur la conception structurée pour que ces modules soient bien organisés afin de parvenir à une solution précise. Ces modules sont organisés en hiérarchie. Ils communiquent entre eux. Une bonne conception structurée suit toujours certaines règles de communication entre plusieurs modules, à savoir :

- ***Cohésion*** - regroupement de tous les éléments fonctionnellement liés.
- ***Couplage*** - communication entre différents modules.

Une bonne conception structurée présente une cohésion élevée et des arrangements de couplage.

V.2. CONCEPTION ORIENTEE FONCTION

Dans la conception orientée fonction, le système comprend de nombreux sous-systèmes plus petits appelés fonctions. Ces fonctions sont capables d'effectuer des tâches importantes dans le système. Le système est considéré comme une vue de dessus de toutes les fonctions.

La conception orientée fonction hérite de certaines propriétés de la conception structurée où la méthodologie de diviser et conquérir est utilisée. Ce mécanisme de conception divise le système entier en fonctions plus petites, ce qui fournit des moyens d'abstraction en cachant les informations et leur fonctionnement. Une autre caractéristique des fonctions est que lorsqu'un programme appelle une fonction, la fonction change l'état du programme, ce qui n'est parfois pas acceptable par les autres modules. La conception orientée fonction fonctionne bien lorsque l'état du système n'a pas d'importance et que le programme / les fonctions fonctionnent en entrée plutôt que dans un état. C'est ici la place des Procédés de conception orientée fonction :

- L'ensemble du système est vu comme la manière dont les données circulent dans le système au moyen d'un diagramme de flux de données.
- DFD décrit comment les fonctions modifient les données et l'état de l'ensemble du système.
- L'ensemble du système est logiquement divisé en unités plus petites appelées fonctions sur la base de leur fonctionnement dans le système.
- Chaque fonction est ensuite décrite en détail.

V.3. CONCEPTION ORIENTEE OBJET

La conception orientée objet fonctionne autour des entités et de leurs caractéristiques plutôt que des fonctions impliquées dans le système logiciel. Cette stratégie de conception se concentre sur les entités et leurs caractéristiques. Tout le concept de solution logicielle tourne autour des entités engagées. Voyons les concepts importants de la conception orientée objet :

- **Objets** - Toutes les entités impliquées dans la conception de la solution sont appelées objets. Par exemple, la personne, les banques, l'entreprise et les clients sont traités comme des objets. Chaque entité a des attributs qui lui sont associés et a des méthodes à appliquer sur les attributs.
- **Classes** - Une classe est une description généralisée d'un objet. Un objet est une instance d'une classe. Class définit tous les attributs qu'un objet peut avoir et les méthodes qui définissent les fonctionnalités de l'objet. Dans la conception de la solution, les attributs sont stockés au fur et à mesure que des variables et des fonctionnalités sont définies au moyen de méthodes ou de procédures.
- **Encapsulation** - Dans OOD, les attributs (variables de données) et les méthodes (opération sur les données) sont regroupés. L'encapsulation non seulement regroupe les informations importantes d'un objet, mais restreint également l'accès aux données et aux méthodes du monde extérieur. Ceci s'appelle cacher des informations.

- **Héritage** - OOD permet aux classes similaires de s'empiler de manière hiérarchique, les classes inférieures ou les sous-classes pouvant importer, implémenter et réutiliser les variables et méthodes autorisées de leurs superclasses immédiates. Cette propriété de « OOD » est connue sous le nom d'héritage. Cela facilite la définition de classes spécifiques et la création de classes généralisées à partir de classes spécifiques.
- **Polymorphisme** - Les langages OOD fournissent un mécanisme permettant d'attribuer le même nom aux méthodes exécutant des tâches similaires mais dont les arguments varient. Ceci est appelé polymorphisme, ce qui permet une interface unique effectuant des tâches pour différents types. Selon la manière dont la fonction est appelée, la partie respective du code est exécutée.

C'est ici, la place aux procédés de conception dont Le processus de conception logicielle peut être perçu comme une série d'étapes bien définies. Bien que cela varie en fonction de l'approche de conception (orientée fonction ou orientée objet, elle peut comporter les étapes suivantes :

- Une conception de solution est créée à partir d'exigences ou d'un diagramme de séquence de système et / ou de système utilisé antérieur.
- Les objets sont identifiés et regroupés en classes au nom de la similarité des caractéristiques d'attribut.
- La hiérarchie de classe et la relation entre eux sont définies.
- Le cadre d'application est défini.

V.4. APPROCHES DE CONCEPTION LOGICIELLE

Voici deux approches génériques pour la conception de logiciels :

- **La conception descendante (Top Down Design) :** Nous savons qu'un système est composé de plusieurs sous-systèmes et qu'il contient un certain nombre de composants. De plus, ces sous-systèmes et composants peuvent avoir leur ensemble de sous-systèmes et de composants et créer une structure hiérarchique dans le système.

  La conception descendante prend l'ensemble du système logiciel en tant qu'entité unique, puis le décompose pour obtenir plusieurs sous-systèmes ou composants en fonction de certaines caractéristiques. Chaque sous-système ou composant est ensuite traité comme un système et décomposé davantage. Ce processus continue de fonctionner jusqu'à ce que le niveau le plus bas du système dans la hiérarchie descendante soit atteint.

La conception descendante commence par un modèle de système généralisé et continue de définir la partie la plus spécifique. Lorsque tous les composants sont composés, tout le système existe. La conception descendante est plus appropriée lorsque la solution logicielle doit être conçue à partir de zéro et que des détails spécifiques sont inconnus.

- **La Conception ascendante (Bottom-up Design) :** Le modèle de conception ascendant commence par la plupart des composants spécifiques et de base. Il procède à la composition de niveaux de composants supérieurs en utilisant des composants de base ou de niveau inférieur. Il continue de créer des composants de niveau supérieur jusqu'à ce que le système souhaité ne soit pas évolué en un seul composant. À chaque niveau supérieur, la quantité d'abstraction est augmentée.

  La stratégie ascendante est plus appropriée lorsqu'un système doit être créé à partir d'un système existant, où les primitives de base peuvent être utilisées dans le nouveau système. Les approches descendante et ascendante ne sont pas pratiques individuellement. Au lieu de cela, une bonne combinaison des deux est utilisée.

CHAPITRE VI. CONCEPTION DE L'INTERFACE UTILISATEUR DU LOGICIEL

L'interface utilisateur est la vue de l'application physique vers laquelle l'utilisateur interagit pour utiliser le logiciel. L'utilisateur peut manipuler et contrôler le logiciel ainsi que le matériel via l'interface utilisateur. Aujourd'hui, l'interface utilisateur se trouve presque partout où la technologie numérique existe, directement depuis les ordinateurs, les téléphones mobiles, les voitures, les lecteurs de musique, les avions, les navires, etc.

L'interface utilisateur fait partie du logiciel et est conçue de manière à fournir les informations utilisateur du logiciel. L'interface utilisateur fournit une plate-forme fondamentale pour l'interaction homme-ordinateur. L'interface utilisateur peut être graphique, textuelle, audio-vidéo, en fonction de la combinaison matérielle et logicielle sous-jacente. L'interface utilisateur peut être un matériel ou un logiciel ou une combinaison des deux. Le logiciel devient plus populaire si son interface utilisateur est :

- Attrayant ;
- Simple à utiliser ;
- Réactif en peu de temps ;
- Clair pour comprendre ;
- Cohérent sur tous les écrans d'interfaçage.

L'interface utilisateur est divisée en deux catégories:

- Interface en ligne de commande ;
- Interface utilisateur graphique.

VI.1. TYPOLOGIE DE L'INTERFACE D'UTILISATEUR

VI.1.1. Interface en ligne de commande (CLI)

Le CLI a été un excellent outil d'interaction avec les ordinateurs jusqu'à la création des moniteurs d'affichage vidéo. CLI est le premier choix de nombreux utilisateurs techniques et programmeurs. L'interface CLI est l'interface minimale qu'un logiciel peut fournir à ses utilisateurs. Le CLI fournit un invité de commande, l'endroit où l'utilisateur tape la commande et alimente le système. L'utilisateur doit se souvenir de la syntaxe de commande et de son utilisation. Les anciennes CLI n'étaient pas programmées pour gérer efficacement les erreurs de l'utilisateur. Une commande est une référence textuelle à un ensemble d'instructions, qui doivent être exécutées par le système.

Il existe des méthodes telles que les macros, des scripts qui facilitent le fonctionnement de l'utilisateur. CLI utilise moins de ressources informatiques que l'interface GUI. Une interface de ligne de commande textuelle peut comporter les éléments suivants :

- **Invite de commandes -** Il s'agit d'un notificateur basé sur du texte qui affiche principalement le contexte dans lequel l'utilisateur travaille. Il est généré par le logiciel.
- **Curseur** - C'est une petite ligne horizontale ou une barre verticale de la hauteur de la ligne, pour représenter la position du personnage lors de la frappe. Le curseur se trouve principalement dans un état clignotant. Il se déplace lorsque l'utilisateur écrit ou supprime quelque chose.
- **Commande** - Une commande est une instruction exécutable. Il peut avoir un ou plusieurs paramètres. La sortie sur l'exécution de la commande est affichée en ligne à l'écran. Lorsque la sortie est produite, l'invite de commande est affichée sur la ligne suivante.

VI.1.2. Interface utilisateur graphique

L'interface utilisateur graphique fournit à l'utilisateur un moyen graphique d'interagir avec le système. L'interface graphique peut être une combinaison de matériel et de logiciel. A l'aide de l'interface graphique, l'utilisateur interprète le logiciel. Généralement, l'interface graphique consomme plus de ressources que celle de l'interface de ligne de commande. Grâce aux progrès de la technologie, les programmeurs et les concepteurs créent des conceptions d'interface graphique complexes qui fonctionnent avec plus d'efficacité, de précision et de rapidité.

L'interface graphique fournit un ensemble de composants pour interagir avec le logiciel ou le matériel. Chaque composant graphique permet de travailler avec le système. L'interface graphique est composée des éléments ci-après :

- **Fenêtre** - Une zone où le contenu de l'application est affiché. Le contenu d'une fenêtre peut être affiché sous la forme d'icônes ou de listes, si la fenêtre représente la structure du fichier. Il est plus facile pour un utilisateur de naviguer dans le système de fichiers dans une fenêtre d'exploration. Windows peut être minimisé, redimensionné ou agrandi à la taille de l'écran. Ils peuvent être déplacés n'importe où sur l'écran. Une fenêtre peut contenir une autre fenêtre de la même application, appelée fenêtre enfant.

