Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Dr. Tahar Moulay de Saida

Faculté de Technologie

Département d'Informatique

Génie Logiciel II

Présenté par:

Dr. Mostefai Abdelkader

Maître de conférences « B » en Informatique

Mars 2021

Sommaire

Liste des Figures

Liste des Tables

Introduction

Le génie logiciel ou l'ingénierie logicielle (en anglais software engineering) est un domaine qui étudie comment développer et évoluer des logiciels complexes et de qualité sous contraintes de ressources, de coût et de délai.

Le génie logiciel se définit souvent par opposition à la programmation qui est considérée comme facile et consiste à produire un programme par une seule personne. Dans le cas de génie logiciel, l'objectif est la création par de nombreux développeurs des logiciels complexes qui se composent de milliers de lignes de code. L'étude de ce domaine ne se limite pas à l'apprentissage de son vocabulaire, mais il contient des approches, techniques, méthodologies, méthodes d'organisation et de la gestion, outils et de bonnes pratiques que l'étudiant doit maitriser[1,13].

Ce polycopié de cours se veut une continuité du module génie logiciel de deuxième année licence informatique. Son objectif est de présenter à l'étudiant quelques principes, méthodes et techniques avancés du génie logiciel. Il est composé d'un ensemble de cours et travaux dirigés destinés aux étudiants de troisième année licence informatique. Le contenu de ces cours et TDs est conçu conformément au programme de la matière fixé par la tutelle. Ce programme est présenté en annexe1.

Ce polycopié est divisé en dix chapitres.

Le chapitre 1 introduit et définit le domaine génie logiciel, présente les différents modèles de processus logiciels et définit en détail la qualité logicielle et ses modèles.

Les chapitres 2 à 6 présentent en détail les principes et techniques de gestion de projet logiciel. Chapitre 2 définit le concept projet et gestion de projet. Chapitre 3 présente les différentes techniques d'estimation de la charge. Chapitre 4 présente les principes et techniques de gestion de contenu tels que les techniques de collecte des besoins, la PBS, la WBS, les techniques de planification (PERT , GANTT).

Chapitre 5 définit le risque et présente les techniques couramment utilisées pour le gérer. Chapitre 6 présente quelques techniques pour assurer la qualité logicielle.

Chapitre 7 explique en détail comment suivre et contrôler la progression du projet.

Chapitre 8 discute le rôle de la composante humaine dans la réussite de projet.

Chapitre 9 présente la problématique de test logiciel et les techniques utilisées pour le conduire.

Chapitre10 définit la mesure logicielle et présente son rôle dans l'ingénierie logicielle.

Objectifs du cours

L'objectif principal de ces cours et TDs est de connaitre et maitriser quelques approches, techniques et méthodes utilisées dans le développement des logiciels complexes.

A la fin de ces cours l'étudiant doit être capable de :

1. Comprendre ce que c'est le Génie Logiciel ainsi que ses objectifs,
2. Connaître les différents processus de développement logiciel,
3. Comprendre ce que c'est la gestion de projet ainsi que ses objectifs,
4. Connaitre et maitriser les principes et techniques de gestion de projet logiciel (estimation de la charge, planification, gestion des risques, assurance de la qualité, suivi et contrôle de projet.etc).
5. Comprendre ce que c'est le test logiciel ainsi que ses objectifs.
6. Maîtriser les différentes techniques et approches de test.
7. Comprendre le rôle de la mesure dans le domaine de génie logiciel.
8. Connaitre les différentes métriques logicielles.

Chapitre 1 - **Génie logiciel**

1.1 - **Etat des projets logiciels**

Les statistiques de l'analyse de 8380 projets informatiques de 365 compagnies effectuée par le Standish Group International en 1995 montrent que (http://www.standishgroup.com) :

- 1 projet sur 6 (16%) de projets sont réalisés dans les délais et en respectant les coûts prévus
- 1 projet sur 3 échoue (annulé en cours de développement)
- Dans 53% des projets réalisés les coûts et les délais fixés à l'avance sont dépassés.

D'après une enquête mondiale réalisée par le Cabinet Deloitte Consulting en 1998, les causes d'échec les plus probables sont :

• Résistances au changement : 82 % ;
• Engagement de la direction insuffisant : 72 % ;
• Objectifs non réalistes : 65 % ;
• Faiblesses dans la conduite de projet : 54 % ;
• Raisons du changement non convaincantes : 46 % ;
• Compétences de l'équipe projet insuffisantes : 44 % ;
• Périmètre du projet mal défini : 44 % ;
• Pas de gestion du changement : 43 % ;
• Absence de vision transversale : 41 % ;
• Aspects technologiques négligés : 36 %.

Enfin, d'après une autre enquête du Stanish Group International (« *Chaos report* »)
En 1998 sur 7500 applications informatiques, les projets qui réussissent s'expliquent
Par la combinaison des facteurs suivants (total de 100 %) :

• Implication des utilisateurs : 20 % ;
• Soutien des responsables : 15 % ;
• Objectifs clairs : 15 % ;
• Chef de projet expérimenté : 15 % ;
• Jalonnement du projet : 10 % ;
• Planification adéquate : 5 % ;
• Appropriation : 5 % ;
• Equipe compétente : 5 % ;
• Besoins stables : 5 % ;
• Autres : 5 %.

Ces informations montrent clairement le défi qui face le domaine du génie logiciel.

1.2 - **Logiciel**

Le Webster's New Intercollegiate Dictionary définit le logiciel comme suit :

"Software is the entire set of programs, procedures and related documentation associated with a system and especially a computer system."

Blum (1992) donne la définition suivante :

"Software is the detailed instructions that control the operation of a computer system. Its functions are to (1) manage the computer resources of the organisation, (2) provide tools for human beings to take advantage of these resources, and (3) act as an intermediary between organisations and stored information."

Le journal officiel français le définit comme « l'ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitements de l'information' » (arrêté du 22 déc. 1981)

A partir de ces définitions, il est important de noter que, contrairement à l'idée populaire, le logiciel n'est pas formé seulement des programmes (code source) mais il est composé aussi de procédures et de la documentation nécessaires pour comprendre, maintenir, installer et utiliser le logiciel.

Le logiciel comme partie du système d'information a ses propres caractéristiques. Ces propriétés le rend différent des autres produits (ouvrages) et marquent la problématique de sa qualité [12]:

• il est immatériel ;
• il est reproductible ;
• il nécessite une maintenance ;
• il possède une dimension subjective

1.3 - **Engineering (Ingénierie)**

Le terme engineering est définit par le New Intercollegiate Webster's Dictionary, 1979 comme :

"the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to man in structures, machines, products, systems and

processes."

'Étude d'un projet industriel sous tous ses aspects (techniques, économiques, financiers, monétaires et sociaux) et qui nécessite un travail de synthèse coordonnant les travaux de plusieurs équipes de spécialistes'. Dictionnaire LAROUSSE

Donc on peut dire que l'ingénierie est l'application de la science pour résoudre des problèmes réels.

1.4 - **Génie logiciel**

Ce domaine a vu le jour entre le 7 et le 11 octobre 1968, à Garmisch-Partenkirchen, avec la présence de l'OTAN comme réponse à la crise logicielle. Cette crise se manifeste par les symptômes suivants qui persistent à ce jour malgré les avancées réalisées dans ce domaine [1]:

1) Le logiciel n'est pas fiable,
2) Le logiciel ne répond pas aux attentes de ses clients (n'est pas valide).
3) Il est difficile de respecter les délais et les budgets prévus à l'avance.
4) La qualité du logiciel n'est pas satisfaisante
5) Il est difficile de mesurer la progression du logiciel
6) Il est difficile de comprendre comment travaille le groupe de développeurs.

Le Génie logiciel en anglais software engineering est définit : « Discipline de l'informatique qui regroupe un ensemble de connaissances, de procédés et des acquis scientifiques pour la conception, la mise en œuvre, la vérification et la documentation de logiciels dans le but d'en optimiser la production, le support et la qualité ». (Grand dict. terminologique).

Selon L'IEEE 610.12 : Le Génie Logiciel (GL) est l'application d'une approche systématique, disciplinée et quantifiable au développement, à l'exploitation et à la maintenance du logiciel. C'est-à-dire, l'application de l'ingénierie au logiciel.

Une définition récente proposée par Wang and King (2000) est :

"Software engineering is a discipline that adopts engineering approaches such as established methodologies, process, tools, standards, organisation methods, management methods, quality assurance systems, and the like to develop large-scale software with high productivity, low cost, controllable quality, and measurement development schedules."

Le GL se fixe comme objectif le développement des logiciels qui respecte la règle du CQFD suivante : Coût Qualité Fonctionnalités Délai[1 ,2,19].

- Le logiciel développé doit offrir les fonctionnalités attendues par ses utilisateurs ;
- Les coûts du développement fixés à l'avance ne doivent pas être dépassés.
- Les délais fixés à l'avance ne doivent pas être dépassés.
- La qualité du logiciel doit être conforme au contrat de service initial.

1.5 - **Processus logiciel**

Un processus logiciel est un ensemble d'activités liées selon un certain ordre. Ces activités dirigent la production du système logiciel. Vu l'existence de différents types de logiciels, il n'y a pas de méthode universelle applicable à tous ces types. Par conséquent, il n'existe aucun processus logiciel universel[4].

Les entreprises appliquent différents processus selon le type de système logiciel à développer, les besoins de client et l'expérience de l'entreprise.

Selon Sommerville, Bien qu'il existe différents processus logiciels, ils doivent tous inclure, sous une forme ou une autre, quatre activités fondamentales [4]:

1.**La Spécification du logiciel** : définition de besoins fonctionnels et non fonctionnels.

2. **Le développement de logiciel** : production du logiciel répondants aux spécifications.

3. **Validation du logiciel :** Le logiciel doit être validé pour s'assurer qu'il fait ce que le client veut.

4**. Evolution du logiciel :** Le logiciel doit évoluer pour répondre aux besoins changeants des clients.

Ces activités sont des activités complexes et sont composées de sous-activités telles que[4] :

1. **Etude de faisabilité –** étudier si le développement est bénéfique pour l'entreprise.
2. **Etudes des besoins** - Déterminer les besoins fonctionnels et non fonctionnels.
3. **Planification du projet**

- Déterminer comment développer le logiciel.

-Analyse des coûts : faire des estimations de coûts.

-Planification - Création d'un calendrier pour le développement.

-Assurance de la qualité des logiciels - Déterminer les activités qui vous aideront à assurer la qualité du produit.

-Work-breakdown structure—déterminer les sous-taches nécessaire pour réaliser le produit.

4. **Conception -** Déterminer comment le logiciel doit fournir les fonctionnalités.

Conception architecturale - Conception de la structure du système.

Conception d'interface - Spécification des interfaces entre les parties de système.

Conception de la base de données : conception de la structure de la base de données.

Conception détaillée - Conception de chaque partie de produit (algorithme et structure de données)

5. **Implémentation** – production du logiciel en utilisant une technologie de choix.
6. **Test** - Exécution du logiciel avec des données pour trouver les défaillances.

Les tests à faire sont : Tests unitaires, Test d'intégration, Test du système, Test d'acceptation qui est un test effectué pour valider les besoins de client.

Test de régression : exécution sur la nouvelle version de produit, les tests effectués sur la version précédente pour voir si la nouvelle version conserve les capacités précédentes.

7. **Livraison -** Fournir au client une solution logicielle efficace.
8. **Installation —** mettre la solution sur le site opérationnel du client.
9. **Formation :** faire apprendre aux utilisateurs à utiliser le logiciel.
10. **Maintenance** - Mise à jour et amélioration du logiciel pour garantir la continuité de son utilité.

Le résultat de la réalisation de ces activités se matérialise par la production de différents types de documents. Ces documents gardent trace du travail effectué.

1.6 - **Types de documents[13]**

1. **Enoncé des travaux** (statement of work) : description préliminaire des besoins. Ce document est généralement créé par client.
2. **Spécification des exigences logicielle (**Software requirements specification) : description de ce que doit faire le logiciel à développer.

2.1 Le modèle objet : montre les principaux objets et classes.

2.2 les scénarios des cas d'utilisation : montre les comportements possibles du système de point de vue utilisateur.

3. **Planning du projet** : montre les taches, les relations entre eux et l'estimation du temps et de l'effort nécessaire pour les accomplir.

4. **Plan des tests** : décrit comment le software doit être testé.

4.1 **Tests d'acceptation** : tests conçus par le client pour déterminer si le system est valide ou non.

5. **Conception de logiciel** : décrit la structure du logiciel.

5.1 **Conception architecturale** : représentation de haut niveau de la structure du système.

5.2 **Conception détaillée** : la conception des modules de bas niveau ou les objets qui composent le système.

6. **Plan de l'assurance de la qualité logicielle (SQA plan)** : décrit la suite des activités à effectuer pour assurer la qualité du système.

7. **Manuel d'utilisateur :** décrit comment utiliser le logiciel.

8. **Source code : le** code du produit développé.

9. **Rapport de test** : liste les tests effectués et comment il s'est comporté le system.

10. **Rapport des bugs** : montre les défaillances de systèmes.

1.7 - **Modèle de processus logiciel**

Un modèle de processus logiciel (parfois appelé Cycle de vie de logiciel ou modèle SDLC) est une représentation simplifiée d'un processus logiciel[4].

Les modèles présentés dans la section suivante sont des modèles génériques et sont des descriptions abstraites de haut niveau des processus logiciels. Ils peuvent être utilisés pour expliquer les différentes approches du développement logiciel[4].

1.7.1 - **Code and fix**

Pas un vrai modèle. Mais c'est ce que fait la plupart des programmeurs en cas de petit projets et des programmes de type démonstration. C'est un processus cyclique, le développeur commence par l'imagination d'une solution. Après, cette solution conceptuelle est codée et testée. En cas d'erreur, l'erreur est fixée. Ces étapes sont répétées jusqu'à ce qu'une version soit produite.

Dans ce modèle il n'y a pas d'exigences formelles, pas de documentation et pas d'assurance qualité. Il n y'a pas même d'une phase de maintenance puisque la version produite est de type démo.

Ce modèle est présenté par la Figure 1.

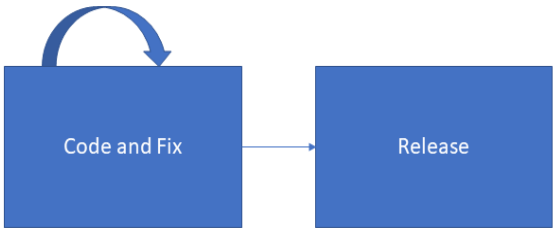


Figure 1 Le modèle code and fix

1.7.2 - **Le modèle en cascade (Waterfall model)**

Ce modèle a été défini par 1970 by Winston Royce et il est considéré comme l'un des plus anciens modèles de processus logiciel. Les phases de ce processus sont présentées par la figure 2.

Ces phases s'exécutent en séquence et aucune phase ne peut démarrer avant que la phase précédente soit terminée.

Il est considéré comme un modèle simple et facile à utiliser.

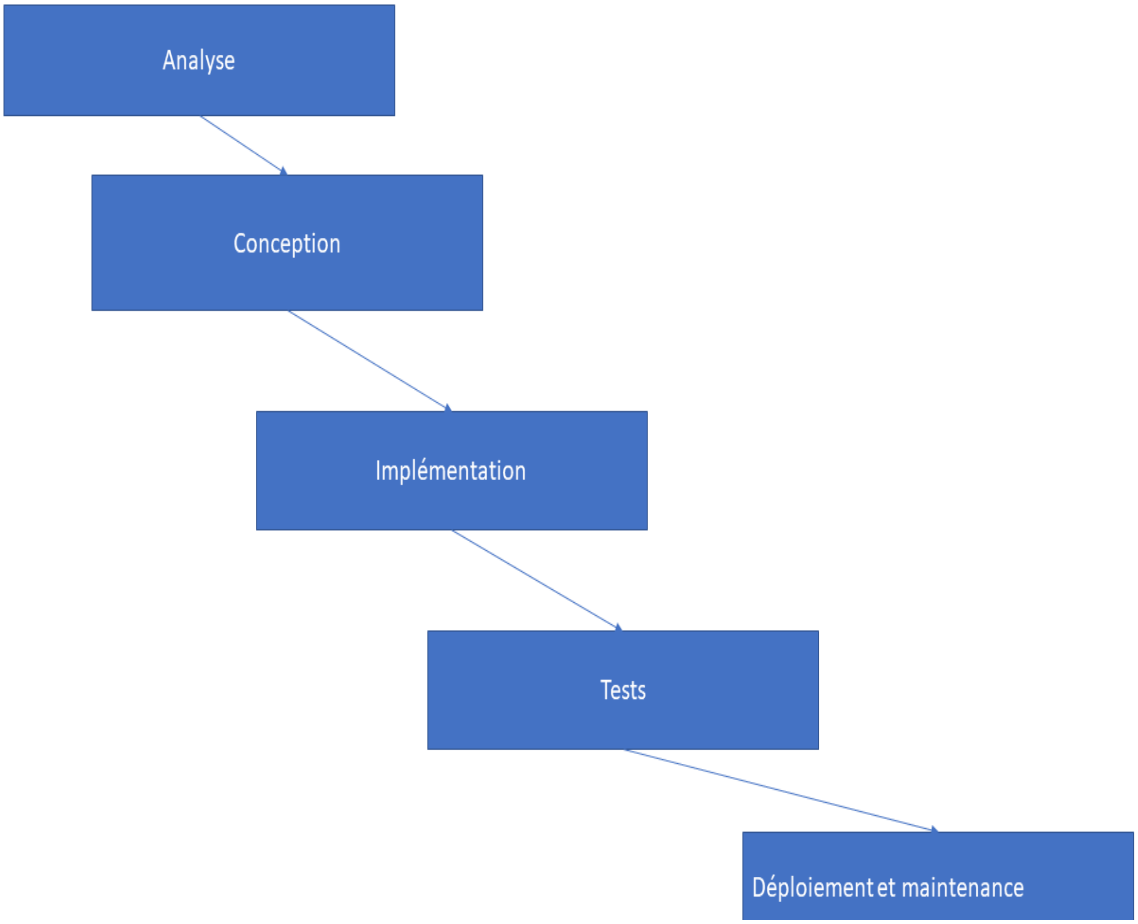


Figure 2 Phases du processus en cascade

1.7.3 - **Le modèle** V

Similaire au modèle en cascade. Mais met l'accent sur les activités V&V (Validation & Vérification). Des tests sont effectués pour chaque activité : des tests d'acceptation sont effectués pour valider les besoins, et des tests d'intégration pendant la phase de conception et des tests unitaire pendant l'implémentation. Pour cette raison, le modèle V est préférable au modèle en cascade.