- **Onglets -** Si une application permet d'exécuter plusieurs instances d'elle-même, elles apparaissent à l'écran sous forme de fenêtres distinctes. L'interface de document à onglets est apparue pour ouvrir plusieurs documents dans la même fenêtre. Cette interface facilite également l'affichage du panneau de préférences dans l'application. Tous les navigateurs Web modernes utilisent cette fonctionnalité.
- **Menu** - Menu est un tableau de commandes standard, regroupées et placées à un endroit visible (généralement en haut) dans la fenêtre de l'application. Le menu peut être programmé pour apparaître ou se cacher sur des clics de souris.
- **Icône** - Une icône est une petite image représentant une application associée. Lorsque vous cliquez sur ces icônes ou double-cliquez dessus, la fenêtre de l'application s'ouvre. L'icône affiche l'application et les programmes installés sur un système sous la forme de petites images.
- **Curseur** - Les périphériques interactifs tels que la souris, le pavé tactile, le stylo numérique sont représentés dans l'interface graphique comme des curseurs. Le curseur à l'écran suit les instructions du matériel presque en temps réel. Les curseurs sont également nommés pointeurs dans les systèmes GUI. Ils sont utilisés pour sélectionner des menus, des fenêtres et d'autres fonctionnalités de l'application.

Il existe également les Composants d'interface graphique spécifiques à l'application. Une interface graphique d'une application contient un ou plusieurs des éléments d'interface graphique répertoriés :

- **Fenêtre d'application** - La plupart des fenêtres d'application utilisent les constructions fournies par les systèmes d'exploitation, mais beaucoup utilisent leurs propres fenêtres créées par le client pour contenir le contenu de l'application.
- **Boîte de dialogue** - Il s'agit d'une fenêtre enfant contenant un message pour l'utilisateur et demandant que certaines actions soient entreprises. Par exemple: L'application génère un dialogue pour obtenir la confirmation de l'utilisateur de supprimer un fichier.
- **Zone de texte** - Fournit une zone permettant à l'utilisateur de taper et d'entrer des données textuelles.
- **Boutons** - Ils imitent des boutons réels et sont utilisés pour soumettre des entrées au logiciel.
- **Bouton radio** - Affiche les options disponibles pour la sélection. Un seul peut être sélectionné parmi tous les produits proposés.

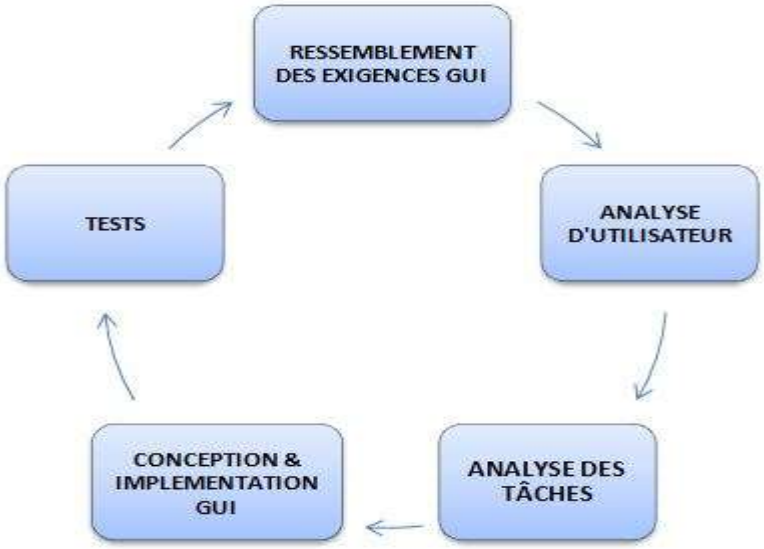
- **Case à cocher** - Fonctions similaires à la zone de liste. Lorsqu'une option est sélectionnée, la case est cochée. Plusieurs options représentées par des cases à cocher peuvent être sélectionnées.
- **Zone de liste** - Fournit la liste des éléments disponibles pour la sélection. Plusieurs articles peuvent être sélectionnés.

Les autres composants graphiques impressionnants sont :

- Sliders ;
- Boîte combo ;
- Grille de données ;
- La liste déroulante.

VI.2. ACTIVITES DE CONCEPTION D'INTERFACE UTILISATEUR

Un certain nombre d'activités sont réalisées pour concevoir une interface utilisateur. Le processus de conception et d'implémentation de l'interface graphique est identique à celui de la SDLC. N'importe quel modèle peut être utilisé pour l'implémentation de l'interface graphique en modèle Cascade, itératif ou spiral. Un modèle utilisé pour la conception et le développement d'une interface utilisateur graphique doit remplir ces étapes spécifiques à savoir :



- **Rassemblement des exigences de l'interface graphique** - Les concepteurs peuvent souhaiter avoir une liste de toutes les exigences fonctionnelles et non fonctionnelles de l'interface graphique. Cela peut être pris de l'utilisateur et de sa solution logicielle existante.

- **Analyse d'utilisateur** - Le concepteur étudie qui va utiliser l'interface graphique du logiciel. Le public cible compte à mesure que les détails de conception changent en fonction du niveau de connaissance et de compétence de l'utilisateur. Si l'utilisateur a des connaissances techniques, une interface graphique avancée et complexe peut être incorporée. Pour un utilisateur novice, plus d'informations sont fournies sur la procédure d'utilisation du logiciel.

- **Analyse des tâches** - Les concepteurs doivent analyser la tâche à accomplir par la solution logicielle. Ici, dans l'interface graphique, peu importe la façon dont cela sera fait. Les tâches peuvent être représentées de manière hiérarchique en prenant une tâche majeure et en la divisant davantage en sous-tâches plus petites. Les tâches fournissent des objectifs pour la présentation de l'interface graphique. Le flux d'informations entre les sous-tâches détermine le flux de contenu de l'interface graphique dans le logiciel.

- **Conception et implémentation de l'interface graphique** - Les concepteurs, après avoir reçu des informations sur les exigences, les tâches et l'environnement utilisateur, conçoivent l'interface graphique et implémentent le code et intègrent l'interface graphique avec des logiciels de travail ou factices en arrière-plan. Il est ensuite testé par les développeurs.

- **Tests** - Les tests d'interface graphique peuvent être réalisés de différentes manières. L'organisation peut avoir une inspection interne, l'implication directe des utilisateurs et la sortie de la version bêta sont peu nombreuses. Les tests peuvent inclure la facilité d'utilisation, la compatibilité, l'acceptation des utilisateurs, etc.

VI.3. OUTILS D'IMPLEMENTATION DE L'INTERFACE GRAPHIQUE

Plusieurs outils permettent aux concepteurs de créer une interface graphique complète en un clic de souris. Certains outils peuvent être intégrés dans l'environnement logiciel (IDE). Les outils d'implémentation d'interface graphique fournissent un puissant tableau de commandes d'interface graphique. Pour la personnalisation du logiciel, les concepteurs peuvent modifier le code en conséquence. Il existe différents segments d'outils d'interface graphique en fonction de leur utilisation et de leur plate-forme.

Exemple : Interface graphique mobile, interface utilisateur graphique, interface graphique à écran tactile, etc. Voici une liste de quelques outils utiles pour créer une interface graphique :

- FLUIDE ;
- AppInventor (Android) ;
- LucidChart ;
- Wavemaker ;
- Visual studio.

VI.4. REGLES D'OR DE L'INTERFACE UTILISATEUR

Les règles suivantes sont mentionnées comme étant les règles d'or de la conception de l'interface graphique, décrites par **Shneiderman** et **Plaisant** dans leur livre (Designing the User Interface).

- **S'efforcer d'assurer la cohérence** - Des séquences cohérentes d'actions devraient être requises dans des situations similaires. Une terminologie identique doit être utilisée dans les invites, les menus et les écrans d'aide. Des commandes cohérentes doivent être employées partout.
- **Permettre aux utilisateurs fréquents d'utiliser des raccourcis** - Le désir de l'utilisateur de réduire le nombre d'interactions augmente avec la fréquence d'utilisation. Les abréviations, les touches de fonction, les commandes masquées et les macros sont très utiles pour un utilisateur expert.
- **Offrir des commentaires informatifs** - Pour chaque action de l'opérateur, il devrait y avoir un retour d'information du système. Pour les actions fréquentes et mineures, la réponse doit être modeste, tandis que pour les actions peu fréquentes et importantes, la réponse doit être plus substantielle.
- **Dialogue de conception pour obtenir la clôture** - Les séquences d'actions doivent être organisées en groupes avec un début, un milieu et une fin. La rétroaction informative à la fin d'un groupe d'actions donne aux opérateurs la satisfaction de l'accomplissement, un sentiment de soulagement, le signal d'abandonner les plans d'urgence et les options de leur esprit, et cela indique que la voie est claire pour préparer le prochain groupe d'actions.
- **Offrez une gestion simple des erreurs** - Dans la mesure du possible, concevez le système de manière à ce que l'utilisateur ne commette pas d'erreur grave. Si une erreur est commise, le système doit pouvoir le détecter et proposer des mécanismes simples et compréhensibles pour gérer l'erreur.

- **Permettre l'inversion facile des actions** - Cette fonctionnalité soulage l'anxiété, car l'utilisateur sait que les erreurs peuvent être annulées. Une inversion facile des actions encourage l'exploration d'options inconnues. Les unités de réversibilité peuvent être une action unique, une entrée de données ou un groupe complet d'actions.

- **Support du locus de contrôle interne** - Les opérateurs expérimentés souhaitent vivement avoir le sentiment qu'ils sont responsables du système et que le système répond à leurs actions. Concevoir le système pour que les utilisateurs soient les initiateurs des actions plutôt que les intervenants.

- **Réduire la charge de mémoire à court terme** - La limitation du traitement des informations humaines dans la mémoire à court terme nécessite de conserver des affichages simples, de consolider les affichages de plusieurs pages, de réduire la fréquence des mouvements de fenêtre et séquences d'actions.

CHAPITRE VII. COMPLEXITE DU DESIGN LOGICIEL

Le terme complexité désigne l'état des événements ou des choses, qui ont plusieurs liens interconnectés et des structures très complexes. Dans la programmation logicielle, à mesure que la conception du logiciel est réalisée, le nombre d'éléments et leurs interconnexions deviennent progressivement énormes, ce qui devient trop difficile à comprendre en même temps. La complexité de la conception logicielle est difficile à évaluer sans utiliser des mesures de complexité. Voyons trois mesures importantes de complexité logicielle.

VII.1. MESURES DE COMPLEXITE DE HALSTEAD

En 1977, M. Maurice Howard Halstead a introduit des mesures pour mesurer la complexité des logiciels. Les métriques de Halstead dépendent de l'implémentation réelle du programme et de ses mesures, qui sont calculées directement à partir des **opérateurs** (*élément effectuant une opération ou une instruction*) et **des opérandes** (*élément sur laquelle porte opération ou instruction d'un langage de programmation*) à partir du code source, de manière statique. **Il permet d'évaluer le temps de test, le vocabulaire, la taille, la difficulté, les erreurs et les efforts pour le code source C / C ++ / Java.**

Selon Halstead, « *un programme informatique est une implémentation d'un algorithme considéré comme un ensemble de jetons pouvant être classés en tant qu'opérateurs ou opérandes* ». Les métriques Halstead considèrent un programme comme une séquence d'opérateurs et leurs opérandes associés. Il définit divers indicateurs pour vérifier la complexité du module.

VII.2. MESURE DE COMPLEXITE CYCLOMATIQUE

En 1976, McCabe, a proposé la mesure de la complexité cyclomatique pour quantifier la complexité d'un logiciel donné. Il s'agit d'un modèle basé sur un graphe qui est basé sur des constructions décisionnelles de programmes telles que les instructions **if-else**, **do-while**, **repeat-until**, ***switch-case et goto*** … Chaque programme comprend des instructions à exécuter pour exécuter certaines tâches et d'autres instructions décisionnelles qui décident des instructions à exécuter. Ces constructions décisionnelles modifient le déroulement du programme. Si nous comparons deux programmes de même taille, celui qui contient le plus de déclarations sera plus complexe, car le contrôle du programme saute fréquemment. Voici les Processus pour créer un graphique de contrôle de flux cyclomatique :

- Casser le programme en blocs plus petits, délimités par des constructions décisionnelles ;
- Créez des nœuds représentant chacun de ces nœuds ;
- Connectez les nœuds comme suit : Si le contrôle peut se brancher du bloc i au bloc j ; et du nœud de sortie au nœud d'entrée.

Pour calculer la complexité cyclomatique d'un module de programme, nous utilisons la formule suivante :

$$V(G) = e - n + 2$$ Où

e est le nombre total d'arêtes ***n*** est le nombre total de nœuds

Selon P. Jorgensen, la complexité cyclomatique d'un module ne devrait pas dépasser 10. C'est par exemple, La complexité cyclomatique du module ci-dessus est :

$e = 10$ $n = 8$

Complexité Cyclomatique = $10 - 8 + 2 = 4$

VII.3. MESURE DE POINT DE FONCTION

Il est largement utilisé pour mesurer la taille du logiciel. Le point de fonction se concentre sur les fonctionnalités fournies par le système. Les fonctionnalités et fonctionnalités du système sont utilisées pour mesurer la complexité du logiciel. Le point de fonction compte sur cinq paramètres, nommés **entrée externe**, **sortie externe**, **fichiers internes logiques**, **fichiers d'interface externe** et **interrogation externe**. Pour prendre en compte la complexité du logiciel, chaque paramètre est classé comme simple, moyen ou complexe. Voyons les paramètres du point de fonction :

- **Entrée externe** : Chaque entrée unique dans le système, de l'extérieur, est considérée comme une entrée externe. L'unicité de l'entrée est mesurée, car deux entrées ne doivent pas avoir le même format. Ces entrées peuvent être des paramètres de données ou de contrôle :

  → **Simple** - si le nombre d'entrées est faible et affecte moins de fichiers internes
  → **Complexe** - si le nombre d'entrées est élevé et affecte plus de fichiers internes
  → **Moyenne** - entre simple et complexe.

- **Sortie externe :** Tous les types de sortie fournis par le système sont comptabilisés dans cette catégorie. La sortie est considérée comme unique si leur format de sortie et / ou leur traitement sont uniques :
  → Simple - si le nombre de sorties est faible ;
  → Complexe - si le nombre de sorties est élevé ;
  → Moyen - entre simple et complexe.

- **Fichiers internes logiques :** Chaque logiciel maintient des fichiers internes afin de conserver ses informations fonctionnelles et de fonctionner correctement. Ces fichiers contiennent des données logiques du système. Ces données logiques peuvent contenir à la fois des données fonctionnelles et des données de contrôle :

  → **Simple** - si le nombre de types d'enregistrement est faible ;
  → **Complexe** - si le nombre de types d'enregistrement est élevé ;
  → **Moyen** - entre simple et complexe.

- **Fichiers d'interface externes :** Le système logiciel peut avoir besoin de partager ses fichiers avec certains logiciels externes ou il peut avoir besoin de transmettre le fichier pour traitement ou comme paramètre à une fonction. Tous ces fichiers sont comptabilisés en tant que fichiers d'interface externes.

  → Simple - si le nombre de types d'enregistrements dans un fichier partagé est faible ;
  → Complexe - si le nombre de types d'enregistrement dans le fichier partagé est élevé ;
  → Moyen - entre simple et complexe.

- **Enquête externe** : Une interrogation est une combinaison d'entrées et de sorties, où l'utilisateur envoie des données pour se renseigner en tant qu'entrées et le système répond à l'utilisateur avec le résultat de la requête traité. La complexité d'une requête est supérieure à l'entrée externe et à la sortie externe. ***Query*** est dit unique si ses entrées et sorties sont uniques en termes de format et de données.

  → Simple - si la requête nécessite peu de traitement et génère peu de données de sortie ;
  → Complexe - si la requête nécessite un processus élevé et génère une grande quantité de données de sortie ;
  → Moyen - entre simple et complexe.

Chacun de ces paramètres dans le système est pondéré en fonction de sa classe et de sa complexité. Le tableau ci-dessous mentionne la pondération donnée à chaque paramètre :

Paramètres	Simple	Moyen	Complexe
Entrées	3	4	6
Sorties	4	5	7
Enquêtes	3	4	6
Fichiers	7	10	15
Interfaces	5	7	10

Le tableau ci-dessus fournit des points de fonction bruts. Ces points de fonction sont ajustés en fonction de la complexité de l'environnement. Le système est décrit en utilisant quatorze caractéristiques différentes :

- Données de communication ;
- Traitement distribué ;
- Objectifs de performance ;
- Charge de configuration d'opération ;
- Taux de transaction ;
- Saisie de données en ligne ;
- Efficacité de l'utilisateur final ;
- Mise à jour en ligne ;
- Logique de traitement complexe ;
- Réutilisation ;
- Facilité d'installation ;
- Facilité d'exploitation ;
- Plusieurs sites ;
- Désir de faciliter les changements.

Ces facteurs caractéristiques sont alors notés de 0 à 5, comme indiqué ci-dessous:

- Aucune influence ;
- Accessoire ;
- Modérer ;
- Moyenne ;
- Important ;
- Essentiel.

Toutes les notes sont ensuite résumées en tant que N. La valeur de N va de 0 à 70 (14 types de caractéristiques x 5 types de notation). Il est utilisé pour calculer les facteurs d'ajustement de la complexité (CAF) en utilisant les formules suivantes :

$$CAF = 0,65 + 0,01N$$

Alors,

Points de fonction fournis (PC) = CAF x FP brut

Ce FP peut ensuite être utilisé dans diverses mesures, telles que:

Coût = $ / FP

Qualité = Erreurs / FP

Productivité = PF / mois-personne.

CHAPITRE VIII. MISE EN ŒUVRE LOGICIELLE

Dans ce chapitre, nous étudierons les méthodes de programmation, la documentation et les défis de l'implémentation logicielle.

VIII.1. PROGRAMMATION STRUCTUREE

Dans le processus de codage, les lignes de code continuent à se multiplier, la taille du logiciel augmente donc. Progressivement, il devient presque impossible de se souvenir du déroulement du programme. Si l'on oublie comment les logiciels et les programmes, fichiers et procédures sous-jacents sont construits, il devient alors très difficile de partager, déboguer et modifier le programme. La solution à cela est la programmation structurée. Il encourage le développeur à utiliser des sous-programmes et des boucles au lieu d'utiliser de simples sauts dans le code, ce qui apporte de la clarté dans le code et améliore son efficacité. La programmation structurée aide également le programmeur à réduire le temps de codage et à La programmation structurée indique comment le programme doit être codé. La programmation structurée utilise trois concepts principaux :

- **Analyse descendante** - Un logiciel est toujours conçu pour effectuer un travail rationnel. Ce travail rationnel est connu comme un problème dans le jargon logiciel. Il est donc très important que nous comprenions comment résoudre le problème. Dans l'analyse descendante, le problème est divisé en petites parties, chacune d'elles ayant une signification. Chaque problème est résolu individuellement et des étapes sont clairement énoncées sur la façon de résoudre le problème.
- **Programmation modulaire** - Pendant la programmation, le code est divisé en un groupe d'instructions plus petit. Ces groupes sont appelés modules, sous-programmes ou sous-programmes. Programmation modulaire basée sur la compréhension de l'analyse descendante. Il décourage les sauts en utilisant les instructions « *goto* » dans le programme, ce qui rend souvent le programme non traçable. Les sauts sont interdits et le format modulaire est encouragé dans la programmation structurée.
- **Codage structuré-** En référence à une analyse descendante, le codage structuré subdivise les modules en d'autres unités de code plus petites dans l'ordre de leur exécution. La programmation structurée utilise la structure de contrôle, qui contrôle le flux du programme, tandis que le codage structuré utilise la structure de contrôle pour organiser ses instructions dans des modèles définissables.