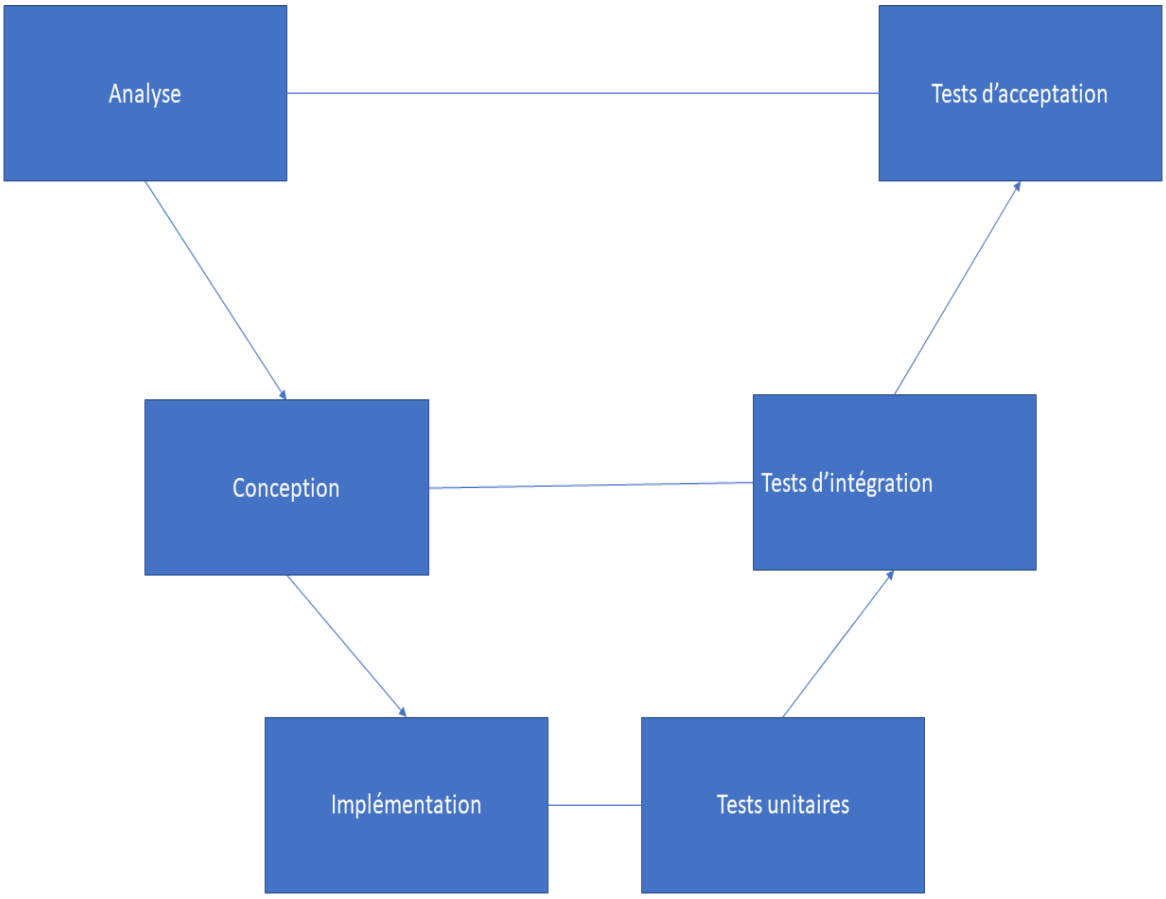


Figure 3 le modèle en V

1.7.4 - **Prototyping**

Dans ce processus un prototype jetable est construit pour aider à comprendre les exigences. Ce prototype est développé sur la base des exigences actuellement connues. Le développement du prototype passe par une phase de conception, de codage et de tests. Ces phases sont généralement faites d'une façon peu formelle ou approfondie. Ce prototype donnera au client une idée réelle sur le futur système ce que lui permettra de mieux comprendre les besoins du système demandé.

1.7.5 - **Le processus Itératif**

Le modèle de processus de développement itératif veut combiner les avantages du prototypage et du modèle en cascade. L'idée de base est que le logiciel doit être développé par incréments,

chaque incrément étends les fonctionnalités déjà développé jusqu'à ce que le système en entier soit produit. Ce processus est présenté par la figure 3.

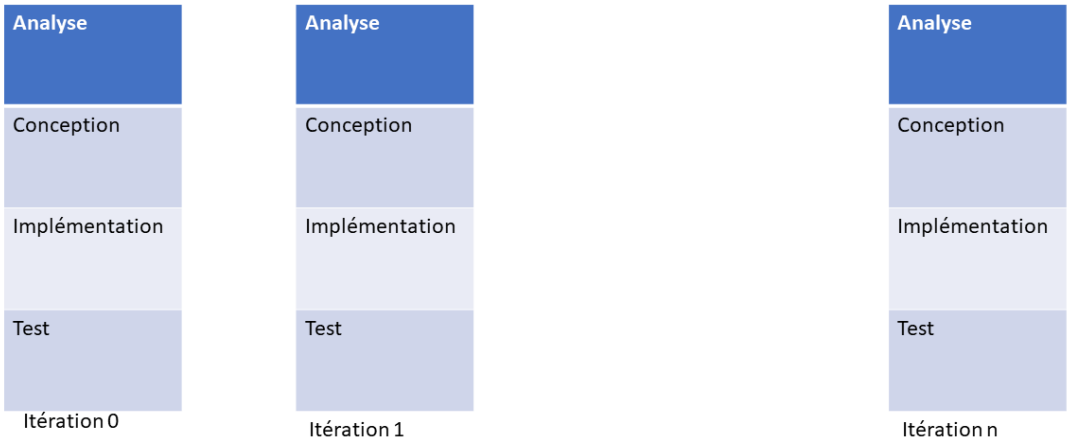


Figure 4 Principe de fonctionnement du processus itératif.

1.7.6 - **Processus Unifié : (Unified Process)**

L'UP est un autre modèle de processus itératif et incrémental. Il a été conçu pour un développement orienté objet en utilisant UML (Unified Language Modeling).

L'UP est un processus piloté par les cas d'utilisation, centré sur l'architecture et basé sur la gestion des risques.

Dans L'UP le développement de logiciels est divisé en cycles, chacun cycle est exécuté comme un projet indépendant et produit une version fonctionnelle.

Chaque cycle lui-même est divisé en quatre phases qui s'exécutent en série :

- Inception

- Elaboration

- Construction

- Transition

Chaque phase peut être réalisée en plusieurs itérations et chaque itération est une exécution en série de plusieurs activités telles que la modélisation de besoins métiers, analyse et conception , implémentation, test, déploiement, gestion de projet et gestion de la configuration (voir figure 5).

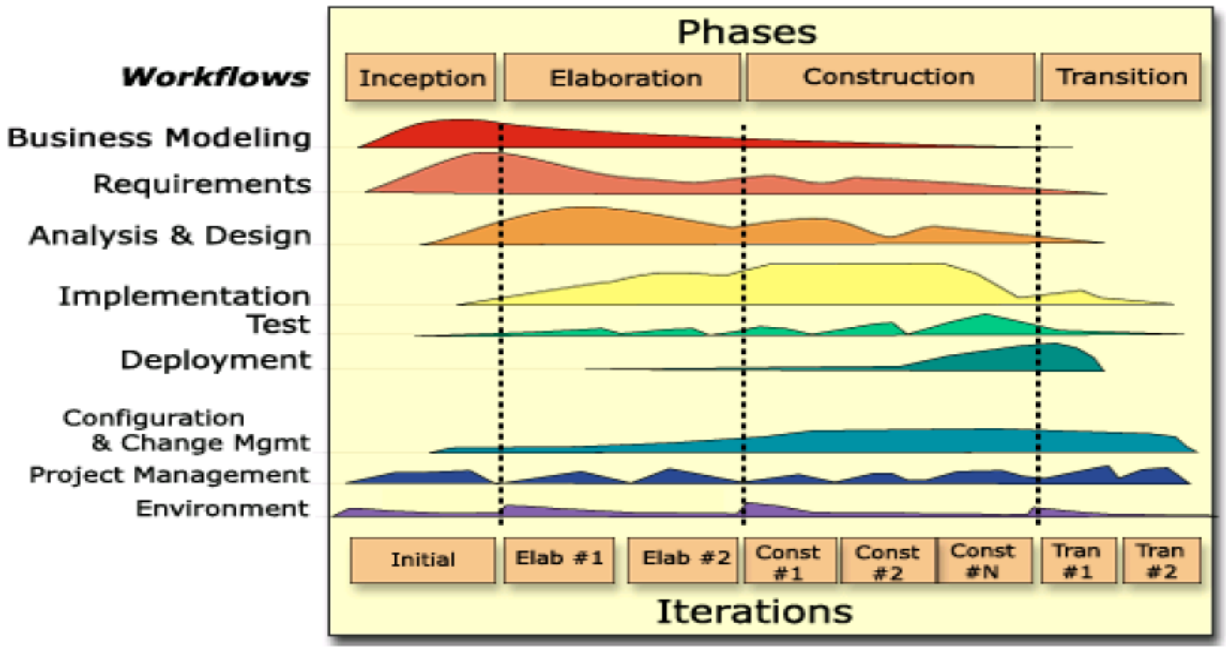


Figure 5 Le processus UP.

Les variantes de UP les plus connues sont :

- RUP : Rational Unified Process
- 2TUP : Two Tracks Unified Process

1.7.7 - **Les méthodes agiles**

Les méthodes agiles telles que XP(i.e., Extreme Programming) et SCRUM sont basées sur une approche de développement itérative, incrémentale et adaptative et visent à être plus pragmatiques pour combler quelques déficits des méthodes classiques de développement logiciel. L'approche agile base le développement logiciel sur une implication maximale du demandeur (client) et vise l'augmentation de la réactivité pour faire face efficacement à ses demandes changeantes[1 ,4,3,12].

Les méthodes agiles sont basées sur la philosophie agile présenté par l'Agile Manifesto [21](agilemanifesto.org, by Kent Beck, Robert C. Martin, Martin Fowler). Cette philosophie est basée sur quatre valeurs fondamentales déclinées en douze principes.

Les 4 valeurs sont :

- Individuals and Interactions Over Processes and Tools.
- Working Software Over Comprehensive Documentation.
- Customer Collaboration Over Contract Negotiation.
- Responding to Change Over Following a Plan.

Et les 12 principes sont présenté par la table suivante :

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
Business people and developers must work together daily throughout the project.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Continuous attention to technical excellence and good design enhances agility.
Simplicity--the art of maximizing the amount of work not done--is essential.
The best architectures, requirements, and designs emerge from self-organizing teams.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Table 1 The agile Minifesto[21]

1.8 - **Les principes du GL**

Les sept principes fondamentaux du GL (proposés par Carlo Ghezzi) sont[2] :

1. Rigueur,
2. Séparation des problèmes (« separation of concerns »),
3. Modularité,
4. Abstraction,
5. Anticipation du changement,
6. Généricité,
7. Construction incrémentale.

1.8.1 - **Rigueur**

Ce principe suggère que l'activité de développement de logiciel doit être conduite d'une manière rigoureuse et formelle. La rigueur et la formalité complémente la créativité demandée durant cette activité. Les méthodes et techniques doivent être basées sur les mathématiques.

1.8.2 - **Séparation des problèmes**

Ce principe a pour objectif la maitrise de la complexité en appliquant la stratégie « diviser pour régner » pour résoudre les problèmes. Elle consiste à considérer séparément les différents aspects d'un problème.

Par exemple :

· On peut séparer dans le temps on traitant les différents aspects l'un à la suite de l'autre par exemple l'utilisation de la notion de cycle de vie du logiciel.

· On peut faire une séparation des 'vues' que l'on peut avoir d'un système (ex : se concentrer sur l'aspect comportement puis on considère l'aspect structure ..etc),

· Séparation du système en parties (un noyau, des extensions, …),

· ..etc.

1.8.3 - **Modularité**

Un système complexe doit être divisé en un ensemble de modules. Ces modules doivent être fortement cohésive et faiblement couplés. Un système qui respecte cette propriété est dit modulaire. Ce type de système facilite la réutilisation et l'évolution.

1.8.4 - **Abstraction**

L'abstraction consiste à identifier les aspects jugés importants à un moment donné et à ignorer les détails.

Un système peut être représenter à différents *niveaux d'abstraction*.

L'abstraction est un outil pour maîtriser la complexité.

1.8.5 - **Anticipation du changement**

Une caractéristique essentielle du logiciel est qu'il doit évoluer sinon il devient obsolète (corrections d'imperfections et pour s'adapter à des *besoins qui changent*).

Ceci demande des efforts particuliers pour *prévoir*, faciliter et gérer ces évolutions inévitables.

1.8.6 - **Généricité**

De préférence il faut penser à une solution plus générale (paramétrable ou adaptable). Cette solution aura le potentiel d'être réutilisable.

1.8.7 - **Construction incrémentale**

De préférence la production doit être faite selon un procédé incrémental qui consiste en une suite des étapes. Le résultat d'une étape est construit en étendant le résultat de l'étape précédente.

1.9 - **Qualité logicielle**

La qualité est un concept controversé et n'a pas de consensus sur sa définition. La vision de la qualité est subjective et on n'a pas de théorie forte pour l'évaluer. Par exemple comment savoir si une entité (logiciel, développeur, étudiant, équipe, produit…etc) est meilleure que l'autre. Ce qui est sûr est que l'absence de la qualité peut causer des dégâts financiers, affecter la santé et pire même il peut causer des pertes de vies humaines.

Pour prouver ce fait on peut citer quelques fameuses erreurs logicielles et leurs graves conséquences.

1. Therac-25 : un appareil médical a causé de 6 morts à cause d'un surdosage de radiations due à une erreur logicielle.
2. Marc Climate Observer : l'écrasement de robot à cause de remplissage de son espace mémoire.
3. Missile Patriot : l'erreur a produit une dérivation de 6 mètres par heure. La conséquence est la mort de 28 soldats à Dhahran.
4. Ariane 5 : réutilisation du code de l'Ariane a causé des pertes estimées de 4, 500 M$.
5. Crash du réseau AT&T (90's) : 9 heures de blocage du réseau à cause d'une mise à jour. Ce blocage a causé des pertes de 60M$. La cause de l'erreur est un break manquant.

En 2002, Le National Institute for Standards and Technology (NIST) a estimé que les problèmes logiciels causent annuellement une perte de 60 Milliard de dollars.

1.9.1 - **Définitions**

Il y 'a plusieurs définitions :

- La qualité, c'est le degré d'excellence.

- La qualité, c'est la valeur de quelque chose pour quelqu'un.

La définition de l'ISO est : la qualité, c'est un ensemble de traits et de caractéristiques d'un produit logiciel portant sur son aptitude à satisfaire des besoins exprimés ou implicites.

L'IEEE définit la qualité comme suit : la qualité correspond au degré selon lequel un logiciel possède une combinaison d'attributs désirés.

Pressman[4]: la qualité c'est la conformité aux exigences explicites à la fois fonctionnelles et de performance aux standards de développement explicitement documentés et aux caractéristiques implicites qui sont attendues de tout logiciel professionnellement développés.

Généralement pour avoir une idée complète de la qualité d'un logiciel on utilise un modèle de qualité.

1.9.2 - **Modèle de qualité**

Le ISO 9126 définit un modèle de qualité comme : « The set of characteristics and the relationships between them, which provide the basis for specifying quality requirements and evaluating quality ».

A partir de cette définition nous comprenons qu'un modèle de qualité fournit une base pour spécifier et évaluer la qualité.

Un modèle de qualité est formé souvent de plusieurs niveaux de lecture. Le premier niveau décrit la qualité selon un point de vue très généraliste. Dans ce niveau on trouve généralement les attributs(facteurs) du produit qui sont intéressants de point de vue d'un évaluateur de système (e.g., utilisateur,entreprise..etc). Ils sont nommés attributs externes parce qu'ils dépendent aussi de l'environnement. Par exemple la maintenabilité, la portabilité, l'utilisabilité,… etc. Ces facteurs sont généralement groupés selon différentes vues (opération, révision, transition) et sont mesurés indirectement en se basant sur d'autres attributs (critères) qui forment le niveau suivant. Par exemple dans le modèle de McCall (section suivante) le facteur efficience est décomposé en efficience de mémoire et de processus. Ces derniers sont mesurés en utilisant des métriques appropriées.

1.9.3 - **Modèle de McCall**

Ce modèle a été proposé par McCall en 1971 est nommé Facteurs Critères Métriques (FCM). Il contient 11 facteurs. Ces facteurs sont définis par vingt-trois critères. Ce modèle est présenté par la Figure 6.

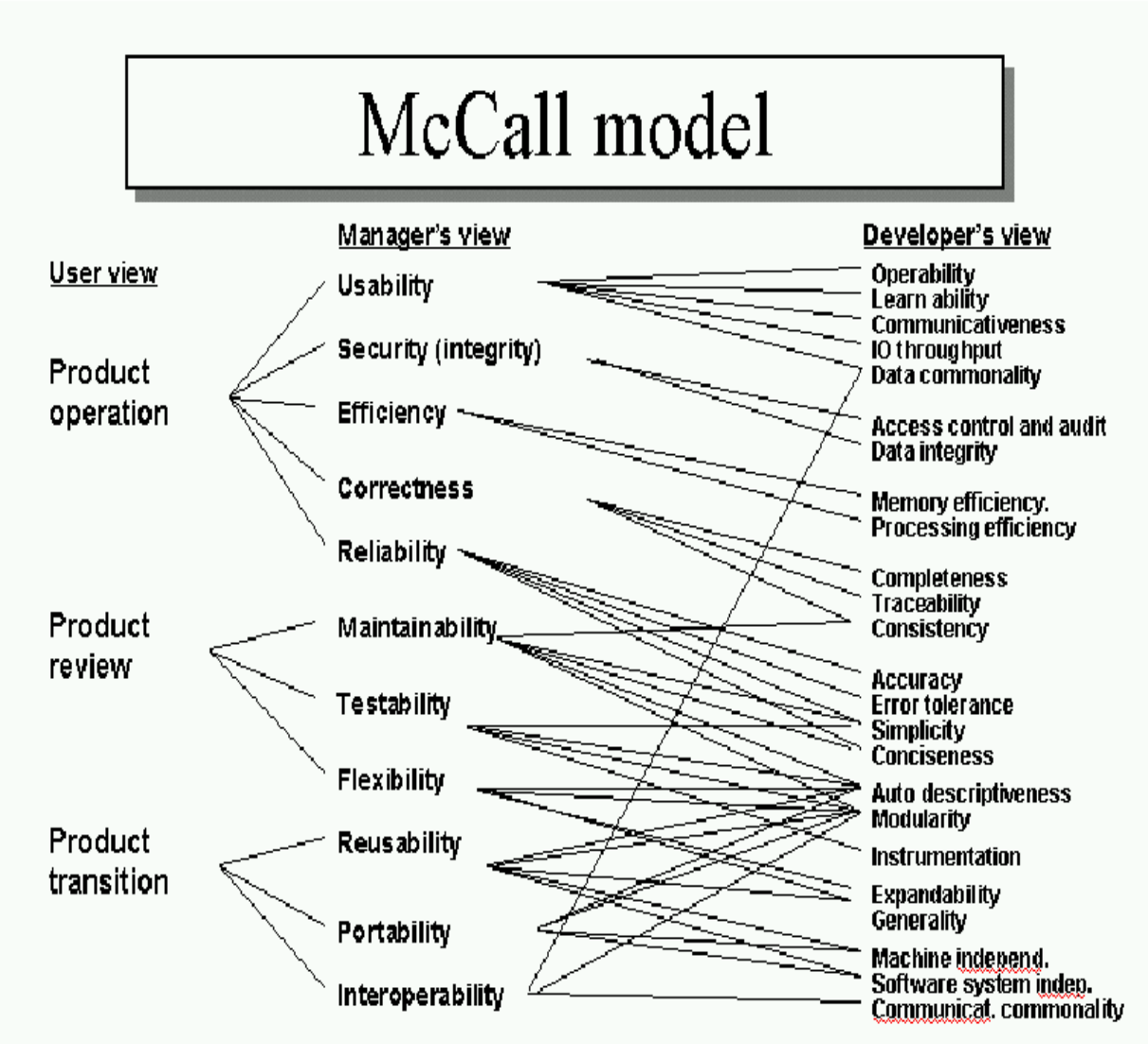


Figure 6 Le modèle McCall[28]

1.9.4 - **Modèle l'ISO/IEC 9126**

Ce modèle est composé de 6 caractéristiques (facteurs) divisés eux même en 21 attributs (Figure 7). Les caractéristiques (facteurs) et les sous caractéristiques de la qualité du produit logiciel selon l'ISO/IEC 9126 sont [24]:

 Capacité fonctionnelle (*Functionality*)[24]

Ensemble d'attributs portant sur l'existence d'un ensemble de fonctions et leurs propriétés données. Les fonctions sont celles qui satisfont aux besoins exprimés ou implicites.

- Pertinence
- Exactitude

- Interopérabilité
- Sécurité
- Conformité

 Fiabilité (*Reliability*)[24]

Ensemble d'attributs portant sur l'aptitude du logiciel à maintenir son niveau de service dans des conditions précises et pendant une période déterminée.

- Maturité (faible fréquence d'apparition des incidents)
- Tolérance aux pannes
- Facilité de récupération : capacité d'un logiciel défectueux à retourner dans un état opérationnel complet (données et connexions réseaux incluses)

 Facilité d'utilisation (*Usability*)[24]

Ensemble d'attributs portant sur l'effort nécessaire pour l'utilisation et sur l'évaluation individuelle de cette utilisation par un ensemble défini ou implicite d'utilisateurs.

- Facilité de compréhension
- Facilité d'apprentissage
- Facilité d'exploitation

 Rendement (*Efficiency*)[24]

Ensemble d'attributs portant sur le rapport existant entre le niveau de service d'un logiciel et la quantité de ressources utilisées, dans des conditions déterminées.

- Comportement temporel : temps de réponse, taux de transactions
- Utilisation des ressources : mémoire, processeur, disque et réseau

 Maintenabilité (*Maintainability*)[24]

Ensemble d'attributs portant sur l'effort nécessaire pour faire des modifications données.

- Facilité d'analyse : identification dans le logiciel de l'origine d'un défaut constaté
- Facilité de modification
- Stabilité

- Testabilité

 Portabilité (*Portability*)[24]

Ensemble d'attributs portant sur l'aptitude de logiciel à être transféré d'un environnement à l'autre.

- Facilité d'adaptation à des changements de spécifications ou d'environnements opérationnels
- Facilité d'installation
- Coexistence
- Interchangeabilité : utilisation de greffons

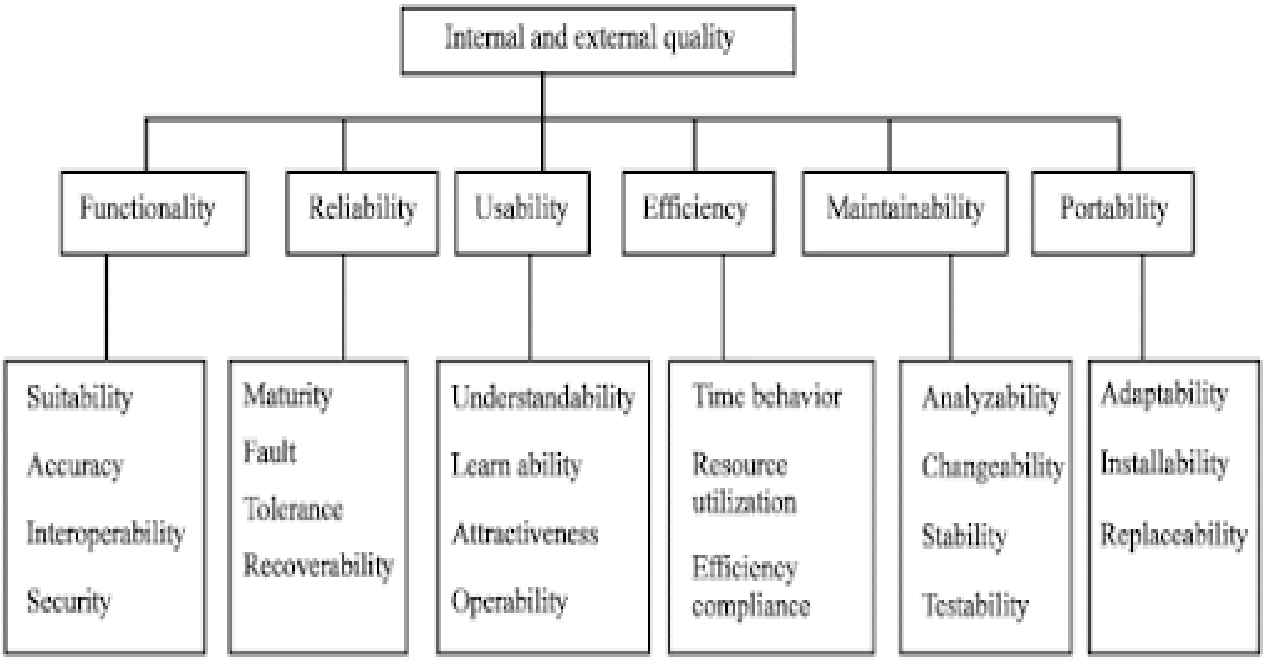


Figure 7 le modèle l'ISO/IEC 9126

Le contenu de la norme ISO/CEI 9126 a été enrichis par la série de normes ISO 250xx, aussi appelée SQuaRE (pour software quality requirements and evaluation, exigences et évaluation de la qualité du logiciel).

Généralement pour évaluer la qualité d'un produit logiciel, on commence par la définition des propriétés de qualité inhérentes à celui-ci. Ces propriétés de qualité sont décomposées en attributs. Ces attributs sont mesurés avec des métriques proposées à ce propos. Les produits logiciels sont classés selon les résultats de ces métriques.

1.10 - **CMMi : Capacity maturity model integrated**

C'est un modèle pour mesurer la maturité de l'entreprise qui élabore le logiciel. Dans ce modèle la maturité est mesurée sur une échelle ordinale de 5 niveaux.

Niveau initial : Le succès des projets dépend des compétences et de la motivation des individus

Niveau piloté : Le développement est effectué en se basant sur les principaux processus de gestion de projets. Le développement est planifié, et suit une démarche d'ingénierie. L'équipe travaille selon des pratiques et procédures connues. Les produits sont vérifiés par rapport aux exigences initiales.

Niveau standardisé : La définition des processus est étendue à l'ensemble de l'entreprise, utilisation des processus améliorés.

Niveau quantifié : L'application systématique de mesure de processus. Des métriques qui servent d'indicateurs sont mises en place et exploitées. L'expérience est capitalisée et réutilisée. Les impacts liés aux évolutions des processus sont évalués.

Niveau optimisé : L'optimisation des processus, mise en place des processus permettant l'amélioration continue, maîtrise du changement

1.11 - **Exercices**

Exercice 1.1 : donner l'ordre des tâches suivantes en cas du modèle en cascade : test d'acceptation, planification de projet, tests unitaires, relecture des besoins, estimation des coûts, conception d'architecturelle, analyse du marché, conception détaillée, test système, revue de conception, mise en œuvre, spécification des besoins.

Exercice 1.2 : Dessinez un diagramme qui représente un modèle de cycle de vie itératif

Exercice 1.3: Comparez les différents processus logiciels.

Chapitre 2 - **Conduite de projet informatique**

2.1 - **Introduction**

Actuellement, le développement et l'évolution des systèmes d'information de l'entreprise est faite en mode projet. La réussite de ces projets requiert, à la fois, des bonnes compétences en ingénierie logicielle et des bonnes compétences en gestion de projet. Le savoir de la gestion de projet est devenu une des clés de réussite des projets au niveau de l'entreprise moderne. Les premières réflexions sur ce savoir ont été commencées en 1950. A ce stade, La gestion de projet n'a été qu'un savoir traitant la planification des ressources pour conduire au mieux les travaux dans différents domaines : aéronautique, travaux publics, militaire, informatique. Actuellement le management de projet est une profession distincte avec des diplômes, des certifications et une carrière pleine d'opportunités. Le référentiel du PMI, appelé Guide du PMBOK (*Project Management Body of Knowledge*) définit et recense les bonnes pratiques de la gestion de projet.

2.2 - **Définitions de projet**

Un projet est l'image d'une situation, d'un état que l'on pense atteindre (Dictionnaire Robert).

Un projet est ce qu'on a l'intention de faire (Larousse).

2.3 - **Les définitions normalisées**

Selon ISO10006 : 2003, un projet est un « processus unique, qui consiste en un ensemble d'activités coordonnées et maîtrisées comportant des dates de début et de fin, entrepris dans le but d'atteindre un objectif conforme à des exigences Spécifiques telles que les contraintes de délais, de coûts et de ressources ».

Le référentiel du PMI, appelé Guide du PMBOK (*Project Management Body of Knowledge*), donne la définition suivante :

« Entreprise temporaire décidée pour obtenir un produit ou un service unique ». L'unicité du produit entraînera celle des activités à mettre en œuvre.

L'AFITEP ((Association francophone de management de projet), définit un projet comme un « ensemble d'actions à réaliser pour satisfaire un objectif défini, dans le cadre d'une mission précise, et pour la réalisation desquelles on a identifié non seulement un début, mais aussi une fin ».

Ces définitions montrent clairement qu'un projet est un travail (activités) à faire sous les contraintes objectif, moyens(coûts), délai. On peut représenter le projet par un triangle de

paramètres délai, moyens et objectif. Cela veut dire que la modification d'un paramètre touchera au moins l'un des deux autres (Figure 5). La qualité se trouve au centre du triangle pour dire que toute modification qui touche l'un des côtés la touchera.

Il y'a deux types de projets [12]:
Les *projets d'ingénierie* qui ont pour objectif la production d'un résultat physique pour un client.
Les *projets produit* ayant pour objectif la création d'un modèle qui fera ensuite l'objet d'une fabrication répétitive. Les projets de système d'information sont des projets de type ingénierie.

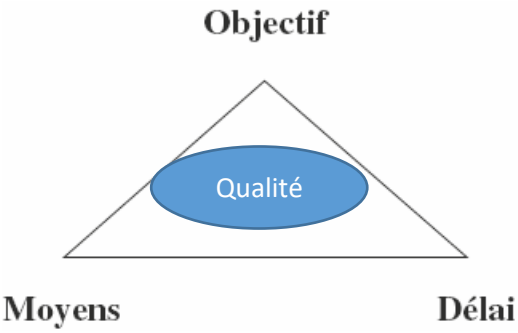


Figure 8 Le triangle projet

Le projet se distingue des opérations effectuées au niveau de l'entreprise. Les opérations sont permanentes et répétitives alors que le projet et temporaire et unique. (PMBOK)
Par exemple l'opération Commande est déclenchée à chaque besoin d'achat. À l'inverse, le projet est unique et ne peut être pris en charge par des procédures standards[12].

2.4 - **Exemples**

1. La création et la mise en ligne d'une application de gestion d'un cabinet médical par un groupe d'étudiants.
2. Développement d'un véhicule autonome.
3. Evolution du parc informatique d'une entreprise selon la technologie de la veille.
4. Construction d'un pont.

2.5 - **Exemples de projets personnels**

Obtenir un diplôme ou une certification
Réalisation d'un rapport.
Réalisation d'un mémoire de fin d'étude.
Planifier un voyage.