VIII.2. PROGRAMMATION FONCTIONNELLE

La programmation fonctionnelle est un style de langage de programmation, qui utilise les concepts de fonctions mathématiques. Une fonction en mathématiques devrait toujours produire le même résultat à la réception du même argument. Dans les langages procéduraux, le déroulement du programme passe par des procédures, c'est-à-dire que le contrôle du programme est transféré à la procédure appelée. Pendant que le flux de contrôle passe d'une procédure à une autre, le programme change d'état. Dans la programmation procédurale, il est possible qu'une procédure produise des résultats différents lorsqu'elle est appelée avec le même argument, car le programme lui-même peut être dans un état différent lors de l'appel. C'est une propriété ainsi qu'un inconvénient de la programmation procédurale, dans laquelle la séquence ou le moment d'exécution de la procédure devient important. La programmation fonctionnelle fournit des moyens de calcul sous forme de fonctions mathématiques qui produisent des résultats indépendamment de l'état du programme. Cela permet de prédire le comportement du programme. La programmation fonctionnelle utilise les concepts suivants :

- **Fonctions de première classe et d'ordre élevé** - Ces fonctions peuvent accepter une autre fonction en tant qu'argument ou renvoyer d'autres fonctions en tant que résultats.
- **Fonctions pures** - Ces fonctions n'incluent pas les mises à jour destructives, c'est-à-dire qu'elles n'affectent pas les E / S ni la mémoire. Si elles ne sont pas utilisées, elles peuvent facilement être supprimées sans entraver le reste du programme.
- **Récursion** - La récursivité est une technique de programmation dans laquelle une fonction s'appelle et répète le code du programme, sauf si une condition prédéfinie correspond. La récursivité est la manière de créer des boucles dans la programmation fonctionnelle.
- **Évaluation stricte** - C'est une méthode d'évaluation de l'expression transmise à une fonction sous forme d'argument. La programmation fonctionnelle a deux types de méthodes d'évaluation, stricte (désireux) et non stricte (lazy). Une évaluation stricte évalue toujours l'expression avant d'appeler la fonction. Une évaluation non stricte n'évalue pas l'expression sauf si elle est nécessaire.
- **λ-calcul -** La plupart des langages de programmation fonctionnels utilisent le λ-calcul comme système de types. Les expressions λ sont exécutées en les évaluant lorsqu'elles se produisent.

N.B. Common Lisp, Scala, Haskell, Erlang et F # sont quelques exemples de langages de programmation fonctionnels.

VIII.3. STYLE DE PROGRAMMATION

Le style de programmation est un ensemble de règles de codage suivies par tous les programmeurs pour écrire le code. Lorsque plusieurs programmeurs travaillent sur le même projet logiciel, ils doivent souvent travailler avec le code de programme écrit par un autre développeur. Cela devient fastidieux ou parfois impossible si tous les développeurs ne suivent pas un style de programmation standard pour coder le programme. Un style de programmation approprié consiste à utiliser des noms de fonction et de variable pertinents pour la tâche envisagée, à utiliser une indentation bien placée, un code de commentaire pour la commodité du lecteur et une présentation générale du code. Cela rend le code de programme lisible et compréhensible par tous, ce qui facilite le débogage et la résolution des erreurs. En outre, un style de codage approprié facilite la documentation et la mise à jour.

La pratique du style de codage varie selon les organisations, les systèmes d'exploitation et la langue de codage. Les éléments de codage suivants peuvent être définis dans les directives de codage d'une organisation :

- **Conventions de dénomination** - Cette section explique comment nommer des fonctions, des variables, des constantes et des variables globales.
- **Mise en retrait** - Il s'agit de l'espace disponible au début de la ligne, généralement entre 2 et 8 espaces ou un seul onglet.
- **Espace blanc** - Il est généralement omis à la fin de la ligne.
- **Opérateurs** - Définit les règles d'écriture d'opérateurs mathématiques, d'affectation et logiques. Par exemple, l'opérateur d'affectation «=» devrait avoir un espace avant et après, comme dans «x = 2».
- **Structures de contrôle** - Les règles d'écriture des instructions « *if-then-else, case-switch, while-until* » et pour les instructions de flux de contrôle uniquement et de manière imbriquée.
- **Longueur de ligne et retour à la ligne** - Définit le nombre de caractères devant être présents sur une ligne, une ligne comportant généralement 80 caractères. L'emballage définit la manière dont une ligne doit être enveloppée, si elle est trop longue.
- **Fonctions** - Ceci définit comment les fonctions doivent être déclarées et appelées, avec et sans paramètres.
- **Variables** - Cela indique comment les variables de différents types de données sont déclarées et définies.
- **Commentaires** - Ceci est l'un des composants de codage importants, car les commentaires inclus dans le code décrivent ce que le code fait réellement et toutes les autres descriptions associées. Cette section permet également de créer des documentations d'aide pour d'autres développeurs.

VIII.4. DOCUMENTATION DU LOGICIEL

La documentation logicielle est une partie importante du processus logiciel. Un document bien écrit fournit un excellent outil et un moyen de stockage des informations nécessaire pour connaître le processus logiciel. La documentation du logiciel fournit également des informations sur l'utilisation du produit. Une documentation bien entretenue doit comprendre les documents suivants :

VIII.4.1. DOCUMENTATION DES BESOINS

Cette documentation fonctionne comme un outil clé pour le concepteur de logiciels, le développeur et l'équipe de test pour effectuer leurs tâches respectives. Ce document contient toutes les descriptions fonctionnelles, non fonctionnelles et comportementales du logiciel prévu. La source de ce document peut être des données préalablement stockées sur le logiciel, des logiciels déjà en cours d'exécution chez le client, des entretiens avec le client, des questionnaires et des recherches.

Généralement, il est stocké sous forme de feuille de calcul ou de traitement de texte avec l'équipe de gestion logicielle haut de gamme. Cette documentation sert de base au logiciel à développer et est principalement utilisée dans les phases de vérification et de validation. La plupart des cas de test sont construits directement à partir de la documentation des exigences.

VIII.4.2. DOCUMENTATION DE CONCEPTION DE LOGICIEL

Ces documentations contiennent toutes les informations nécessaires à la création du logiciel. Il contient :

→ une architecture logicielle de haut niveau ;
→ des détails de conception du logiciel ;
→ des diagrammes de flux de données ;
→ la conception de la base de données.

Ces documents fonctionnent comme un référentiel pour les développeurs pour implémenter le logiciel. Bien que ces documents ne donnent aucun détail sur la façon de coder le programme, ils fournissent toutes les informations nécessaires pour le codage et la mise en œuvre.

VIII.4.3. DOCUMENTATION TECHNIQUE

Ces documentations sont maintenues par les développeurs et les codeurs actuels. Ces documents, dans leur ensemble, représentent des informations sur le code. Lors de la rédaction du code, les programmeurs mentionnent également l'objectif du code, qui l'a écrit, où sera-t-il requis, que fait-il et comment, quelles autres ressources le code utilise-il, etc.

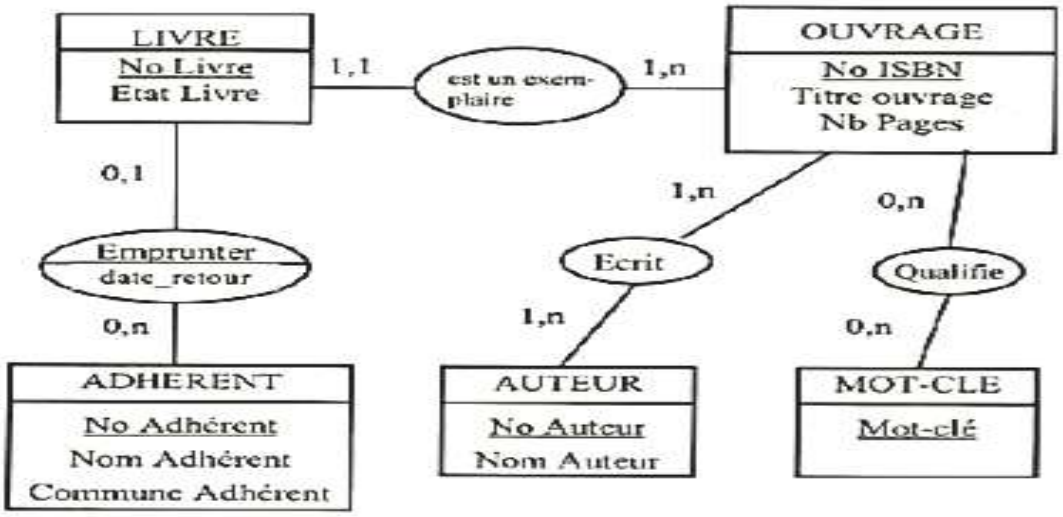
La documentation technique améliore la compréhension entre les différents programmeurs travaillant sur le même code. Il améliore la capacité de réutilisation du code. Cela rend le débogage facile et traçable. Il existe divers outils automatisés disponibles et certains sont fournis avec le langage de programmation lui-même. Par exemple, Java vient avec JavaDoc, un outil permettant de générer une documentation technique sur le code.

Le mot-clé de la documentation technique est « *comment* ». Il ne s'agit pas ici de dire pourquoi le logiciel existe, ni de décrire ses fonctionnalités attendues. Ces informations figurent dans d'autres documents comme le cahier des charges. Il ne s'agit pas non plus d'expliquer à un utilisateur du logiciel ce qu'il doit faire pour effectuer telle ou telle tâche : c'est le rôle de la documentation utilisateur. La documentation technique doit expliquer ***comment*** fonctionne le logiciel.

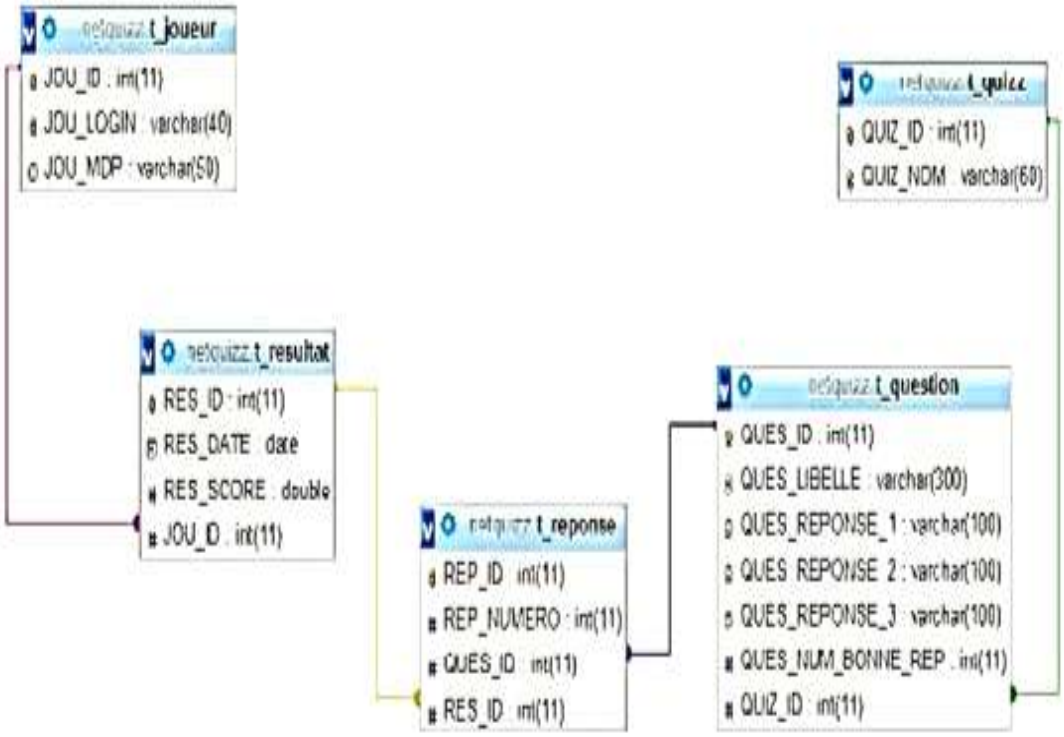
La documentation technique est écrite par des informaticiens, pour des informaticiens. Elle nécessite des compétences techniques pour être comprise. Le public visé est celui des personnes qui interviennent sur le logiciel du point de vue technique : « *développeurs, intégrateurs, responsables techniques, éventuellement chefs de projet* ». Dans le cadre d'une relation maîtrise d'ouvrage ou maîtrise d'œuvre pour réaliser un logiciel, la responsabilité de la documentation technique est à la charge de la maîtrise d'œuvre. Le contenu de la documentation technique varie fortement en fonction de la structure et de la complexité du logiciel associé. Néanmoins, nous allons décrire les aspects qu'on y retrouve le plus souvent :

1. **La Modélisation :** La documentation technique inclut les informations liées au domaine du logiciel. Elle précise comment les éléments - métiers ont été modélisés informatiquement au sein du logiciel.