2.6 - **Les caractéristiques du projet**

Un projet possède les caractéristiques suivantes :

- Unique et innovant
- Complexe
- Temporaire
- Le Coût et la qualité sont fixés à l'avance.
- A réaliser avec des ressources limitées
- Doit avoir un sponsor
- Comporte de l'incertitude

2.7 - **Les acteurs d'un projet (parties prenantes)**

Le Maître d'ouvrage personne physique ou morale propriétaire de l'ouvrage. Il représente le client. Il détermine les objectifs, le budget et les délais de réalisation. L'AFNOR (Association française de normalisation) définit la maîtrise d'ouvrage comme *« une personne physique ou morale qui sera propriétaire de l'ouvrage [et] assure le paiement des dépenses liées à la réalisation »*.

le Maître ***d'œuvre*** personne physique ou morale qui reçoit mission du maître d'ouvrage pour assurer la conception et la réalisation de l'ouvrage**.** Le maitre d'œuvre est représenté par le chef de projet. L'AFNOR définit la maîtrise d'œuvre comme *« une personne physique ou morale qui réalise l'ouvrage pour le compte du maître d'ouvrage, assure la responsabilité globale de la qualité technique, du délai et du coût »*.

2.8 - **Le chef de projet (gestionnaire de projet)**

Le chef de projet est défini par l'AFITEP et l'AFNOR comme : « *la personne physique chargée par le maître d'œuvre, dans le cadre d'une mission définie, d'assurer la maîtrise du projet, c'est-à-dire de veiller à sa bonne réalisation dans les objectifs de technique, de coût et de délai.* ». Elle ajoute aussi « *Souvent le maître d'ouvrage identifie, en son sein, un interlocuteur au chef de projet du maître d'œuvre. Cet interlocuteur est aussi appelé chef de projet* ».

2.9 - **QU'EST-CE QUE LE MANAGEMENT DE PROJET ?**

L'objectif du management de projet est d'atteindre les objectifs du projet sous les contraintes cout, temps, qualité et avec des ressources limitées.
Pour y faire, la gestion de projet a pour mission l'organisation et la surveillance du travail, la gestion des ressources et l'animation et la motivation d'un groupe de personnels pour atteindre ces objectifs.
Le management de projet comprend trois principaux aspects : la gestion du temps, ressources et la production (Figure 7.). Ces activités sont fortement liées et cette liaison est permanente.

2.10 - **Les différents types de gestion[12]**

La Gestion de délai : elle consiste à déterminer un parcours qu'on va suivre, un calendrier de réalisation et une maîtrise d'enveloppe temps.

La Gestion de ressources : les moyens constituent le budget du projet, donc il s'agit de transformer le budget en travail.

La Gestion des productions : l'objectif d'un projet doit à son terme être concrétisé par une ou plusieurs fournitures. Il faut s'assurer que ce qui est produit se rapproche du but final.
Remarque : la solidarité entre les sommets du triangle de gestion doit être permanente.

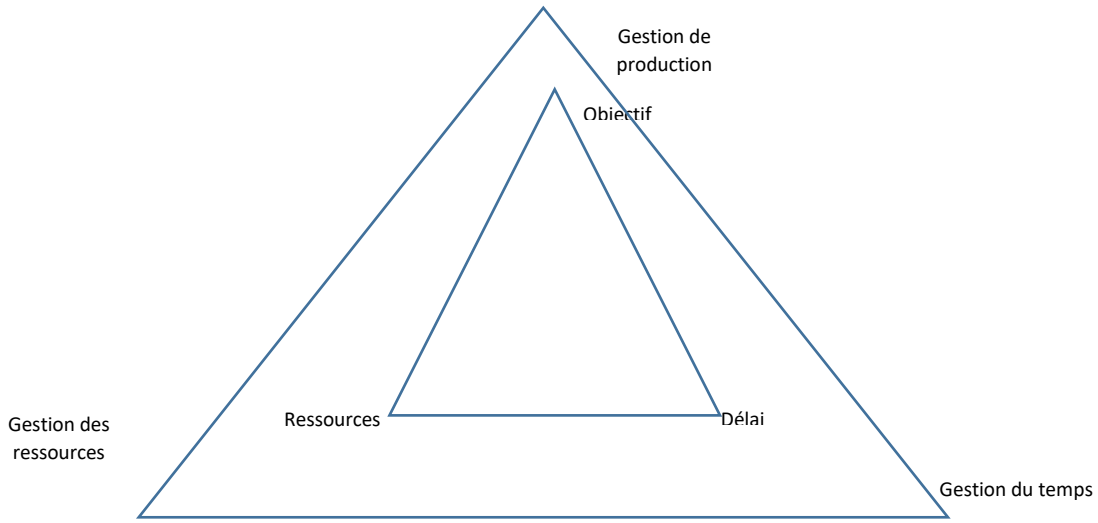


Figure 9 Le triangle gestion de projet[12]

2.11 - **Les définitions normalisées du management de projet**

Selon L'AFITEP et l'AFNOR Le management est défini comme « *l'ensemble des tâches permettant de conduire une opération quelconque à bonne fin* ».

Le référentiel de l'IPMA « Le management de projet consiste à planifier, organiser, suivre et maîtriser tous les aspects d'un projet, ainsi que la motivation de tous ceux qui sont impliqués dans le projet, de façon à atteindre les objectifs de façon sûre et dans les critères définis de coûts, délais et performance. Cela inclut les tâches de direction nécessaires aux performances du projet. ».

La gestion de projet est définit par le PMBOK comme :"the application of knowledge, skills, tools, and techniques to project activities to meet project requirements." (PMBOK)

2.12 - **Les activités de la gestion de projet**

Le projet passe par les étapes suivantes :

2.12.1 - **Initialisation**

Cette activité définit et démarre officiellement le projet. Elle identifie les contraintes financières, humaines, temporelles, techniques…etc et pose une démarche générale de conduite de projet. Dans le cas de la première phase du projet, elle désigne le chef de projet.

Le projet est lancé en signant un document nommé *project charter*.(une charte de projet). Ce document contient des items tels que l'objectif du projet, la description du produit(s) et les jalons[14].

2.12.2 - **Planification**

Dans cette activité, le chef de projet et son équipe développent le contenu du projet, définissent et raffinent les objectifs du projet ainsi que les activités nécessaires pour les réalisés. Le résultat final est un planning de travail, avec une répartition des tâches et des ressources[14].

2.12.3 - **Exécution**

Cette activité assure le bon déroulement du travail en coordonnant le personnel et les autres ressources dans la réalisation du travail planifié. Les objectifs de cette activité sont :

1. La coordination des ressources (budget, membres de l'équipe) pour réaliser les activités de projet.
2. La gestion et l'intégration des activités réalisées.
3. Assurer l'implémentation du contenu de projet et les changements approuvés.[14]

2.12.4 - **Suivi &contrôle**

Déclenché périodiquement pour mesurer l'état d 'avancement du projet par rapport au planifié, pour prendre des mesures correctives en cas de dérapage et pour traiter les demandes de changement [14].

2.12.5 - **Clôture**

Cette activité officialise l'achèvement d'une phase ou projet et capitalise l'expérience.

2.13 - **Exercices**

Exercice 2.1

Parmi ces scénarios, quels sont les scénarios de type projet et les scénarios de type opérations

1. Etendre ta maison
2. Tricoter une écharpe
3. Rangement des livres dans une bibliothèque.
4. Construire une cage pour oiseaux
5. Changer le filtre à air de votre voiture chaque 4 mois.
6. Développer une application web
7. Assemblage d'un produit
8. Arroser les plantes d'intérieur 2 fois par semaine.
9. Organisation d'une conférence.
10. Aller à la salle de gym 4 fois par semaine.
11. Renouvellement du parc informatique d'une entreprise pour qu'il soit conforme avec les technologies actuelles.
12. Rédaction d'un rapport

Exercice2.2 (http://www.exam-pm.com/preparation-examen-ipma-serie-12/)

Cochez la bonne réponse

1. Au sens du management de projet on appelle ouvrage :(

A. L'ensemble des actions réalisées au cours d'un projet

B.Un élément matériel du projet

C. L'objet physique ou intellectuel du projet

D. La méthode utilisée pour réaliser le projet

2. Les parties prenantes d'un projet sont :

A. Les seuls membres de l'équipe projet

B. Les futurs utilisateurs du projet

C. Des personnes ou groupes de personnes qui participent au projet, ont un intérêt dans les performances du projet, ou sont impactés par le projet.

3. Dans un projet, la définition des phases est :

A. Toujours la même

B. Liée à la nature du projet

C. Peut être modifiée en cours de projet

D. Décidée par le client

4. **Une bonne définition du contenu d'un projet est particulièrement utile pour :**

A. Le maître d'ouvrage

B. Le maître d'œuvre

C. Les deux

5. **Les critères de succès d'un projet sont :**

A. Dépendants de l'impact des changements dans l'environnement du projet

B. Fixés une fois pour toutes en début de projet

C. Uniquement dépendants des clauses contractuelles

6. **Le succès d'un projet s'évalue avant tout par :**

A. le respect des coûts et des délais

B. l'acceptation du produit par le commanditaire

C. la satisfaction optimale des parties prenante

Chapitre 3 - **Estimation de la charge**

3.1 - Introduction

L'estimation de la charge (effort) est une activité fondamentale dans le processus d'ingénierie logicielle. Elle est conduite pour peser le poids du projet, une phase ou une activité en termes de budget à dépenser et le temps requis pour les réaliser. Ce chapitre présente en détail les techniques couramment utilisées pour réaliser cette activité.

3.2 - LE MANAGEMENT DES PROJETS SYSTÈME D'INFORMATION

3.2.1 - Caractéristiques d'un projet système d'information

Un système d'information est un : « Ensemble organisé de ressources : matériel, logiciel, personnel, données, procédures… permettant d'acquérir, de traiter, stocker, communiquer des informations (sous formes données, textes, images, sons…) dans des organisations. ». Un système d'information est complexe et avec un caractère immatériel.

3.3 - Estimation des charges

L'objectif de l'activité d'estimation des charges est de déterminer le nombre de ressources à utiliser pour réaliser le projet.

La *charge* représente une quantité de travail nécessaire pour réaliser le travail. Cette quantité est indépendante du nombre de personnes qui vont réaliser ce travail.

Elle est mesurée en nombre de jour-personne, mois-personne…

Un mois-personne représente l'équivalent du travail d'une personne pendant un mois, en général 20 jours.

Ainsi, un projet de 40 mois-personne représente l'équivalent du travail d'une personne pendant 40 mois. Si on évalue le coût complet du mois-personne à 50 00DA en moyenne, le projet sera estimé à 200000DA.

La taille des projets est liée à sa charge. Les standards qualifiés un projet :

• Un très petit projet si la charge est inférieure à 6 mois-personne ;

• Un petit projet si la charge est comprise entre 6 et 12 mois-personne;
• Un projet moyen si la charge est comprise entre 12 et 30 mois-personne;
• Un grand projet si la charge est comprise entre 30 et 100 mois-personne;
• Un très grand projet si la charge est supérieure à 100 mois-personne.
Durée : dépend de la charge et du nombre de personnes infectées.
Exemple : 60 M/h peut être 1 personne pendant 5 ans ou 10 personnes pendant 6 mois ou 60 personnes pendant 1 mois.
Selon Brooks : *« L'homme-mois comme unité pour mesurer la taille d'un travail est un mythe dangereux et trompeur. Il implique que les hommes et les mois sont interchangeables. Les hommes et les mois sont des biens interchangeables seulement lorsqu'une tâche peut être partitionnée entre plusieurs employés sans qu'il faille une communication entre eux »*

3.4 - **Les différents Niveaux d'estimation**

L'estimation de la charge peut s'effectuer à différents *niveaux [12] à savoir :*
Au *niveau projet[12]* : à ce niveau on veut avoir une estimation de la charge du projet complet.
Avec cette estimation on peut :
• déterminer le budget du projet ;
• avoir une idée sur le poids de projet de point de vue effort ;
• estimer la rentabilité de l'investissement ;
• estimer la durée de projet.

Au *niveau phase*[12] : on veut avoir une estimation de la charge d'une étape spécifique.
Les objectifs d'estimation à ce niveau sont :
• *Ajuster le découpage*. En fonction de la charge on peut découper ou fusionner des phases. Par exemple on peut diviser la phase en deux lots et les gérés comme deux sous-projets.
• *Sous-traiter*. Affecter le travail à un sous-traitant.
• *Prévoir des délais* afin d'ordonnancer les étapes, ainsi que d'autres opérations complémentaires telle que la formation des utilisateurs.
• *Prévoir des ressources*, pour planifier l'affectation d'intervenants (internes ou externes) sur le projet.
Au *niveau étape* (*ou activité*)[12] ici:
• on peut faire une planification précise ;
• construire un calendrier de remise des différents résultats intermédiaires.

• prévoir et effectuer un suivi du projet ou sous-projet, pour surveiller les écarts ;
• prévoir l'affectation des ressources, car il peut y avoir une montée en charge ou une diminution d'une étape à l'autre.
Au *niveau tâche[12] :* il s'agit d'évaluer chacune des tâches qui font généralement l'objet d'une affectation individuelle, par exemple la tâche d'élaboration du jeu d'essai Planification de production. L'ordre de grandeur est le jour-personne.
L'estimation permet une planification au niveau le plus fin, qui est indispensable
pour le suivi du travail de l'équipe.

Entre le niveau du projet et celui de la tâche, la visibilité est croissante, ce qui fournit des éléments de plus en plus précis pour l'évaluation. C'est pourquoi on utilisera des techniques différentes, selon le niveau où se situe le besoin d'estimation[12].

3.5 - Les méthodes d'estimation

3.5.1 - **La non méthode**

Exemple : offrir un prix bas pour être sûr de l'avoir, mais sans être sûr d'y gagner quelque chose en définitive (de point de vue financier).

3.5.2 - **La méthode Delphi**

Cette méthode fait appel à l'expérience des experts du domaine. C'est une méthode proposée en 1948 par la Rand Corporation,
On cherche une estimation qui est le résultat de la convergence de plusieurs avis d'experts selon la démarche suivante *:*

- Chaque expert propose une estimation basée sur son expérience.
- La publication des résultats
- Les experts ajustent ou maintiennent leurs propositions sur la base des arguments données.
- Republication des nouveaux résultats.
- La troisième et la quatrième étape jusqu'à l'arrivé à une convergence, un consensus sur une estimation ou après une analyse de disparité, la moyenne est prise comme estimation.

3.5.3 - **La méthode de répartitions proportionnelle**

Elle s'appuie sur le découpage du projet en différentes phases. On commence par faire L'estimation de la charge globale. Ensuite, on détermine la charge pour chaque phase du cycle de vie.

Etape	Ratio
Etude préalable	10 % de la charge totale
Etude détaillée	20 à 30 % de la charge totale
Etude technique	5 à 15 % de la charge "réalisation"
Réalisation	2 fois la charge "étude détaillé"
Mise en œuvre	30 à 40 % de la charge "réalisation"

Figure 10 Méthode de répartition de charges

3.5.4 - **La méthode COCOMO**

Une des méthodes les plus anciennes, la méthode COCOMO (*COnstructive COst MOdel*), proposée par B. W. Boehm en 1981 pour estimer la charge des applications écrites en COBOL[13]. COCOMO est une méthode d'estimation paramétrique *basée sur le* nombre de milliers d'instructions sources livrées (KISL). Plus précisément Barry Bohm a étudié plusieurs projets pour dériver un modèle mathématique (formule) qui décrit la relation entre la taille d'une application mesurée en KLOC et sa charge. Formellement : trouver les valeurs des paramètres a,b et c tel que charge=a.$(KLOC)^b$+c. Barry Bohm a étudié aussi la relation entre le délai d'un projet et sa charge. Formellement délai=a1.$(charge)^{b1}$+c1.

Sur la base du délai trouvé, on peut déterminer la taille moyenne de l'équipe qui est égale à charge/délai.

L'approche COCOMO est basée sur les hypothèses suivantes :

- Il est facile à un informaticien d'estimé le nombre de lignes source.
- La complexité d'écriture d'un programme est la même quel que soit le langage de programmation.

L'estimation du nombre des lignes de code peut être faite :

- En se basant sur l'expérience ;

- En se basant sur la taille des anciens projets ;
- La taille des solutions des concurrents ;
- Ou par décomposition de projet en parties plus petites et en calculant une estimation sur la base des estimations de ces parties.

  Une approche standard utilisée pour estimer la taille d'une partie est d'utiliser trois valeurs d'estimations :

  -la taille minimale possible de la partie(mini), La taille maximale possible de la partie (maxi)et une estimation de la taille la plus probable(pbi).

On prend alors comme estimation la valeur calculée suivante :

moyenne = (maxi + pbi + mini)/3

Souvent, notamment si l'on a un ensemble d'estimations qui se répartissent autour d'une valeur la plus probable, on prend une moyenne pondérée :

moyenne = (maxi + 4.pbi + mini)/6

Pour trouver un bon modèle pour estimer la charge et le délai de réalisation du projet, l'étude a divisé les projets étudiés en trois catégories :

1. Application (separate, organic) : moins de 50 000 instructions e.g., data processing, scientific)
2. Utilitaire (semidetached) : entre 50 000 et 300 000 instructions. e.g., compilers, linkers, analyzers
3. Système (embedded) : plus de 300 000.

Les modèles trouvés pour chaque catégorie de projet sont :

Type de projet	Charge en **mois-personne**	Délai en mois
Application	$2.4(KLOC)^{1.05}$	$2.5(charge)^{0.38}$
Utilitaire	$3.0*(KLOC)^{1.12}$	$2.5(charge)^{0.35}$
Système	$3.6*(KLOC)^{1.20}$	$2.5(charge)^{0.32}$

Figure 11 Formules du modèle COCOMO.

Par exemple un projet de type application et de taille 5K aura une charge de 13p-mois et prend 6.36 mois et la taille moyenne de l'équipe est $13/6.36 \cong 2$

3.5.5 - **Modèle COCOMO intermédiaire**

Une estimation brute (nominale) est modifiée selon le degré d'influence de 15 facteurs de productivité.

Ces facteurs représentent un avis subjectif sur le produit, le matériel, le personnel, et les attributs du projet. Chaque facteur prend une valeur nominative de 1, et peut varier selon son importance dans le projet. Le degré d'influence est mesuré sur une échelle nominale (très bas, bas, nominal, élevé, très élevé, très très élevé).

Facteurs de productivité

- Logiciel
- *RELY*: Fiabilité requise
- *DATA*: Volume des données manipulées
- *CPLX*: Complexité du produit
- Matériel
- *TIME*: Contraintes de temps d'exécution
- *STOR*: Contraintes de taille mémoire
-*VIRT*: Instabilité de la mémoire
- Personnel
- *ECAP*: Aptitude de l'équipe
- *AEXP*: Expérience du domaine
- *VEXP*: Expérience de la machine virtuelle
- *LEXP*: Maîtrise du langage
- Proj - Compétence de l'analyste
- *MODP*: Pratique de développement évoluées
- *TOOL*: Utilisation d'outils logiciels
- *SCED*: Contraintes de délais

Multiplicateurs	Très bas	Bas	Nominal	Elevé	Très élevé	Extrêmement élev
RELY	0,75	0,88	1	1,15	1,4	
DATA		0,94	1	1,08	1,16	
CPLX	0,7	0,85	1	1,15	1,3	1,65
TIME			1	1,11	1,3	1,66
STOR			1	1,06	1,21	1,56
VIRT		0,87	1	1,15	1,3	
ECAP	1,44	1,18	1	0,86	0,7	
AEXP	1,29	1,13	1	0,91	0,82	
VEXP	1,21	1,1	1	0,9		
LEXP	1,14	1,07	1	0,95		
MODP	1,24	1,1	1	0,91	0,82	
TOOL	1,24	1,1	1	0,91	0,83	
SCED	1,23	1,08	1	1,04	1,1	

Table 2. Coefficients multiplicateurs des facteurs de productivité

3.4.5.1 Calcul de la charge (HM) dans le modèle intermédiaire

On commence par l'estimation de nombre d'instruction source livrées (en KDSI), puis on calcule le nombre d'homme*mois par la formule appropriée du tableau ci-dessous. On obtient la charge basique HM (charge nominale).

On estime ensuite les 15 facteurs de productivité et on calcule le facteur d'ajustement (a) en multipliant les coefficients de ces facteurs.

En fin on multiplie l'effort 'nominal' par le facteur d'ajustement pour trouver la charge finale : HM = HM base * a.

3.5.6 - **La méthode de points de fonction**

Proposée par Alan Albrecht d'IBM en 1979 et devenu un standard à l'IFPUG (*International Function Point Users Group*) et à la FFPUG (*French FunctionPoint Users*).

Le principe de la méthode est d'identifier et de quantifier les fonctionnalités à développer. L'identification est faite à partir de la description du futur système dictée par le cahier de charge ou les documents décrivant les besoins du projet.

La méthode classifie les fonctionnalités en cinq types de fonctionnalités avec trois degrés de complexité simple, moyenne et complexe.

3.5.7 - **Principe de calcul des points de fonction**

L'analyste identifie chaque instance de chaque fonctionnalité type (entrée, sortie, interrogation, fichier interne, fichier externe) offerte par le système. Chaque instance est classifiée comme étant de complexité faible, moyenne ou élevée. Le nombre de fonctionnalité type de chaque classe de complexité est multiplié par un coefficient (reflétant le degré de complexité) pour trouver la valeur PF de la fonctionnalité. Les PFs de tous les types de fonctionnalités sont sommés pour trouver le nombre de PF de tout le système. Ce dernier nombre est transformé en charge.

3.5.8 - **Les types de fonctionnalités**

Entrée (ENT) : une fonction élémentaire qui permet d'introduire des données à l'intérieur du domaine. Les écrans, boites de dialogue, messages à travers lesquels un utilisateur entre une donnée font partie des entrées externes.

Sortie (SORT) : est une fonction élémentaire, significative pour l'utilisateur, qui envoie des données vers l'extérieur du domaine. Elles peuvent prendre la forme de fichiers de sortie envoyés à d'autres application, d'écrans, de rapports et de de graphes destinés à l'utilisateur final.

Interrogation (INT) : fonction de type question/réponse. Son objectif est l'extraction de données. Le résultat de l'interrogation ne contient aucune donnée calculée ou dérivée. La frontière entre requête externe et sortie est assez floue. Nous dirons que les requêtes concernent exclusivement les accès à une base de données alors que les sorties combinent ces accès avec des calculs et traitements plus ou moins complexes.

Fichier interne (GDI : groupe de données interne) : fichier logique un groupe de données que l'utilisateur perçoit comme logiquement liées. Par exemple un fichier, une table dans une base de données

Interface externe (gde groupe de données externe) : fichier logique créer et maintenue par d'autre domaine et utilisée par notre application.

Après identification des fonctionnalités, elles sont classifiées en simple moyenne et complexe.

3.4.8.1 La complexité et le nombre de points de fonction

Après que les instances de chaque fonctionnalité type aient été ainsi classifiées, chaque instance est affectée d'un score : faible, moyenne, élevée. Pour les transactions (entrées, sorties et interrogations) le score est calculé sur le nombre de fichiers mis à jour ou référencés et le nombre de types de données impliquées. Voir tableaux ci- dessous. Dans ces tableaux S désigne faible, M désigne moyenne et C désigne haute.

Type de fonctionnalité	Multiplicateur		
	Faible	Moyen	Haut
Entrée	3	4	6
Sortie	4	5	7
Fichier	7	10	15
Interface	5	7	10
Interrogation	3	4	6

Table 3. Complexité des multiplicateurs

Nombre des types enregistrements	Nombres de type de données		
	<20	20-50	>5
1	S	S	M
2-5	S	M	C
>5	M	C	C

Table 4. Complexité IFPUG de Fichier

Nombre des types enregistrements	Nombres de type de données		
	<20	5-15	>15
1	S	S	M
2-5	S	M	C
>5	M	C	C

Table 5. Complexité IFPUG d'une entrée

Nombre des types enregistrements	Nombres de type de données		
	<6	6-19	>19
0-1	S	S	M
2-3	S	M	C
>3	M	C	C

Table 6. Complexité IFPUG d'une sortie

Une requête est notée (faible, moyenne ou haute) comme une sortie mais on lui attribue plutôt une valeur comme à une entrée externe.

Exemple

La table suivante présente le nombre de points fonction d'un système composé de 5 entrées, 4 sorties, 2 interrogations, 1e interface et 4 fichiers.

	Simple	Moyenne	Complexe	PFC
Entrées	3	2		3*3+4*2
Sorties		4		4*5
Interrogations			2	6*2
Interfaces	1			1*7
Fichiers		3	1	3*7+1*15
				PFB=92

Table 7. Exemple de calcul de la charge en PF d'un système.

3.4.8.2 La transformation du nombre de points de fonctionnels en charge

Il n'existe pas une règle formelle pour trouver le coefficient de transformation. Ce coefficient est lié à l'environnement matériel et humain. Il est préférable, pour une entreprise, de déterminer ses propres coefficients à partir de sa propre base de projets.
Par exemple si 1 PF=2p-m alors la charge du projet présenté par la table 6 est 92*2=184 p-m.