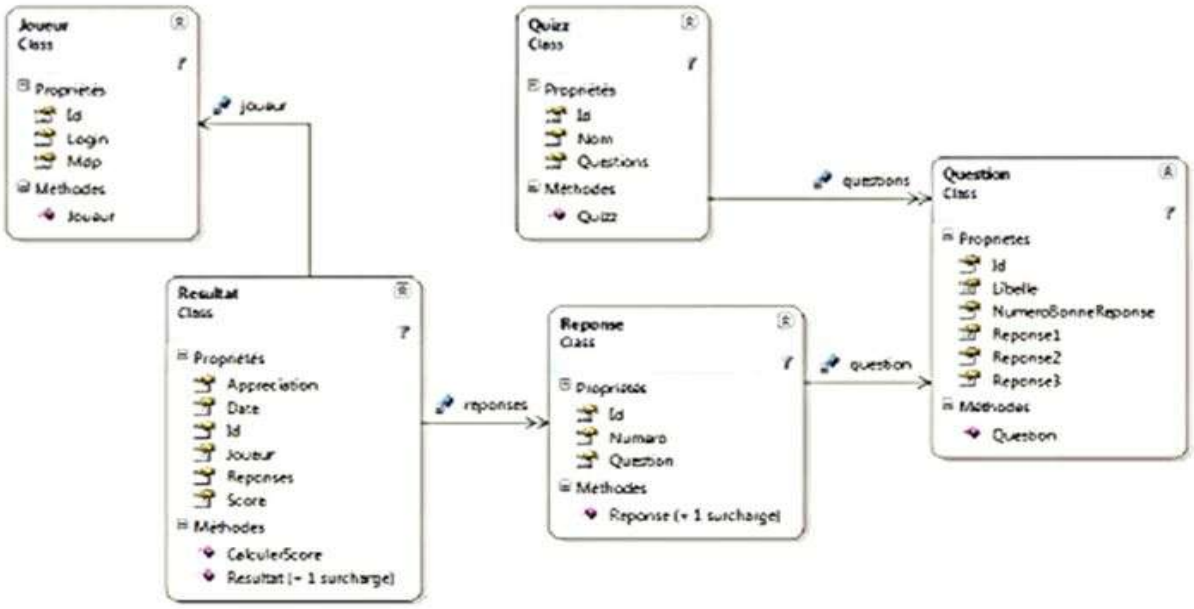
    - Si le logiciel a fait l'objet d'une modélisation de type entité-association, la documentation technique présente le modèle conceptuel résultat.



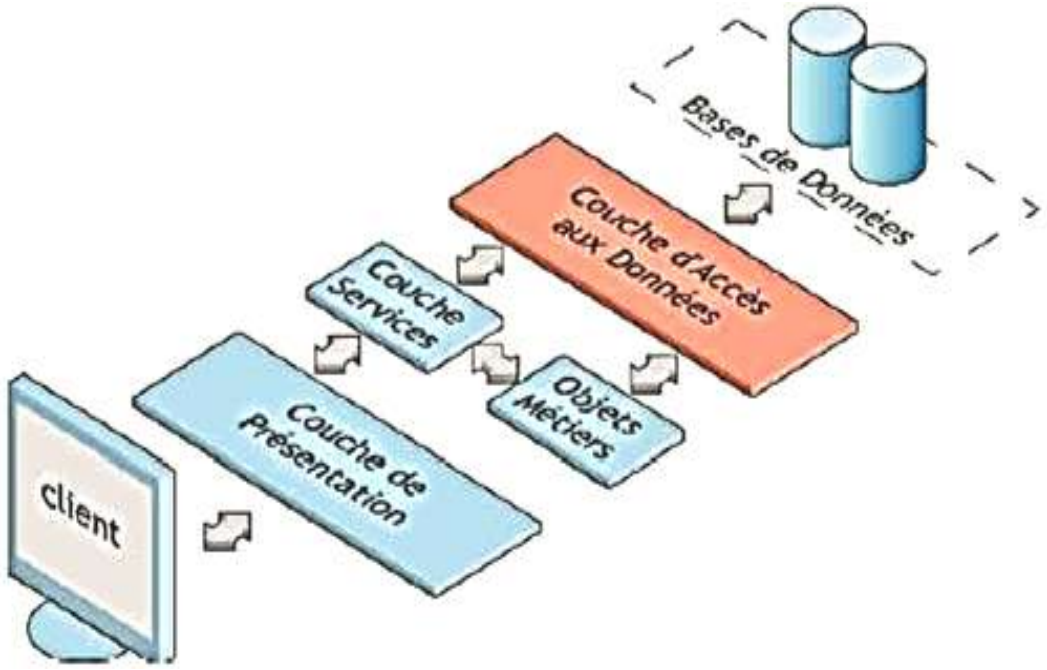
- Si le logiciel a fait l'objet d'une modélisation orientée objet, la documentation technique inclut une représentation des principales classes (*souvent les classes métier*) sous la forme d'un diagramme de classes respectant la norme UML.



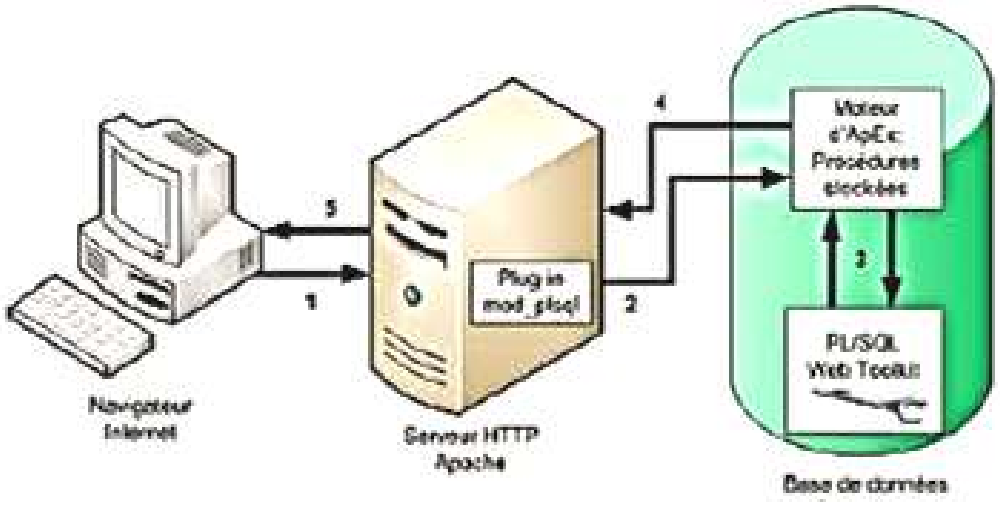
- Si le logiciel utilise une base de données, la documentation technique doit présenter le modèle d'organisation des données retenu, le plus souvent sous la forme d'un modèle logique sous forme graphique



2. **L'Architecture :** La phase d'architecture d'un logiciel permet, en partant des besoins exprimés dans le cahier des charges, de réaliser les grands choix qui structureront le développement : technologies, langages, patrons utilisés, découpage en sous-parties, outils, etc. La documentation technique doit décrire tous ces choix de conception. L'ajout de schémas est conseillé, par exemple pour illustrer une organisation logique multicouche.



L'implantation physique des différents composants (*appelés parfois tiers*) sur une ou plusieurs machines doit également être documentée.



3. **La Production du code source :** Afin d'augmenter la qualité du code source produit, de nombreux logiciels adoptent des normes ou des standards de production du code source : conventions de nommage, formatage du code, etc. Certains peuvent être internes et spécifiques au logiciel, d'autres sont reprises de normes existantes (*exemples : normes PSR-x pour PHP*).Afin que les nouveaux développeurs les connaissent et les respectent, ces normes et standards doivent être présentés dans la documentation technique.

4. **La Génération :** Le processus de génération (« ***build*** ») permet de passer des fichiers sources du logiciel aux éléments exécutables. Elle inclut souvent une phase de compilation du code source. Sa complexité est liée à celle du logiciel. Dans les cas simples, toute la génération est effectuée de manière transparente par l'IDE utilisé. Dans d'autres, elle se fait en plusieurs étapes et doit donc être documentée.

5. **Le Déploiement :** La documentation technique doit indiquer comment s'effectue le déploiement du logiciel, c'est-à-dire l'installation de ses différents composants sur la ou les machines nécessaires. Là encore, cette étape peut être plus ou moins complexe et nécessiter plus ou moins de documentation.

6. **La Documentation du code source :** Il est également possible de documenter un logiciel directement depuis son code source en y ajoutant des commentaires. Certains langages disposent d'un format spécial de commentaire permettant de créer une documentation auto-générée. C'est par exemple, *Le langage Java* qui a été le premier langage à introduire une technique de documentation du code source basée sur l'ajout de commentaires utilisant un format spécial. En exécutant un outil du JDK appelé « ***javadoc*** », on obtient une documentation du code source au format HTML.

```
/**
* Valide un mouvement de jeu d'Echecs. *
* @paramleDepuisFile File de la pièce à déplacer
* @paramleDepuisRangée Rangée de la pièce à déplacer
* @paramleVersFile File de la case de destination
* @paramleVersRangée Rangée de la case de destination
* @return vrai si le mouvement d'échec est valide ou faux si invalide */
booleanestUnDeplacementValide(intleDepuisFile, intleDepuisRangée, intleVersFile, i
ntleVersRangée) {
// ...
}
```

Voici un exemple de la méthode Java documentée au format javadoc.