3.6 - **Exercices**

Exercice 3.1

En appliquant la méthode COCOMO, estimer la taille moyenne de l'équipe qui faudrait prévoir pour développer un logiciel estimé à environ 30 000 instructions sources, le projet est simple et l'équipe du développement est relativement réduite.
Exercice 3.2

```
import java.net.*;
import java.io.*;

/**
 * This program is a socket client application that connects to a time server
 * to get the current date time.
 *
 * @author www.codejava.net
 */
public class TimeClient {

    public static void main(String[] args) {
        String hostname = "time.nist.gov";
        int port = 13;

        try (Socket socket = new Socket(hostname, port)) {

            InputStream input = socket.getInputStream();
            InputStreamReader reader = new InputStreamReader(input);
```

```
            int character;
            StringBuilder data = new StringBuilder();
            while ((character = reader.read()) != -1) {
               data.append((char) character);
            }
            System.out.println(data);
        } catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Calculez LOC, NCLOC, CLOC et la densité de commentaire de ce programme

Exercice 3. 3

1. Calculer la charge fonctionnelle en PF du programme suivant:
   Le programme prend un entier saisi au clavier, affiche son successeur sur écran et l'enregistre (le successeur) dans un fichier.
   On suppose que toutes les fonctionnalités sont simples et de poids=4.

Exercice 3.4 [13]

Un dentiste souhaite développer un logiciel de gestion des rendez-vous.

Quand un patient téléphone pour prendre un rendez vous, le réceptionniste vérifie le calendrier et programme le patient le plus tôt possible. Si le patient est satisfait par la date proposée alors le réceptionniste enregistre le RDV et le motif de la visite. Le système vérifiera le nom du patient et extrait les autres détails concernant le patient (ID du patient, adresse, date naissance) à partir des enregistrements des patients.

Après chaque visite l'hygiéniste ou l'assistant marque le RDV comme terminé, ajoute des commentaires et le cas échéant programme le rendez-vous suivant du patient.

Le système doit permettre à ses utilisateurs d'interroger les données des patients par nom et par date. Les résultats de l'interrogation (détails de patient et la date de RDV) seront affichés sur écran. Le réceptionniste peut annuler des RDVs. Il peut aussi imprimer une liste des prochains RDVs pour rappeler les patients deux jours d'avance. Le système extrait les numéros de téléphones à partir des enregistrements des patients. Le réceptionniste doit avoir la possibilité d'imprimer des plans de travail hebdomadaire et mensuel.

Questions :

1. Identifier les différents types de fonctionnalités.
2. Si on suppose que toutes les entrées et les interrogations ont une complexité simple, les fichiers internes et les fichiers externes ont une complexité moyenne et les sorties sont complexes. Calculer les points fonctionnels du projet (PFs).
3. Si la charge de 1 PF= 1 jour/personne, calculer la charge en mois-personne(1 mois =24 jours)

Exercise 3.5

4. Trouver le modèle d'estimation de la charge à partir des données historiques du tableau ci-dessous :
5. 1 ISL : instruction source livrée =instruction source livrée
6.

Taille (ISL)	Charge en heure-personne
20	85
30	125
60	200
80	325
120	485
300	124

Exercice 3.6

Spécification du problème

On souhaite gérer les emprunts d'ouvrages d'un fond de bibliothèque.

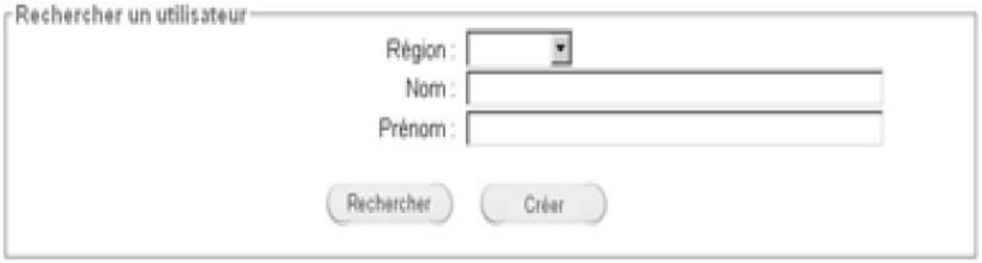
1. Le bibliothécaire est un employé de la bibliothèque. Le système à concevoir doit assister le Bibliothécaire dans sa tâche.
2. Le bibliothécaire communique avec les emprunteurs.
3. Le bibliothécaire prête des livres à des emprunteurs.
4. Il doit gérer l'achat de nouveaux titres.
5. Les titres les plus demandés sont achetés en plusieurs exemplaires.
6. Les vieux livres sont retirés lors qu'ils ne sont plus demandés depuis longtemps ou en mauvais état.
7. Un emprunteur peut réserver un livre qui est indisponible (déjà prêté ou répertorié mais non encore acheté).
8. Lorsqu'un livre devient disponible (rendu ou acheté), un emprunteur qui l'avait réservé doit être averti.
9. La réservation est annulée quand le livre est emprunté.
10. Une réservation peut être annulée à tout moment.
11. La mise à jour (ajout, suppression et modification) des informations relatives aux exemplaires, emprunteurs et fond bibliothèque doit être aisée.
12. De même que la mise à jour des informations relatives aux prêts et réservations.
13. Le bibliothécaire peut obtenir la liste des livres empruntés dont la date de retour est dépassée.
14. Limitation : le système ne gère pas les messages aux emprunteurs.

q)

1. Estimer la charge fonctionnelle en PFs.
2. Déterminer la complexité de chaque fonctionnalité.

Exercice 3.7

Calculer la charge fonctionnelle de l'application suivante en utilisant la méthode des points de fonction (toutes les fonctions sont simples et le poids =4) :



Chapitre 4 - **Gestion de contenu et planification**

4.1 - **Introduction**

La planification est la plus importante activité dans la gestion de projet. Ce processus répond au fameux «qcqqc» : quoi ? (Objectif) comment ? (Plan d'action) quand ? (Calendrier) qui ? (Ressources) combien ? (Budget). Cette activité crée un planning à partir du contenu détaillé du projet. Ce chapitre commence par la présentation du processus de collecte des besoins (comment détailler le contenu du projet). Puis il présente les outils utilisés pour créer un planning de projet tels que la WBS, la PBS, PERT et GANT.

4.2 - **La Collection des besoins**

L'objectif est de définir et documenter les caractéristiques et les fonctions fournies par les produits résultats de projet.
The PMBOK® Guide,5 Edition, définit les besoins comme les "conditions or capabilities that must be met by the project or present in the product, service, or result to satisfy an agreement or other formally imposed specification."

Une mauvaise collection et définition des besoins conduira à un travail supplémentaire qui peut consommer plus que le ½ du coût de projet surtout dans le cas des projets de développement de logiciels. Selon la figure 9 Une erreur aura un impact (coût) moins cher si elle est détectée et fixée au niveau de la phase de l'analyse des besoins par rapport à une détection tardive dans d'autres phases.
Les techniques utilisées pour collecter les besoins sont : les interviews des parties prenantes, les questionnaires, les enquêtes et l'observation. En cas de développement d'un logiciel, le prototypage et l'analyse des documents sont des outils efficaces pour collecter les besoins. Le banchmarking est aussi une autre technique qui consiste à générer des idées en comparant le projet actuel à des projets similaires. Les projets comparés sont soit interne soit externe.

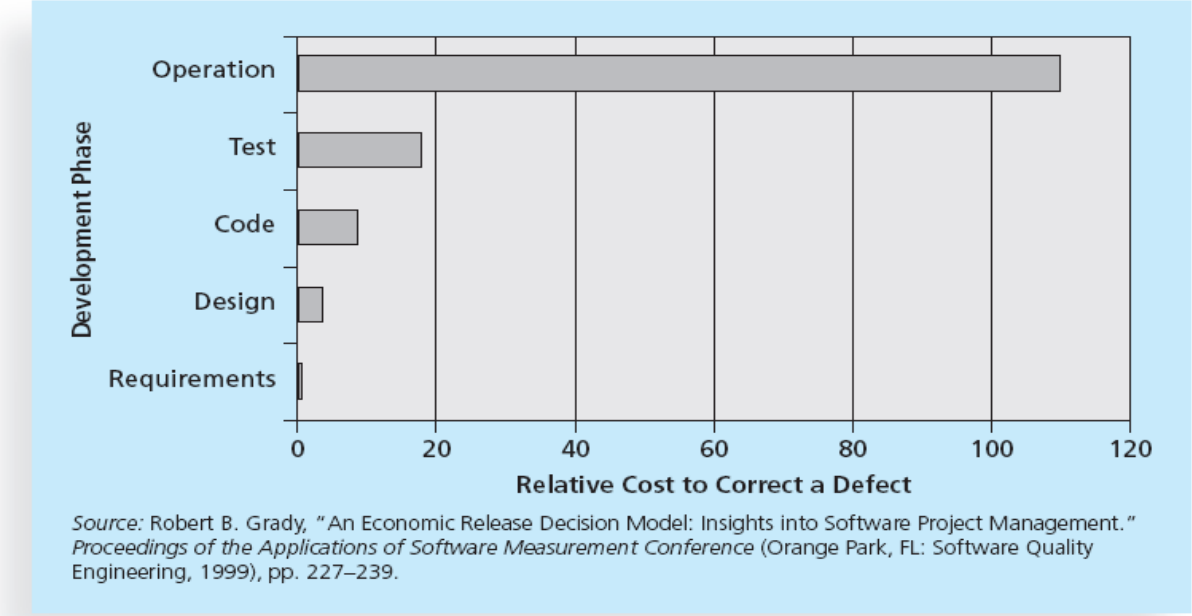


Figure 12 Le coût de correction d'une erreur.

4.3 - **A requirements traceability matrix (RTM)**

Plus la documentation qui décrit les besoins. Le groupe crée une RTM qui est une table qui recense tous les besoins, leurs attributs et leurs états (exemple Figure 10).

Besoin N°	Nom	Type	Source	Etat
B31	Mémoire laptop	Matériel	Spécifications du laptop	Terminé

Figure 13.Exemple d'une entrée de la matrice RTM.

4.4 - **Planification du projet**

La planification est la plus importante activité dans la gestion de projet. L'atteinte des objectifs du projet passe surement par la création d'un bon plan. La planification requiert l'identification des tâches (activités) à réaliser, dans quel ordre seront effectués et quels sont les ressources nécessaires à leurs exécutions. Ce processus répond au fameux «qcqqc» : quoi ? (Objectif) comment ? (Plan d'action) quand? (Calendrier) qui ? (Ressources) combien? (budget)0.

4.4.1 - **Découpage du projet**

Le découpage de projet a pour objectif la répartition de la production et les ressources[12].
Le découpage est basé à la fois sur une approche cartésienne de réduction de la difficulté et sur l'approche systémique de prise en compte des liens entre les Eléments. Le résultat de découpage de projet est l'identification des sous-ensembles quasi autonomes, Les caractéristiques de ces sous –ensembles sont[12] :

• chaque sous-ensemble du projet donne lieu à un résultat bien identifié ;

• la charge de chaque sous ensemble est évaluable ;

• les dépendances entre les sous-ensembles sont repérables : certains sous-ensembles peuvent être réalisés parallèlement, d'autres sont liés entre eux par des contraintes d'antériorité ;

• le découpage est fait d'une manière récursive, un sous-ensemble peut aussi être décomposé.

Deux critères sont souvent utilisés pour découper un projet[12] :

Le *critère temporel* est utilisé dans la plupart des projets. Il permet la répartition du travail dans le temps : la décomposition montre la succession d'étapes et de phases à exécuter. À chacune, on affecte une date de début prévue et une date de fin visée.

Le *critère structurel* : organisation du travail en se basant sur la structure du produit final : la décomposition fait apparaitre les différents composants (modules) qu'il faut obtenir.

4.4.2 - **Un jalon (millestone)**

Un événement significatif qui marque le début, la fin d'une étape importante ou la réalisation d'un livrable. Ce sont des activités à durée nulle. Exemple la fin de la conception. Ces événements aident dans le suivi et le contrôle de la progression du projet.

4.4.3 - **Un livrable**

Il est généralement le résultat de la réalisation d'une partie de projet. Il est interne ou externe.

Un livrable externe est un résultat ou produit à livrer au maitre d'ouvre tandis qu'un livrable interne est un résultat ou produit à vérifier et à valider en interne.

Les jalons se matérialisent par :

Des réunion, Des décisions de type go/no go ..etc

4.5 - LES DÉCOUPAGES NORMALISÉS

Les normes internationales proposent trois découpages : PBS, WBS et OBS.

Le PBS est parfois appelé « structure du produit » ou « arborescence produit »,
« *Représentation des liens de composition entre les divers constituants d'un produit complexe* » [AFITEP, 2000].

Le découpage WBS, *Work Breakdown Structure* (structure de décomposition du travail), représente, sous forme d'une arborescence, les différents lots de travaux (tâches) nécessaires pour parvenir aux délivrables décrit dans le PBS. Il est basé sur le critère structurel et sur le critère temporel. La WBS est la réponse aux deux questions : « Que doit-on faire ? Comment doit-on s'y prendre ? ».
L'AFITEP et l'AFNOR traduit la WBS en Organigramme des tâches et le définit comme le « découpage hiérarchisé et arborescent du processus de réalisation en éléments plus faciles à analyser et à maîtriser, appelés lots de travaux ou tâches ».
Le premier niveau de WBS correspond généralement au cycle de vie utiliser par l'entreprise.

Un bon WBS respecte les règles suivantes [13]:

1.Un WBS est un arbre

2. Les tâches (lot de travaux) et les descriptions des livrables doivent être compréhensibles et non ambiguës.

3. Chaque tâche doit avoir un critère de terminaison (généralement un livrable) par exemple : un document d'analyse.

4. Tous les livrables doivent être identifiés.

5. La terminaison des sous tâches implique la terminaison de la tâche mère.

Exemple :

Projet de fin d'étude : un mémoire et une application web pour gérer le stock.

La PBS est présentée par la figure 14. La WBS est présentée par la figure 15.

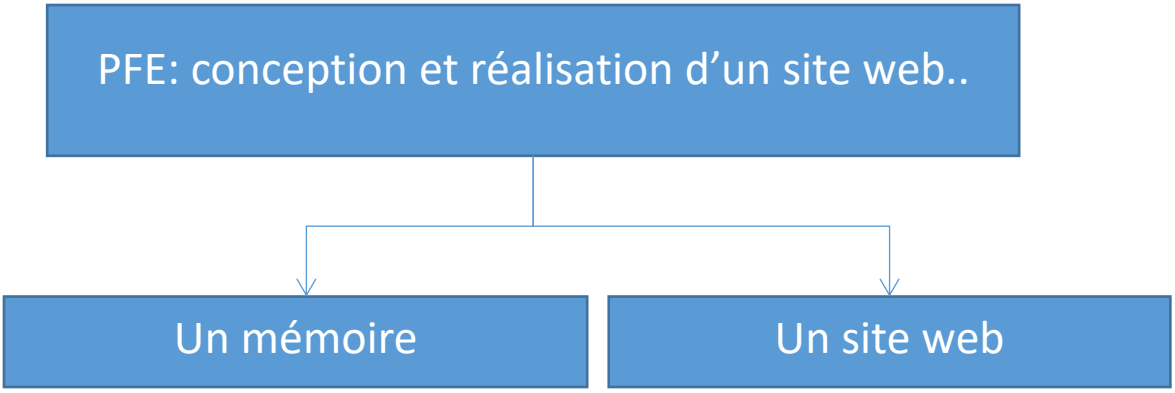


Figure 14 La PBS du projet de fin d'étude.

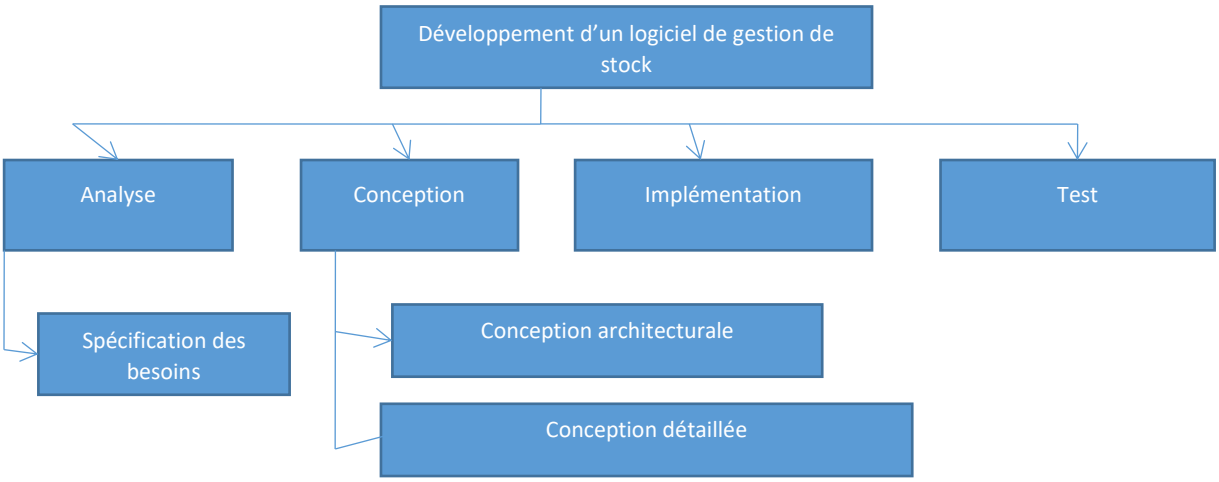


Figure 15 Exemple d'une WBS.

L'OBS, *Organisation Breakdown Structure* (structure de décomposition de l'organisation), est un WBS annoté avec les noms des personnes responsables de la production des différents éléments. Selon l'AFITEP *l'OBS est « la structure des différents niveaux de responsabilités de réalisation de l'ensemble des lots de travaux d'un même organigramme des tâches »* [AFITEP, 2000].

4.6 - **Décomposition des lots de travaux**

Ce processus prend en les lots des travaux figurant dans la WBS et les décomposent en activités simples et gérables. La décomposition est faite jusqu'à un niveau qui permet l'estimation de la durée et la charge, le suivi, le contrôle et la gestion. En pratique les WBSs sont construites jusqu'à ce niveau.

Le résultat final de ce processus est une liste des activités à réaliser plus les attributs et les détails qui montrent quand est-ce qu'une activité est réalisée. Le processus identifie aussi les jalons (millestones) à utiliser dans le projet.

4.7 - **Estimation de la durée des activités**

On utilise soit :

- **Estimation à un point** : l'estimateur donne une seule estimation.
- **Par analogie.**
- **Estimation paramétrique :** l'estimateur utilise les anciennes données pour produire une nouvelle estimation. Le résultat est généralement une relation entre le nombre de ligne et la durée et l'installation…. Deux techniques sont couramment utilisées pour trouver cette relation : regression analysis ou learning curve techniques.
- **Estimation à trois points :** Recherche d'une estimation sur la base de trois valeurs d'estimation :
    1. Une valeur optimiste (f), c'est-à-dire faible car on fait l'hypothèse qu'on ne rencontrera aucun problème (disponibilité des ressources, productivité, relations avec le client ou les utilisateurs…) ;
    2. Une valeur pessimiste (F), qui est élevée car on suppose que l'on va se heurter à de nombreuses difficultés ou imprévus ;
    3. Une valeur considérée comme la plus probable (p), avec une part vraisemblable de difficulté.

On prend alors comme estimation la valeur calculée suivante :
Moyenne = (f + F + p) /3, ou on prend une moyenne pondérée : moyenne = (f + F + 4p) /6.

4.8 - **Ordonnancement des activités**

En premier, on commence par l'identification des relations de dépendance entre les activités. En général les activités ne peuvent être effectuées toutes en parallèle et sont liées par une relation de dépendance qui représente le fait qu'une activité(s) ne peut (vent) commencée que si d'autres activités ont terminé. Par exemple le test ne peut commencer que si l'activité codification est terminée. Cette relation existe même dans le cas où on a une illimitée de ressources.

Les activités et les dépendances entre eux sont généralement représentées par un graphe orienté. Sur la base de ce graphe, on calcule un calendrier (échéancier de projet). Le calendrier montre les dates début et fin de chaque activité. Les marges des activités et le chemin critique de projet. Une estimation plus précise de la durée minimale et le cout minimal du projet est aussi calculable à partir de ce graphe. Une technique couramment utilisée pour trouver tous ces détails est la technique PERT. Il y'a d'autres techniques basées sur le même principe que PERT mais ne sont pas présentés dans ce cours par exemple la méthode Critical Path Method ou CPM.

4.8.1 - **PERT (Program Evaluation and Review Technique)**

Dans cette technique les activités et les dépendances sont représentées par un graphe orienté.