L'avantage de cette approche est qu'elle facilite la documentation du code par les développeurs, au fur et à mesure de son écriture. Depuis, de nombreux langages ont repris l'idée.

```
/// <summary>
/// Classe modélisant un adversaire de Chifoumi
/// </summary>
public class Adversaire {
/// <summary>
/// Générateur de nombres aléatoires
/// Utilisé pour simuler le choix d'un signe : pierre, feuille ou ciseau
/// </summary>
private static Random rng = new Random();
/// <summary>
/// Fait choisir aléatoirement un coup à l'adversaire
/// Les coups possibles sont : "pierre", "feuille", "ciseau"
/// </summary>
/// <returns>Le coup choisi</returns>
public string ChoisirCoup()
{
// ...
}}
```

Voici un exemple de documentation d'une classe C#, qui utilise une syntaxe légèrement différente.

VIII.4.4. DOCUMENTATION UTILISATEUR

Cette documentation est différente de toutes les explications ci-dessus. Toutes les documentations précédentes sont conservées pour fournir des informations sur le logiciel et son processus de développement. Mais la documentation utilisateur explique comment le logiciel doit fonctionner et comment il doit être utilisé pour obtenir les résultats souhaités. Ces documentations peuvent inclure des procédures d'installation de logiciel, des guides de procédures, des guides de l'utilisateur, une méthode de désinstallation et des références spéciales pour obtenir plus d'informations telles que la mise à jour de licence, etc.

Contrairement à la documentation technique, la documentation d'utilisation ne vise pas à faire comprendre comment le logiciel est conçu. Son objectif est d'apprendre à l'utilisateur à se servir du logiciel. La documentation d'utilisation doit être :

- ***Utile*** : une information exacte, mais inutile, ne fait que renforcer le sentiment d'inutilité et gêne la recherche de l'information pertinente ;
- ***Agréable*** : sa forme doit favoriser la clarté et mettre en avant les préoccupations de l'utilisateur et non pas les caractéristiques techniques du produit.

Le public visé est l'ensemble des utilisateurs du logiciel. Selon le contexte d'utilisation, les utilisateurs du logiciel à documenter peuvent avoir des connaissances en informatique (*exemples : cas d'un IDE ou d'un outil de SCM*). Cependant, on supposera le plus souvent que le public visé n'est pas un public d'informaticiens. La Conséquence essentielle est que toute information trop technique est à bannir de la documentation d'utilisation. Pas question d'aborder, l'architecture MVC ou les design patterns employés : ces éléments ont leur place dans la documentation technique. D'une manière générale, s'adapter aux connaissances du public visé constitue la principale difficulté de la rédaction de la documentation d'utilisation.

LES FORMES DE LA DOCUMENTATION UTILISATEUR

1. **Le Manuel utilisateur :** La forme la plus classique de la documentation d'utilisation consiste à rédiger un manuel utilisateur, le plus souvent sous la forme d'un document bureautique. Ce document est structuré et permet aux utilisateurs de retrouver les informations qu'ils recherchent. Il intègre très souvent des captures d'écran afin d'illustrer le propos. Un manuel utilisateur peut être organisé de deux façons :

- **Le Guide d'utilisation** : ce mode d'organisation décompose la documentation en grandes fonctionnalités décrites pas à pas et dans l'ordre de leur utilisation. Exemple pour un logiciel de finances personnelles : création d'un compte, ajout d'écritures, pointage...Cette organisation plaît souvent aux utilisateurs car elle leur permet d'accéder facilement aux informations essentielles. En revanche, s'informer sur une fonctionnalité avancée ou un détail complexe peut s'avérer difficile.

- **Manuel de référence** : dans ce mode d'organisation, on décrit une par une chaque fonctionnalité du logiciel, sans se préoccuper de leur ordre ou de leur fréquence d'utilisation. Par exemple, on décrit l'un après l'autre chacun des boutons d'une barre de boutons, alors que certains sont plus « *importants* » que d'autres. Cette organisation suit la logique du créateur du logiciel plutôt que celle de son utilisateur. Elle est en général moins appréciée de ces derniers.

2. **Le Tutoriel :** De plus en plus souvent, la documentation d'utilisation inclut un ou plusieurs tutoriels, destinés à faciliter la prise en main initiale du logiciel. Un tutoriel est un guide pédagogique constitué d'instructions détaillées pas à pas en vue d'objectifs simples. Le tutoriel a l'avantage de "prendre l'utilisateur par la main" afin de l'aider à réaliser ses premiers pas avec le logiciel qu'il découvre, sans l'obliger à parcourir un manuel utilisateur plus ou moins volumineux. Il peut prendre la forme d'un document texte, ou bien d'une vidéo ou d'un exercice interactif. Cependant, il est illusoire de vouloir documenter l'intégralité d'un logiciel en accumulant les tutoriels.

3. ***Le FAQ* :** Une Foire Aux Questions (en anglais *Frequently Asked questions*) est une liste de questions-réponses sur un sujet. Elle peut faire partie de la documentation d'utilisation d'un logiciel. La création d'une FAQ permet d'éviter que les mêmes questions soient régulièrement posées.

4. **L'Aide en ligne :** L'aide en ligne est une forme de documentation d'utilisation accessible depuis un ordinateur. Il peut s'agir d'une partie de la documentation publiée sur Internet sous un format hypertexte. Quand une section de l'aide en ligne est accessible facilement depuis la fonctionnalité d'un logiciel qu'elle concerne, elle est appelée aide contextuelle ou aide en ligne contextuelle.

Les principaux formats d'aide en ligne sont le *HTML* et le *PDF*. Microsoft en a publié plusieurs formats pour l'aide en ligne des logiciels tournant sous Windows : *HLP*, *CHM* ou encore *MAML*. Un moyen simple et efficace de fournir une aide en ligne consiste à définir des infos bulle (*tooltips*). Elles permettent de décrire succinctement une fonctionnalité par survol du curseur

VIII.5. DEFIS D'IMPLEMENTATION LOGICIELLE

L'équipe de développement doit faire face à certains défis lors de la mise en œuvre du logiciel. Certains d'entre eux sont mentionnés ci-dessous :

- **Réutilisation de code** - Les interfaces de programmation des langages actuels sont très sophistiquées et disposent de vastes fonctions de bibliothèque. Néanmoins, pour réduire les coûts du produit final, la direction de l'entreprise préfère réutiliser le code, créé précédemment pour un autre logiciel. Les programmeurs rencontrent d'énormes problèmes de vérification de la compatibilité et de détermination de la quantité de code à réutiliser.

- **Gestion des versions** - Chaque fois qu'un nouveau logiciel est fourni au client, les développeurs doivent conserver une documentation relative à la version et à la configuration. Cette documentation doit être extrêmement précise et disponible à temps.

- **Cible-hôte** - le programme logiciel en cours de développement dans l'entreprise doit être conçu pour les ordinateurs hôtes du côté des clients. Mais parfois, il est impossible de concevoir un logiciel qui fonctionne sur les machines cibles.

CHAPITRE IX. VUE D'ENSEMBLE DES TESTS LOGICIELS

Le test de logiciel est une évaluation du logiciel par rapport aux exigences recueillies auprès des utilisateurs et des spécifications du système. Les tests sont effectués au niveau de la phase du cycle de vie du développement logiciel ou au niveau du module dans le code du programme. Les tests du logiciel ont pour cibles, les éléments tels que :

- **Erreurs** - Ce sont des erreurs de codage réelles commises par les développeurs. En outre, il existe une différence entre la sortie du logiciel et celle désirée, considérée comme une erreur.
- **Défauts** - En cas d'erreur, l'erreur se produit. Un défaut, également appelé « *bogue* », résulte d'une erreur pouvant entraîner l'échec du système.
- **Échec** - On dit que l'échec est l'incapacité du système à exécuter la tâche souhaitée. Une défaillance se produit lorsque la panne existe dans le système.

IX.1. LES COMPOSANTS DU TEST LOGICIEL

Les tests logiciels comprennent *la validation* et *la vérification.*

IX.1.1. LA VALIDATION DU LOGICIEL

La validation consiste à examiner si le logiciel répond ou non aux besoins de l'utilisateur. Il est effectué à la fin du SDLC. Si le logiciel correspond aux exigences pour lesquelles il a été créé, et il est validé telles que :

- La validation garantit que le produit en développement est conforme aux exigences de l'utilisateur.
- La validation répond à la question – « D*éveloppons-nous le produit qui tente de répondre aux besoins de ce logiciel?* ».
- La validation met l'accent sur les besoins des utilisateurs.

IX.1.2. LA VERIFICATION DU LOGICIEL

La vérification consiste à confirmer si le logiciel répond aux exigences de l'entreprise et s'il est développé conformément aux spécifications et méthodologies appropriées :

- La vérification garantit que le produit en cours de développement est conforme aux spécifications de conception.
- La vérification répond à la question – « *Développons-nous ce produit en respectant toutes les spécifications de conception?* » ;
- Les vérifications se concentrent sur la conception et les spécifications du système.

IX.2. TEST MANUEL OU AUTOMATISE

Les tests peuvent être effectués manuellement ou à l'aide d'un outil de test automatisé :

- **Manuel** - Ce test est effectué sans l'aide d'outils de test automatisés. Le testeur de logiciel prépare les scénarios de test pour différentes sections et niveaux du code, exécute les tests et communique le résultat au gestionnaire. Les tests manuels demandent beaucoup de temps et de ressources. Le testeur doit confirmer si les bons cas de test sont utilisés ou non. La majeure partie des tests implique des tests manuels.
- **Automatisé**- Ce test est une procédure de test effectuée à l'aide d'outils de test automatisés. Les limites des tests manuels peuvent être surmontées à l'aide d'outils de test automatisés.

Un test doit vérifier si une page Web peut être ouverte dans Internet Explorer. Cela peut être facilement fait avec des tests manuels. Mais pour vérifier si le serveur Web peut supporter la charge d'un million d'utilisateurs, il est impossible de tester manuellement. Il existe des outils logiciels et matériels qui aident le testeur à effectuer *des tests de charge*, *des tests de contrainte* et *des tests de régression.*

IX.3. APPROCHES DES TESTS LOGICIELS

Les tests peuvent être effectués sur la base de deux approches :

- Test de fonctionnalité ;
- Test d'implémentation.

AINSI, Lorsque la fonctionnalité est testée sans prendre en compte la mise en œuvre réelle, on parle de test de boîte noire. L'autre côté est connu sous le nom de test en boîte blanche, dans lequel non seulement la fonctionnalité est testée, mais également la manière dont elle est mise en œuvre est analysée. Les tests exhaustifs sont la meilleure méthode pour un test parfait. Chaque valeur possible dans la plage des valeurs d'entrée et de sortie est testée. Il n'est pas possible de tester chaque valeur dans un scénario du monde réel si la plage de valeurs est large.

IX.3.1. TEST DE LA BOITE NOIRE

Il est effectué pour tester la fonctionnalité du programme. Il s'appelle aussi test « *comportemental* ». Le testeur, dans ce cas, a un ensemble de valeurs d'entrée et les résultats souhaités respectifs. Lors de la saisie, si la sortie correspond aux résultats souhaités, le programme est testé « *ok* » et pose problème, sinon. Dans cette méthode de test, le testeur ignore la conception et la structure du code, et les ingénieurs de test et les utilisateurs finaux effectuent ce test sur le logiciel. Les Techniques de test de la boite noire sont :

- **Classe d'équivalence** - L'entrée est divisée en classes similaires. Si un élément d'une classe réussit le test, on suppose que toute la classe est réussie.
- **Valeurs limites** - L'entrée est divisée en valeurs extrêmes supérieure et inférieure. Si ces valeurs réussissent le test, on suppose que toutes les valeurs intermédiaires peuvent également réussir.
- **Représentation graphique des causes-effets** - Dans les deux méthodes précédentes, une seule valeur d'entrée à la fois est testée. Cause (entrée) - L'effet (sortie) est une technique de test dans laquelle des combinaisons de valeurs d'entrée sont testées de manière systématique.
- **Test par paire** - Le comportement du logiciel dépend de plusieurs paramètres. Dans les tests par paires, les paramètres multiples sont testés par paires pour leurs différentes valeurs.
- **Test basé sur l'état** - Le système change d'état lors de la fourniture d'une entrée. Ces systèmes sont testés en fonction de leurs états et de leurs entrées.

IX.3.2. TEST DE LA BOITE BLANCHE

Il est effectué pour tester le programme et sa mise en œuvre, afin d'améliorer l'efficacité ou la structure du code. Il est également connu sous le nom de « *test structurel* ». Dans cette méthode de test, le testeur connaît la conception et la structure du code. Les programmeurs du code effectuent ce test sur le code. Voici quelques techniques de test de la boîte blanche :

- **Test de flux de contrôle** - Le but du test de flux de contrôle est de configurer des scénarios de test couvrant toutes les instructions et conditions de la branche. Les conditions de branche sont testées pour être à la fois vrai et faux, de sorte que toutes les instructions peuvent être couvertes.

- **Test du flux de données** - Cette technique de test vise à couvrir toutes les variables de données incluses dans le programme. Il teste où les variables ont été déclarées et définies et où elles ont été utilisées ou modifiées.

IX.4. NIVEAUX DES TESTS LOGICIELS

Le test lui-même peut être défini à différents niveaux de SDLC. Le processus de test est parallèle au développement logiciel. Avant de passer à l'étape suivante, une étape est testée, validée et vérifiée. Les tests séparés sont effectués uniquement pour vous assurer qu'il ne reste aucun bogue ou problème caché dans le logiciel. Le logiciel est testé à différents niveaux :

- **Test d'unité** - Pendant le codage, le programmeur effectue des tests sur cette unité de programme pour savoir si elle est exempte d'erreur. Les tests sont effectués dans le cadre d'une approche de test en boîte blanche. Les tests unitaires aident les développeurs à décider que des unités individuelles du programme fonctionnent conformément aux exigences et sont exemptes d'erreur.

- **Test d'intégration** - Même si les unités de logiciel fonctionnent correctement, il est nécessaire de déterminer si les unités, si elles étaient intégrées ensemble, fonctionneraient également sans erreur. Par exemple, la transmission d'arguments et la mise à jour des données, etc.

- **Test du système** - Le logiciel est compilé en tant que produit, puis testé dans son ensemble. Cela peut être accompli en utilisant un ou plusieurs des tests suivants :

→ **Test de fonctionnalité** - Teste toutes les fonctionnalités du logiciel par rapport aux exigences.

→ **Test de performance** - Ce test prouve l'efficacité du logiciel. Il teste l'efficacité et le temps moyen pris par le logiciel pour effectuer la tâche souhaitée. Les tests de performance sont effectués au moyen de tests de charge et de tests de contrainte, dans lesquels le logiciel est soumis à une charge utilisateur et de données importante, dans diverses conditions environnementales.

→ **Sécurité et portabilité** - Ces tests sont effectués lorsque le logiciel est censé fonctionner sur différentes plates-formes et accessible par nombre de personnes.

- **Test d'acceptation** : Lorsque le logiciel est prêt à être remis au client, il doit passer par la dernière phase de test, où il est testé pour l'interaction et la réponse de l'utilisateur. Ceci est important car même si le logiciel répond à toutes les exigences de l'utilisateur et si celui-ci n'aime pas son apparence ou son fonctionnement, il peut être rejeté.

  → **Tests alpha** - L'équipe de développeurs effectue elle-même des tests alpha en utilisant le système comme s'il était utilisé dans un environnement de travail. Ils essaient de savoir comment les utilisateurs réagiraient à certaines actions du logiciel et comment le système devrait réagir aux entrées.

  → **Test bêta** - Une fois le logiciel testé en interne, il est confié aux utilisateurs pour être utilisé dans leur environnement de production uniquement à des fins de test. Ce n'est pas encore le produit livré. Les développeurs s'attendent à ce que les utilisateurs, à ce stade, apportent des problèmes minutieux, qui ont été ignorés.

- **Les tests de régression** : Chaque fois qu'un logiciel est mis à jour avec un nouveau code, une nouvelle fonctionnalité ou une nouvelle fonctionnalité, il est testé de manière approfondie pour détecter toute incidence négative du code ajouté. Ceci est connu sous le nom de test de régression.

IX.5. DOCUMENTATION DE TEST LOGICIEL

Les documents de test sont préparés à différentes étapes :

- **Avant de tester :** Le test commence par la génération de cas de test. Les documents suivants sont nécessaires pour référence :
  - → **Document SRS** - Document sur les exigences fonctionnelles ;
  - → **Document relatif à la politique de test** - il décrit la distance à laquelle les tests doivent être effectués avant la publication du produit ;
  - → **Document de stratégie de test** - Ce document mentionne des aspects détaillés de l'équipe de test, de la matrice de responsabilité et des droits / responsabilité du responsable du test et de l'ingénieur de test.
  - → **Document de matrice de traçabilité** - Il s'agit d'un document SDLC, lié au processus de collecte des exigences. Au fur et à mesure que de nouvelles exigences apparaissent, elles sont ajoutées à cette matrice. Ces matrices aident les testeurs à connaître la source des besoins. Ils peuvent être tracés en avant et en arrière.
- **En cours de test :** Les documents suivants peuvent être requis lors du démarrage et de la réalisation des tests :
  - → **Document sur le scénario de test** - Ce document contient la liste des tests requis. Il comprend un plan de test unitaire, un plan de test d'intégration, un plan de test système et un plan de test d'acceptation.
  - → **Description du test** - Ce document est une description détaillée de tous les cas de test et des procédures pour les exécuter.
  - → **Rapport de cas de test -** Ce document contient le rapport de cas de test résultant du test.
  - → **Journaux de test** - Ce document contient des journaux de test pour chaque rapport de scénario de test.
- **Après le test** : Les documents suivants peuvent être générés après les tests :
  - → **Résumé du test** - Ce résumé du test est une analyse collective de tous les rapports et journaux de test. Il résume et conclut si le logiciel est prêt à être lancé. Le logiciel est publié sous le système de contrôle de version s'il est prêt à être lancé.

IX.6. TESTS ET CONTROLE DE LA QUALITE, ASSURANCE DE LA QUALITE ET AUDIT

Nous devons comprendre que les tests logiciels sont différents de l'assurance de la qualité des logiciels, du contrôle de la qualité des logiciels et de l'audit des logiciels :

- **Assurance qualité des logiciels** - Il s'agit de moyens de surveillance des processus de développement logiciel, qui garantissent que toutes les mesures sont prises conformément aux normes de l'organisation. Cette surveillance est effectuée pour s'assurer que les méthodes de développement de logiciel appropriées ont été suivies.

- **Contrôle de la qualité du logiciel** - Il s'agit d'un système permettant de maintenir la qualité du produit logiciel. Il peut inclure des aspects fonctionnels et non fonctionnels du produit logiciel, qui renforcent la bonne volonté de l'organisation. Ce système veille à ce que le client reçoive un produit de qualité adapté à ses besoins et certifié « *utilisable* ».

- **Audit logiciel** - Il s'agit d'un examen de la procédure utilisée par l'organisation pour développer le logiciel. Une équipe d'auditeurs, indépendante de l'équipe de développement, examine le processus logiciel, les procédures, les exigences et d'autres aspects de SDLC. L'audit de logiciel a pour but de vérifier que le logiciel et son processus de développement, à la fois en conformité avec les normes, les règles et les règlements.

CHAPITRE X. PRESENTATION DES OUTILS DE GESTION DE LOGICIELS

CASE est synonyme de génie logiciel assisté par ordinateur. Cela signifie le développement et la maintenance de projets logiciels à l'aide de divers outils logiciels automatisés.

X.1. LES OUTILS « CASE »

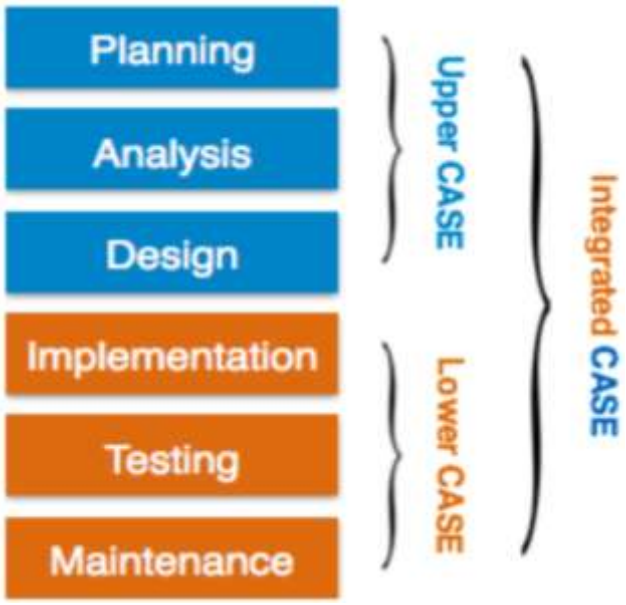
Les outils CASE sont un ensemble de programmes d'application utilisés pour automatiser les activités SDLC. Les outils CASE sont utilisés par les gestionnaires de projets logiciels, les analystes et les ingénieurs pour développer des systèmes logiciels. De nombreux outils CASE sont disponibles pour simplifier les différentes étapes du cycle de vie du développement logiciel, tels que les outils d'analyse, les outils de conception, les outils de gestion de projet, les outils de gestion de base de données, les outils de documentation, pour en nommer quelques-uns.