Chaque activité a une estimation de la durée de sa réalisation et une liste des activités qui lui sont dépendantes (les activités qui doivent être réalisées avant qu'elle puisse commencer).

Le graphe est utilisé pour calculer pour chaque activité :

- Sa date début au plus tôt et sa date début au plus tard,
- Sa date de fin au plus tôt et sa date de fin au plus tard,
- Sa marge.
- On calcule aussi le chemin critique du projet et la durée minimale du projet.

4.7.1.1 Dates de début au plus tôt et au plus tard

La date de « début_au_plus_tôt » d'une activité T indique, en fonction des liens définis dans le projet pour l'activité, la meilleure date de démarrage de l'activité.
La date de « début_au_plus_tard » pour cette même tâche indique la dernière date possible à laquelle peut démarrer la tâche afin de ne pas toucher à la date fine de projet.

4.7.1.2 Dates de fin au plus tôt et au plus tard

Ces deux dates indiquent les limites de fin d'une activité en fonction de l'ordonnancement et des contraintes associées.

4.7.1.3 Chemin critique

Le chemin critique d'un projet mène de début projet jusqu'à sa fin. Il est formé de l'ensemble des activités qui ont une marge égale à zéro. Il est le plus long chemin dans le graphe. Le moindre retard dans une de ces activités augmente la durée de projet (date fin de projet).

4.7.1.4 Marge

Les activités non critiques ont une certaine flexibilité en ce qui concerne leurs dates de démarrage c.-à-d. que ces activités peuvent être retardées sans que le projet soit en retard. La marge de chaque tâche est calculée sur la date au plus tôt ou au plus tard, de début ou de fin.

4.7.1.5 Algorithme pour calculer les Dates de début au plus tôt(ddt) et les dates de fin au plus tôt(dft)[13]

Répéter pour chaque activité, A, du projet
Si les prédécesseurs de A sont terminés alors
{
date de début au plus tôt (A)=Max(Dates de fin au plus tôt de ses prédécesseurs).
dates de fin au plus tôt(A)= Max (Dates de fin au plus tôt de ses prédécesseurs)+durée(A)
}
Fin .
La plus grande date de fin au plus tôt parmi toutes les dates de fin au plus tôt détermine la fin de projet.

4.7.1.6 Identification des activités critiques (chemin critique) [13]

1. Sélectionner l'activité(s) qui a la plus grande **dft** parmi toutes les activités du projet et marquer la comme critique.
2. Répéter
   Sélectionner parmi les prédécesseurs de l'activité(s) critique(s) les activités qui ont la plus grande dft et marquez-les comme critiques.
   Jusqu'à ce qu'a la sélection d'une activité de départ.
   Fin.

4.7.1.7 Calcul des Marges (calcul des dates début au plus tard(ddtr) et dates fin au plus tard(dftr)) [13]

Sélectionner une activité (s),A, non critique(s) qui a(ont) la plus grande dft.

Si ces activités n'ont pas de successeurs alors dftr=date fin du projet ;

sinon

dftr=min(dates début au plus tard des successeurs de A(activité en cours de traitement))

marge=dftr-dft ; ddtr=marge+ddt

Fin.

Par exemple, la table 7 [13]montre un exemple d'un projet, la figure 16 présente le réseau de PERT du projet et la table 8 présente les résultats de calculs des dates début, fin, le chemin critique et les marges

Tâche	Durée	Dépendances
a	8	
b	10	
c	8	a,b
d	9	a
e	5	b
f	3	c,d
g	2	d
h	4	f,g
i	3	e,f

Table 8. Un exemple d'un projet[13]

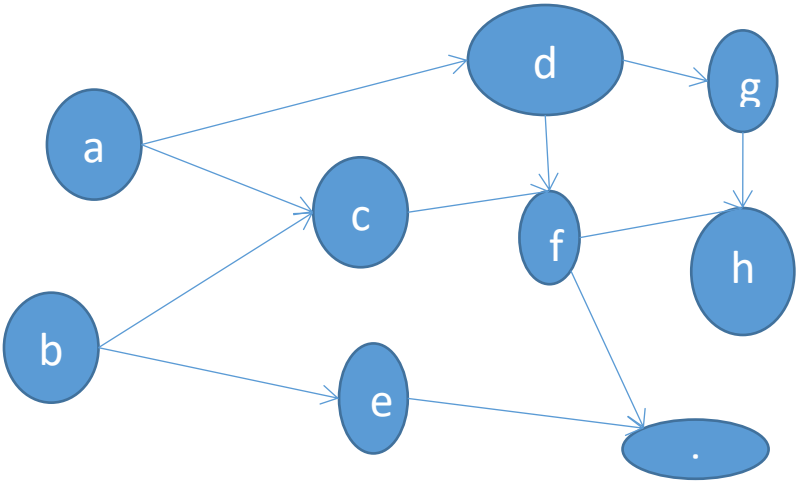


Figure 16 Réseau PERT du projet du table 7.

Tâche	Durée	Dépendances	Date début		Date fin		Critique ?
			Tôt	Tard	Tôt	Tard	
a	8		0	1	8	9	
b	10		0	0	10	10	*
c	8	a,b	10	10	18	18	*
d	9	a	8	9	17	18	
e	5	b	10	14	15	19	
f	3	c,d	18	18	21	21	*
g	2	d	17	19	19	21	
h	4	f,g	21	21	25	25	*
i	3	e,f	21	22	24	25	

Table 9. Résultats de calcul des ddt,ddtr,dftn dftr , marges et le chemin critique du projet de la table 7[13].

4.9 - **Autres modèles**

Extensions des réseaux PERT

– PERT Charge pour prendre en compte les ressources affectées au projet

– PERT Cost pour gérer les coûts.

4.10 - **LE GANTT du projet**

Le diagramme de Gantt est développé par GANTT pour représenter graphiquement le travail à effectuer dans un atelier. Il est utilisé pour représenter le planning(calendrier) du projet. Ce **diagramme** est une représentation dans le temps du travail à réaliser et des ressources utilisées. La construction du diagramme est réalisée comme suit (Figure 17):

• en abscisse, on a l'axe du temps ;

• en ordonnée, on peut avoir soit les tâches, soit les personnes affectées aux tâches.

Selon que l'on utilise ou non les marges pour effectuer la planification, on parlera de *planification au plus tôt* ou de *planification au plus tard*.

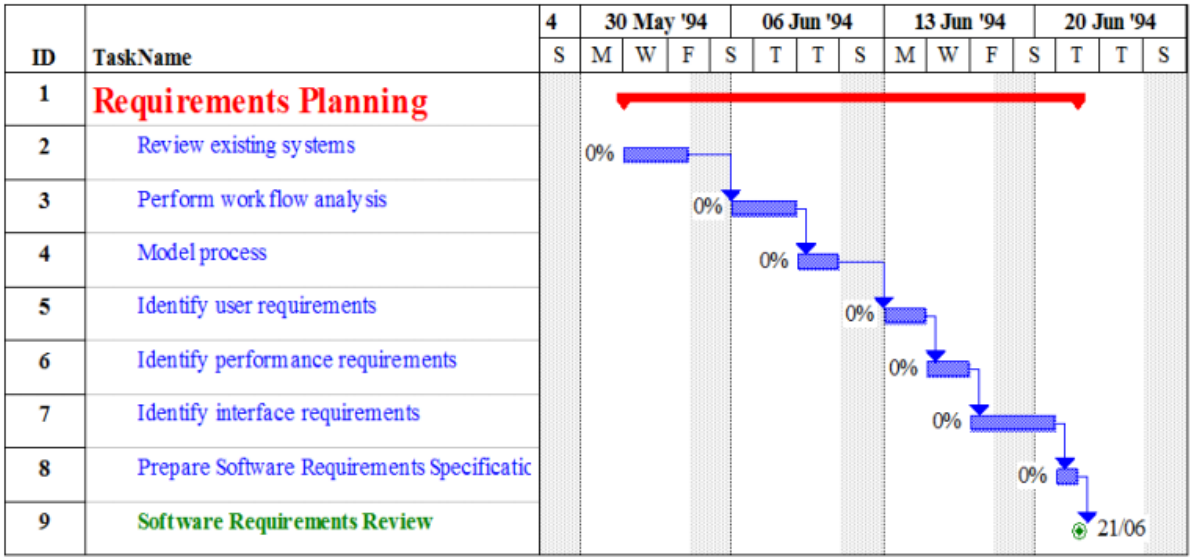


Figure 17 Exemple de diagramme de GANTT

4.11 - **Outils de planification**

1. Open Workbench : Open Workbench est une application libre de gestion et de planification de projet. Il s'agit d'un équivalent libre de Microsoft Project. Cette application permet de faire :

• découpage en activités (WBS - work breakdown structure).

• diagramme de GANTT du projet.

L'application est téléchargeable sur : http://www.openworkbench.org

2. GanttProject : GanttProject est un outil pour créer des diagrammes de Gantt et des reseaux PERT. Il est librement téléchargeable sur : http://www.ganttproject.biz/download

3. OpenProj : OpenProj est un remplaçant idéal de Microsoft Project pour créer des diagrammes Gantt et de PERT, et même pour calculer un prix et le profit. Il est disponible en téléchargement libre sur : http://openproj.org/

4.12 - **Exercices**

Exercice 4.1[20]

Classifier les éléments suivants (attributs du jeu « ninjago ») comme éléments de la PBS ou éléments de la WBS.

La programmation, 34 niveaux de jeu, conception graphique, 4 joueurs, bons éléments graphiques, test, compatible avec les machines MAC et PC, une bataille de type « boss battle » au niveau final du jeu.

Exercice 4.2

Soit les projets a, b et c suivants :

a)

Tâche	Durée	Dépendances
init	4	
a	8	init
b	10	init
c	8	a,b

d	9	a
e	5	b
f	3	c,d
g	2	d
h	4	f,g
i	3	e,f

b)

Tâche	dep	Durée
a		10
b	a	5
c	a	2
d	a	3
e	b,c	7
f	b,d	9
g	c,d	5
h	e,f,g	6

c)

Tâche	Prédécesseurs	Durée
a		6
b	a	6
c	a	3
d	i,b	4
e	d,b	4
f	e	6

g	i,j	5
h	g	10
i		4
j	i	3
k	e,g	2

d)

Tâche	Prédécesseurs	Durée
a		10
b	e	10
c	d,f	10
d	a,f,b	20
e	a,f	8
f	a	5

1) Pour chaque projet, tracez le réseau PERT et calculer les dates début, fin et le chemin critique.

Exercice 4.3

2) Le projet consiste à développer un mini compilateur d'un sous-ensemble de JAVA. Le logiciel développé doit avoir une interface pour écrire le code et le compiler.
3) Etablir le découpage PBS et WBS.
4) Proposez une estimation de chaque activité
5) Déterminer les dépendances entre ces activités et dessinez le diagramme de PERT.
6) Calculer les dates début, fin et le chemin critique.
7) Tracer le diagramme GANTT de ce projet.

Chapitre 5 - **Gestion des Risques**

5.1 - **Introduction**

Tous les projets de développement et d'évolution des systèmes d'information vont faire face à des risques internes ou externes. Il n'y a plus de garanties dans les projets. La plus simple activité peut aller mal à cause des problèmes imprévisibles. A n'importe quel moment des choses peuvent apparaitre et changent les sorties de projet. Ces choses sont nommés risques. Connaitre à l'avance les difficultés inhérentes au projet permet l'adaptation de la conduite du projet de façon à éviter ou limiter les risques. Ce chapitre présente les différentes techniques de gestion de risques.

5.2 - **Gestion de risques**

La gestion des risques a pour objectif l'identification, l'analyse et le traitement des risques tout au long du cycle de vie de projet afin d'atteindre les objectifs du projet.

5.3 - **Le risque**

Définition 1 : Larousse : Possibilité, probabilité d'un fait, d'un événement considéré comme un mal ou un dommage : Les risques de guerre augmentent.

Définition 2: Le risque est un danger, inconvénient plus ou moins probable auquel on est exposé (Larousse).

Définition 3 : L'AFITEP définit le risque comme un *événement ou une condition possible dont la concrétisation aurait un effet sur au moins un des objectifs du projet*. Les objectifs peuvent se rapporter au contenu, aux délais, aux coûts et la qualité. Ces définitions montrent que le risque est lié à des impacts et il comporte de l'incertitude. Les risques qui n'ont pas un impact négatif (c-à-d un impact positif) sont dits opportunités par exemple [20]:

1. Diminution du cout d'un matériel dans le marché ce qui entraine un gain.
2. Si on peut combiner les commandes des équipements c1, c2 et c3 alors ces commandes vont être moins chère de 20% par rapport au planifié.
3. Si on peut avoir une personne expérimentée et productive en mois de mai alors le lot de travail n° x figurant sur le chemin critique sera fait plus vite que le planifié.

L'Incertitude est le résultat du manque des connaissances sur un événement entrainant la réduction de confiance envers les conclusions dérivées des données. La recherche des incertitudes peut aider à l'identification des risques.

Exemple d'événement indésirable : départ d'une personne clé, le client potentiel fait faillite…

Une autre définition de L'AFITEP est : le risque comme *la possibilité qu'un projet ne s'exécute pas conformément aux prévisions de dates d'achèvement, de coût et de spécifications, ces écarts par rapport aux prévisions étant considérés comme difficilement acceptables voire inacceptables.*

Définition 4 : Le PMBOK le considère comme une menace dont la concrétisation, incertaine, aurait un impact positif ou négatif sur au moins un objectif du projet tel que les délais, le coût, le contenu ou la qualité.

La démarche générale de management des risques comprend cinq étapes :

1. Identifier les risques ;
2. Evaluer leur impact possible sur les coûts, le délai et la qualité ;
3. Définir des actions ou prendre des dispositions aptes à réduire les risques jugés inacceptables;
4. Suivre les actions ou la mise en œuvre des dispositions, et surveiller régulièrement l'état des risques ;
5. Capitaliser l'expérience.

5.4 - **Identification des risques**

Ce processus consiste à identifier les risques possibles. La classification des risques peut aider dans le processus d'identification. Les classes sont généralement les types de sources de risques identifiés par des entreprises qualifiées qui ont travaillé sur des projets similaires.

Les risques sont classifiés en[13 ,24] :

Risques de projet concernent le déroulement du projet.

Risques techniques portent sur la qualité du produit.

Risques commerciaux peuvent affecter sa viabilité.

Les experts ajoutent aussi les risques communs à tous les projets.

On peut aussi classer les risques comme :

Externe : lié aux marché, environnement, gouvernement.

Interne : délai, cout, qualité, manque de l'expérience, mauvaise planification, personnes, groupe, matériel, équipements…

Technique : changement dans la technologie.

Invisible :10% des risques sont invisibles.

La table 9 présente quelques risques et indique leurs types.

Risque	*Projet*	*Tech*	*Commun*	*Propre*
Matériel non disponible		*		*
Spécifications incomplètes		*		

Table 10. Quelques risques et leurs types[13 ,22].

5.5 - Techniques de collecte d'information pour identifier les risques

Plusieurs techniques sont utilisées : Les interviews, le braistroming, la méthode delphi, analyse SWOT, analyse root-cause.

5.5.1 - **Analyse des risques**

L'objectif est de déterminer la criticité des risques.

5.5.2 - **Analyse qualitative**

L'objectif de cette analyse est de déterminer l'incidence du risque, c'est-à-dire le niveau d'acceptabilité du risque.

Pour y faire, Il est nécessaire de qualifier la probabilité d'apparition de chaque risque :

• **Probabilité faible** : il est peu probable que le risque se réalise ;

• **Probabilité moyenne** : il existe des signes indiquant que le risque est susceptible de se réaliser;

• **Probabilité forte** : le risque est certain ou en passe de se réaliser.

Puis leur criticité en termes de coût, délai, qualité et contenu technique :

• **Criticité faible** : ne compromet pas l'atteinte des objectifs du projet en termes de coût, délai, qualité ou fonctionnalités ;

• **Criticité moyenne** : peut affecter le périmètre du projet, éventuellement nécessite un avenant;
• **Criticité forte** : peut avoir comme conséquence une perte financière, une insatisfaction du client ou l'arrêt du projet.
La combinaison des deux facteurs, probabilité et criticité, permet de déterminer l'incidence du risque, c'est-à-dire le niveau d'acceptabilité du risque (voir figure 18).
On distingue alors 3 niveaux d'acceptabilité :
• Risque négligeable ;
• Risque à suivre ;
• Risque à traiter

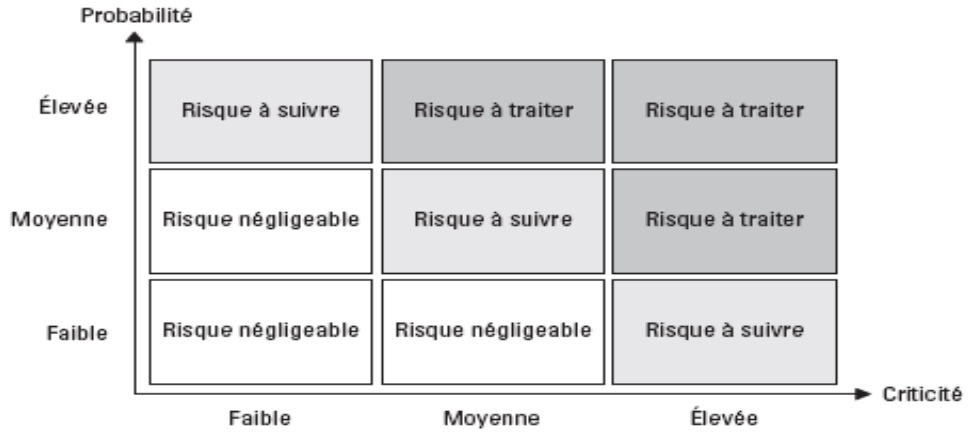


Figure 18 Les incidences possibles d'un risque

5.5.3 - **Analyse quantitative**

Une analyse profonde et complète l'analyse qualitative. Son objectif est d'analyser numériquement les risques.

5.6 - **Calcul du risque**

Le calcul du risque consiste à analyser numériquement la probabilité d'occurrence et l'impact du risque (le cout des conséquences) afin de trouver la valeur monétaire attendue du risque (VMA).

VMA de risque = impact*probabilité de risque

Si, par exemple, les conséquences sont de 6000 € et que l'événement a une certaine probabilité de se produire trois fois tous les 6 ans, on dira que le risque est de :6000*0.5=3000€

Exemple2

Jeu à deux dés à six faces
L'obtention d'un 7 fait perdre 60 DA, quel est le risque ?
Le risque=60*(1/6)=10

5.7 - **Arbre de décision**

Un outil pour représenter plusieurs alternatives afin de prendre une décision.

Exemple

Tu veux faire un voyage par vol de la cité c1 à la cité c2. Tu peux prendre le vol A ou le vol B. En se basant sur l'arbre suivant (figure 19) quel est le vol que tu va prendre et quelle est la valeur monétaire attendue de votre décision.

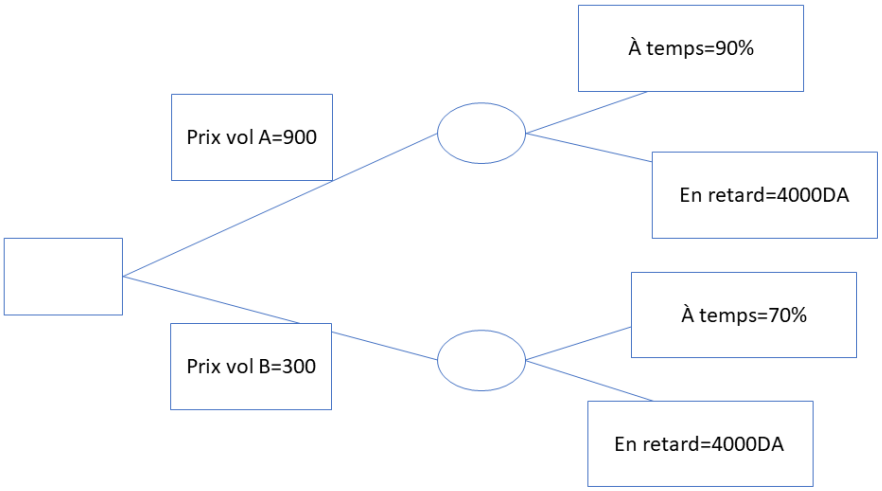


Figure 19Arbre de décision du problème[14]

Réponse

L'arbre de décision est présenté par la Figure 15.

VMA(airline A)=(10%*4000DA)+(90%*0DA)+900DA=1300DA.

VMA(airline A)=(30%*4000DA)+(70%*0DA)+300DA=1500DA.

Donc le vol A est moins risqué que le vol B.

5.8 - **Stratégies de traitement des risques**

1. *Eviter* : éliminer le risque par des traitements qui en empêche l'apparition.
2. *Atténuer le risqu*e (mitigation) : en cas d'impossibilité d'évitement, alors prendre des actions pour diminuer la VMA du risque.
3. *Transférer* : chargé (payer) une autre entité de ce risque (ex : compagnie d'assurance) afin qu'elle accepte les impacts.
4. *Accepter :* en cas ou l'entreprise ne peut ni éviter, ni atténuer et ne peut pas transférer le risque alors il accepte.[20]

5.9 - **Atténuation des risques**

Une stratégie proactive qui consiste à trouver des approches pour diminuer la probabilité et/ou l'impact[22].

Par exemple si on risque d'avoir des problèmes avec le logiciel développé (e.g., non satisfaction de client), il est préférable d'étudier à l'avance le logiciel en se basant sur un prototype.

5.10 - **Plan de gestion de risque**

Le plan contient :

1. Un identificateur et description de risque.

2. Une estimation de la probabilité et de l'impact

3 Une liste des actions d'atténuations, les plans de secours et leurs triggers, les individus responsables.