L'utilisation des outils CASE accélère le développement du projet pour produire le résultat souhaité et permet de détecter les failles avant de passer à l'étape suivante du développement logiciel.

X.2. COMPOSANTS DES OUTILS CASE

Les outils CASE peuvent être divisés dans les parties suivantes en fonction de leur utilisation à un stade particulier du SDLC :

- **Référentiel central** - Les outils CASE nécessitent un référentiel central, qui peut servir de source d'informations communes, intégrées et cohérentes. Le référentiel central est un lieu de stockage central où sont stockées les spécifications du produit, les cahiers des charges, les rapports et diagrammes associés, ainsi que d'autres informations utiles concernant la gestion. Le référentiel central sert également de dictionnaire de données.



- **Outils Upper Case** - Les outils Upper CASE sont utilisés dans les étapes de planification, d'analyse et de conception de SDLC.
- **Outils Lower Case** - Les outils Lower CASE sont utilisés pour la mise en œuvre, les tests et la maintenance.
- **Outils de cas intégrés**- Les outils CASE intégrés sont utiles à toutes les étapes du processus SDLC, de la collecte des exigences aux tests et à la documentation.

Les outils CASE peuvent être regroupés s'ils ont une fonctionnalité, des activités de processus et une capacité d'intégration similaires avec d'autres outils.

X.3. TYPES DES OUTILS DE « CASE »

Juste pour rappel, La portée des outils CASE se retrouve dans le cycle de vie de développement logiciel (SDLC). Maintenant, nous allons brièvement passer en revue divers outils CASE :

- **Les Outils de diagramme** - Ces outils sont utilisés pour représenter les composants du système, les données et les flux de contrôle entre divers composants logiciels et la structure du système sous forme graphique. Par exemple, l'outil *Flow Chart Maker* pour créer des organigrammes à la pointe de la technologie.

- **Les Outils de modélisation de processus** - La modélisation de processus est une méthode permettant de créer un modèle de processus logiciel, utilisé pour développer le logiciel. Les outils de modélisation de processus aident les responsables à choisir un modèle de processus ou à le modifier en fonction des exigences du logiciel. Par exemple, *EPF Composer.*

- **Les Outils de gestion de projet** - Ces outils sont utilisés pour la planification de projet, l'estimation des coûts et des efforts, la planification de projet et la planification des ressources. Les gestionnaires doivent respecter scrupuleusement l'exécution du projet avec chaque étape mentionnée de la gestion de projet logiciel. Les outils de gestion de projet aident à stocker et à partager les informations de projet en temps réel dans l'ensemble de l'organisation. Par exemple, *Creative Pro Office, Trac Project, Basecamp.*

- **Les Outils de documentation** - La documentation dans un projet logiciel commence avant le processus logiciel, couvre toutes les phases de SDLC et après l'achèvement du projet.

  Les outils de documentation génèrent des documents pour les utilisateurs techniques et les utilisateurs finaux. Les utilisateurs techniques sont principalement des professionnels internes de l'équipe de développement qui se réfèrent au manuel du système, au manuel de référence, au manuel de formation, aux manuels d'installation, etc. Les documents de l'utilisateur final décrivent le fonctionnement et les procédures du système, tels que le manuel d'utilisation. Par exemple, *Doxygen, DrExplain, Adobe RoboHelp* pour la documentation.

- **Les Outils d'analyse** - Ces outils aident à rassembler les exigences, à vérifier automatiquement toute incohérence, inexactitude dans les diagrammes, redondances de données ou omissions erronées. Par exemple, *Accepter 360, Accompa, CaseComplete* pour l'analyse des besoins, Analyste visible pour l'analyse totale.

- **Les Outils de conception** - Ces outils aident les concepteurs de logiciels à concevoir la structure de blocs du logiciel, qui peut ensuite être décomposée en modules plus petits en utilisant des techniques de raffinement. Ces outils fournissent des détails sur chaque module et les interconnexions entre les modules. Par exemple, *conception de logiciel d'animation*.

- **Les Outils de gestion de la configuration** - Une instance de logiciel est publiée sous une version. Les outils de gestion de la configuration traitent de :

  → Gestion des versions et des révisions ;
  → Gestion de la configuration de base ;
  → Gestion du contrôle des modifications.

Les outils CASE y contribuent par le suivi automatique, la gestion des versions et des versions. Par exemple, *Fossil, Git, Accu REV*.

- **Les outils de changement de contrôle** - Ces outils sont considérés comme faisant partie des outils de gestion de la configuration. Ils traitent des modifications apportées au logiciel après la fixation de sa base ou la première publication du logiciel. Les outils CASE automatisent le suivi des modifications, la gestion des fichiers, la gestion du code, etc. Cela aide également à appliquer la politique de changement de l'organisation.

- **Les Outils de programmation -** Ces outils comprennent des environnements de programmation tels que IDE (Integrated Development Environment), une bibliothèque de modules intégrée et des outils de simulation. Ces outils fournissent une aide complète pour la création de logiciels et incluent des fonctionnalités de simulation et de test. Par exemple, *Cscope* pour rechercher du code dans C, Eclipse.

- **Les Outils de prototypage** - Le prototype du logiciel est une version simulée du produit logiciel prévu. Le prototype donne l'apparence initiale du produit et simule peu d'aspect du produit réel.

  Les outils de prototypage CASE viennent essentiellement avec des bibliothèques graphiques. Ils peuvent créer des interfaces utilisateur et une conception indépendantes du matériel. Ces outils nous aident à construire des prototypes rapides basés sur des informations existantes. En outre, ils fournissent une simulation du prototype du logiciel. Par exemple, le prototype *du compositeur Serena, Mockup Builder.*

- **Les Outils de développement Web** - Ces outils vous aident à concevoir des pages Web contenant tous les éléments connexes, tels que les formulaires, le texte, les scripts, les graphiques, etc. Les outils Web fournissent également un aperçu en direct de ce qui est en cours de développement et de son apparence une fois terminé. Par exemple, *Fontello, Adobe Edge Inspect, Foundation 3, Brackets*.

- **Les Outils d'assurance qualité -** L'assurance qualité dans une organisation logicielle surveille le processus d'ingénierie et les méthodes adoptées pour développer le logiciel afin de garantir la conformité de la qualité aux normes de l'organisation. Les outils d'assurance qualité comprennent des outils de configuration et de contrôle des modifications, ainsi que des outils de test logiciel. Par exemple, *SoapTest, AppsWatch, JMeter.*

- **Les Outils de maintenance** - La maintenance logicielle inclut les modifications apportées au logiciel après sa livraison. Les techniques de journalisation automatique et de rapport d'erreur, la génération automatique de tickets d'erreur et l'analyse de la cause première sont quelques outils de CASE, qui aident l'organisation logicielle dans la phase de maintenance du SDLC. Par exemple, *Bugzilla pour le suivi des défauts, HP Quality Center*.

CONCLUSION

Le génie logiciel est la science des bonnes pratiques de développement de logiciel. Cette science étudie en particulier la répartition des phases dans le temps, les bonnes pratiques concernant les documents clés que sont le cahier des charges, le diagramme d'architecture ou le diagramme de classes. Le but recherché est d'obtenir des logiciels de grande ampleur qui soient fiables, de qualité, et correspondent aux attentes de l'usager.

Par contre, *Le développement de logiciel* consiste à étudier, concevoir, construire, transformer, mettre au point, maintenir et améliorer des logiciels. Ce travail est effectué par les employés d'éditeurs de logiciels, de sociétés de services et d'ingénierie informatique (SSII), des travailleurs indépendants (*freelance*) et des membres de la communauté du logiciel libre. Le développement d'une application exige de :

- *Procéder par étapes,* en prenant connaissance des besoins de l'utilisateur ; effectuer l'analyse ; trouver une solution informatique ; réaliser ; tester ; installer et assurer le suivi.
- *Procéder avec méthode*, en partant du général aux détails et aux techniques ; fournir une documentation ; s'aider de méthodes appropriées ; et Savoir se remettre en question.
- *Choisir une bonne équipe,* en trouvant les compétences adéquates ; définir les missions de chacun et coordonner les différentes actions pour la construction du logiciel.

- *Contrôler les coûts et délais,* en considérant l'aspect économique ; maîtriser de la conduite du projet ; et investissements aux bons moments.
- *Garantir le succès du logiciel,* en répondant à la demande et assurer la qualité du logiciel.
- Envisager l'évolution du logiciel, du matériel et de l'équipe.

Ainsi, Un logiciel est créé petit à petit par une équipe d'ingénieurs conformément à un cahier des charges établi par un client demandeur ou une équipe interne. Le logiciel est décomposé en différents modules et un chef de projet, ou *architecte*, se charge de la cohérence de l'ensemble. Différentes activités permettent de prendre connaissance des attentes de l'usager, créer un modèle théorique du logiciel, qui servira de plan de construction, puis construire le logiciel, contrôler son bon fonctionnement et son adéquation au besoin. La planification et la répartition des travaux permettent d'anticiper le délai et le coût de fabrication.

Le logiciel est accompagné d'une procédure d'installation, d'une procédure de vérification de bonne installation, de documentation (parfois créé automatiquement à partir de commentaires placés à cet effet dans le code source) et d'une équipe d'assistance au déploiement et à la maintenance, désignée sous le nom de support. Outre les travaux d'analyse, de conception, de construction et de tests, une procédure de *recette*, permettra de déterminer si le logiciel peut être considéré comme utilisable.

Nous espérons que le support de cours de l'introduction au Génie logiciel, vous a été utile et vous permettra d'approfondir vos connaissances informatiques relatives à la conception et à la production des logiciels. L'auteur de cet ouvrage, le Docteur YENDE RAPHAEL Grevisse, vous remercie prestement du temps que vous consacrerez à l'exploitation de support de cours. N'hésitez pas à nous contacter pour plus d'information. Bonne lecture et digestion.