Exemple de plan [13]:

Risk ID: 1-010-77	Prob: 10 percent	Impact: very high
Description: Specialized hardware may not be available.		
Mitigation strategy: Build simulator, accelerate hardware development.		
Risk trigger: Hardware falling 1 week or more behind schedule.		
Contingency plan: Outsource hardware development as backup, deliver system on simulator.		
Status/date/responsible person: Created – Jan 1,01 – Fred Jones Sim. completed – Feb 10, 01 – Bill Olson		

5.11 - **Exercices**

Exercice 5.1[13,22]

Classifier les risques suivants en risques de type technique, commercial, projet, juridique, commune ou propre.

Matériel non disponible, Spécifications incomplètes, Utilisation de méthodes spécialisées, la démission d'une personne clé, Sous-estimation des efforts nécessaires, peu de clients potentiels, l'entrée en vigueur de nouvelles lois.

2- Citer quelques opportunités.

Exercice 5.2[13]

Si une faute a une probabilité de 50% de ne pas être détectée. Cette faute peut provoquer une défaillance de 20000 DA.

1. Calculer la VMA du risque.

2. Si le recrutement d'un expert nous coûtait 2000DA et son travail éliminera ce type de faute par 50%.

Est-ce qu'il est utile de recruter de cet expert ?

Exercice 5.3

Proposez des stratégies pour atténuer les risques listés dans l'exercice N °1.

Exercice 5.4[20]

A partir des réponses suivantes, dites quels est le type de traitement de risque qui a été choisi par l'entreprise face à certains risques :

Réponse au risque	Stratégie
Supprimer un lot de travail du plan	
Mettre un membre de l'équipe en contact permanant avec le client afin de lever les ambigüités au niveau du cahier de charge	
Affecter un lot de travail à une personne compétente dès qu'elle est libre.	
Négocier tôt avec le fournisseur les équipements nécessaires au projet afin de garantir un prix faible	
Affecter un lot de travail à une entreprise plus expérimentée	
Former l'équipe	
Demander au client de faire une partie de travail	

Chapitre 6 - **Assurance de la qualité**

6.1 - **Introduction**

Ce chapitre présente quelques techniques utilisées pour assurer la qualité. Il commence par la définition du processus d'assurance qualité ensuite il présente en détail l'inspection formelle un processus pour réaliser la qualité logicielle.

6.2 - **Assurance de la qualité**

Il n'y a pas de consensus sur la définition du terme qualité. La norme AFNOR X50-120, définit la qualité comme « l'ensemble des propriétés et caractéristiques d'un produit ou service qui lui confèrent l'aptitude à satisfaire des besoins exprimés ou implicites ».

Il y'a de la qualité dans le cas où le produit ou le service ne satisfait les besoins de client.

Le logiciel comme partie du SI présente des caractéristiques qui marquent la problématique de sa qualité [12]:

• il est immatériel ;
• il est reproductible ;
• il nécessite une maintenance ;
• il possède une dimension subjective.

La technique couramment utilisée pour réaliser la qualité est l'inspection[24 ,13]. L'objectif des inspections est de trouver les erreurs. La littérature montre que les inspections formelles sont plus efficaces que les approches informelles. L'inspection permet d'apprécier la qualité d'un document en fonction d'un référentiel. La métrique couramment utilisée pour évaluer les résultats de l'inspection est le nombre de fautes trouvées par KLOC.

Une autre approche moins formelle est la lecture *croisée* qui *met* en jeu deux acteurs appartenant à des projets différents, par exemple deux sous-projets du même projet global. Elle favorise la cohérence entre projets ou sous-projets.

6.3 - **Inspection formelle**

L'inspection formelle est une activité formelle et planifiée.

Un concepteur présente des documents sur un projet à un autre groupe de concepteurs qui en évaluent les aspects techniques avec pour objectif de trouver les erreurs[24].

Le contrôle est effectué par des personnes techniquement compétentes et avec une participation active de l'auteur. Elle est périodique au cours du processus de développement et se porte sur un produit fini.

6.4 - **Rôles pour une inspection**

Il existe différents rôles, les principaux rôles sont[24,13] :

- Le modérateur :
    - Il choisit l'équipe
    - Il dirige l'inspection et rendre compte des résultats.

- Le lecteur :
    - Il n'est généralement pas l'auteur du produit
    - Il guide l'équipe dans la structure du produit
- Le secrétaire :
    - Il consigne le déroulement de l'inspection
    - Il note toutes les erreurs trouvées
- L'auteur :
    - Il est à l'origine du produit examiné
    - Il répond aux questions durant l'inspection
    - Il corrige les erreurs et fait un rapport au modérateur

Etapes de l'inspection

- Présentation générale par l'auteur au reste de l'équipe
- Préparation
- Les membres de l'équipe étudient le produit dans la limite d'un temps calculé en fonction du nombre de LOC
- Ils peuvent s'aider d'une liste de contrôles

6.5 - **Réunion pour l'inspection**

- Organisée par le modérateur

- Le lecteur conduit l'inspection
- Le secrétaire enregistre les problèmes dans un rapport
- En cas de désaccord, il est possible de produire des rapports individuels

6.6 - **Checklist**

Le processus d'inspection ou de relecture peut être guidé par les Checklists. Une Checklist est une liste des items à vérifier durant la relecture. Dés fois ces items sont exprimés sous forme de questions auxquelles il faut répondre. Ces items sont généralement construits par des personnes expérimentées dans le domaine du document ou produit analysé[22].

Les check-lists sont des référentiels et orientent l'inspection ou la relecture vers des problèmes potentiels dans le produit ou le document analysé.

Exemples des items à vérifier au niveau conceptuel et au niveau de programmes :

Est-ce que toutes les classes ont été identifiées ?

Est-ce que toutes les relations ont été identifiées ?

Est-ce que toutes les variables ont été initialisées ?

6.7 - **Exercices**

Exercice 6.1 : Comment évaluer une liste de contrôle (un check-list)?

Exercice 6.2 :

Construire une check-list pour vérifier des programmes écrits en JAVA.

Construire une check-list pour vérifier la conception du logiciel.

Exercice 6.3 : Comment mesure-t-on l'efficacité d'une inspection ?

Chapitre 7 - **Suivi de Projet**

7.1 - Introduction

Durant la phase d'exécution de projet informatique, il est possible que projet dérape des objectifs fixés à l'avance. Par conséquent il est important de déclencher périodiquement un processus de suivi et contrôle pour mesurer l'état d 'avancement du projet par rapport au planifié, pour prendre des mesures correctives en cas de dérapage et pour traiter les demandes de changement.

Ce chapitre introduit la problématique de pilotage de projet informatique. Il présente les différentes techniques utilisées pour suivre et contrôler la progression du projet.

7.2 - Suivi et contrôle de projet

Le suivi et contrôle fait référence à l'ensemble des activités et des processus nécessaires pour réussir la conduite des projets et la gestion des risques. Généralement l'absence des moyens de contrôle effectifs de contenu, de la qualité, de coût et des risques entrainent l'échec ou l'annulation de projet.

Le suivi et le contrôle est l'ensemble des activités efficaces que le chef de projet les exécute pour garder les performances de projet et les ressources utilisées dans des niveaux optimaux.

Le processus de pilotage comprend :
• Le pilotage de la progression des travaux par rapport au planifié et de l'échéance ;
• Le suivi des couts en regard de ce qui a été fixé à l'avance ;
• Les éventuels arbitrages entre les objectifs (le périmètre), les ressources et les délais

7.3 - Suivi individuel

Il permet de détecter les difficultés éventuelles d'un intervenant à partir du rapport hebdomadaire qu'il doit fournir[12].

7.4 - Le suivi économique

7.4.1 - La méthode de la valeur acquise

La méthode de la valeur acquise (*Earned Value Method)* est une méthode pour mesurer les performances du projet, par rapport aux contenu, délai et cout prévus de projet à une date t et de faire des prévisions des performances futurs date d'achèvement et cout de projet. Les

résultats de l'analyse de la valeur acquise indiquent les déviations potentielles de projet par rapport aux contenus, cout et délai de bases (prévus), des opportunités pour améliorer le projet, ou juste pour avoir connaissances sur la santé du projet.

Par rapport à la gestion des couts, l'analyse de la valeur acquise s'intéresse aux relations existantes entre trois indicateurs normalisés (calculés par la plupart des logiciels de suivi de projet). Ces indicateurs déterminent les performances de projet.

• *Valeur planifiée* (VP) : c'est le coût du travail planifié, basé sur l'estimation des charges et le coût prévu des ressources. C'est qu'on a planifié de faire jusqu'à la date t. par exemple si le budget de projet est 500000 et son delai est de 12 mois alors la VP du 6 mois=250000

• *Coût réel* (CR): c'est les sommes d'argents dépensées à la date t. ce sont les dépenses provenant du temps consommé. C'est le cumul de ce que l'on a effectivement dépensé jusqu'à la date t. par exemple il est possible qu'à la fin du sixième mois nos dépense sont de 230000.
• *Valeur acquise* (VA) : cet indicateur est basé sur l'avancement réel du projet et représente le travail physique effectué a la date t. C'est ce que l'on aurait dû dépenser pour le travail effectivement réalisé jusqu'à la date t. c'est la valeur estimée (cout) du travail effectué jusqu'à la date t.

VA=VP*% de l'achevé à la date t.

Par exemple si au sixième mois on a achevé que 45% du travail alors la VA=500000*45/100=225000

Budget à l'achèvement (BAA) : c'est le budget du projet et correspond à la VP jusqu'à la fin prévue du projet.

Ces trois indicateurs sont cumulatifs et peuvent être calculés pour une activité, un ensemble d'activités ou l'ensemble du projet.
A partir de ces indicateurs on déduit deux indicateurs complémentaires : l'écart de coût et l'écart de délai.

• *Écart de coût* (EC) = VA – CR. S'il est négatif, on dépense plus que prévu ; s'il est positif, on sous-consomme, et s'il est nul, la productivité réelle correspond exactement à la productivité planifiée.

• *Écart de délai* (ED) = VA – VP. S'il est négatif, on avance moins vite que ce que l'on avait planifié ; s'il est positif, on a pris de l'avance, et s'il est nul, le travail se réalise conformément à ce qui avait été planifié.

La courbe en S (figure 9) est un bon outil pour visualiser l'état d'un projet à une date t et les retards et les écarts à un instant t et pour mesurer leur impact sur les engagements de fin de projet.

Dans la courbe en S (voir figure 9), il s'agit de comparer respectivement :

• ce qui est prévu au budget (coût budgété du travail prévu : VP) ;

• et ce qui est réalisé (coût réel du travail effectué : CA) ;

• au coût budgété du travail effectué (EV : Valeur budgétaire du travail réalisé ou Valeur acquise = Valorisation des tâches effectuées par leurs coûts prévisionnels définis dans le budget à date).

Exemple[12]

On suppose qu'on va calculer la performance après 10 jours de travail.

Le budget total du projet est de 150 KDA.

La VP au jour 10 est de 80 KDA. C'est ce que l'on a prévu de faire et de dépenser, par exemple 8 jours de travail pour l'ensemble de l'équipe de projet à 10 KDA par jour. Le CR est de 95 KDA. On a donc dépensé plus que ce que l'on avait planifié.

Mais il est possible qu'on fait de l'avance par rapport au prévu ? La VA est l'outil qui va nous donner la réponse. La VA= 70 KDA, c'est-à-dire que, par rapport à l'avancement réel, on aurait dû dépenser uniquement 70 KDA.

L'écart de coût est de : 70 KDA – 95 KDA = – 25 KDA.

L'écart de planning est de : 70 KDA – 80 KDA = – 10 KDA.

Le projet, au jour 10, est donc en retard et en surconsommation.

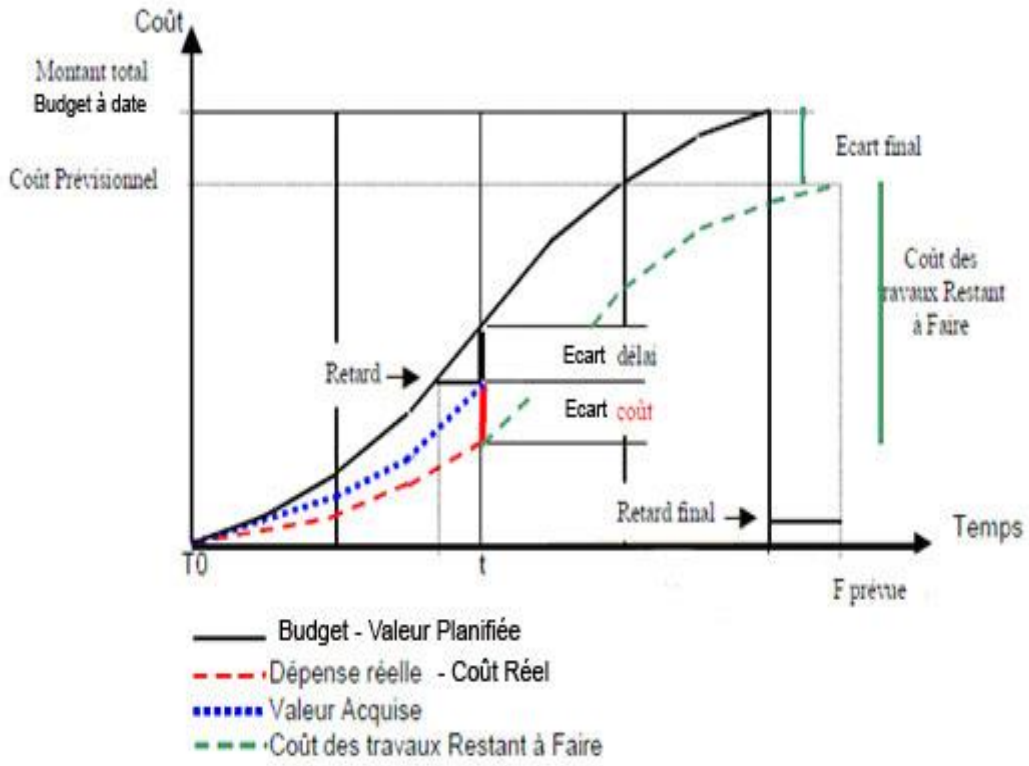


Figure 20courbe en S

On peut traduire les écarts sous forme d'indicateurs de performance :

• *Indice de performance des coûts* (IPC) = VA/CR

• *Indice de performance des délais* (IPD)= VA/VP

On peut alors faire une prévision des coûts à l'achèvement du projet :

• *Prévision à l'achèvement* (PAA) *Estimate at completion* (EAC) : c'est l'estimation du coût total du projet, compte tenu des performances jusqu'à ce jour et des risques prévisibles.

Pour calculer la PAA, on distingue trois cas de figure.

• 1. On considère que les écarts sont accidentels et on n'en tient pas compte pour le futur. Alors :

PAA = Coûts réels à ce jour + Travail restant = CR + (Budget à l'achèvement – Valeur acquise).

• 2. On considère que les écarts sont représentatifs du futur et on fait une projection. Alors : PAA = Coûts réels à ce jour + (Travail restant/indice de performance des coûts) = CR + (BAA – VA)/IPC.

On peut aussi utiliser l'IPC.

• 3. On considère que les conditions ont changé, et l'on fait une nouvelle estimation. Alors : PAA = Coûts réels à ce jour + Coût estimé à l'achèvement (*Estimate to complete* (ETC))= CR + CEA.

On peut enfin calculer l'écart prévisible en fin de projet :

• *Écart à l'achèvement* (EAC) *Variance at completion* (VAC).

EAC = Budget à l'achèvement – Prévision à l'achèvement = BAA – PAA.

Éventuellement, on calcule l'indice de performance nécessaire pour terminer le projet dans le budget :

• *Indice de performance requis* (IPR) En anglais : *To complete performance index* (TCPI). :

IPR = Travail restant/Budget restant = (BAA – VA)/(BAA – CR).

7.5 - **Suivi des erreurs**

Le suivi des erreurs est une des bonnes pratiques de gestion qui consiste à garder trace des erreurs produites et les intervalles de temps qui sépare leurs apparitions[13]. Ces informations peuvent être utilisées pour planifier la livraison (déterminer la date) et pour motiver les développeurs et testeurs afin qu'ils fixent la réduction des erreurs comme un objectif aussi[13].

Taux d'erreur[13]

Le taux d'erreur est défini comme étant l'inverse du temps inter-erreurs. Par exemple, si chaque 3 jours deux erreurs se produisent alors le taux d'erreur instantané est de 0,33 erreur / jour. Le taux d'erreur instantané actuel peut servir comme une bonne estimation du taux d'erreur actuel. Si les défauts responsables des erreurs ne sont pas éliminés après détection des erreurs, alors le taux d'erreur cumulé (la somme de toutes les erreurs trouvées divisées par le temps total) donnera une bonne estimation des futurs taux d'erreur. Généralement, on corrige les erreurs et on élimine les défauts ce que rend le taux d'erreur en état de diminution et les temps inter-erreurs en état d'augmentation. La projection graphique de ces données peut donner une indication sur le futur du taux d'erreur.

Lorsque le graphe résultat de la projection s'intersecte avec l'axe des x, la valeur estimée du taux d'erreur sera zéro (absence des erreurs). Si l'axe des abscisses x représente le nombre d'erreurs, la valeur de l'ordonnée quand x=0 peut être utilisée comme une estimation du nombre total d'erreurs dans le logiciel[13].

– on peut projeter graphiquement ces données et utiliser des techniques de régression linéaire pour estimer le futur des taux d'erreur.

Exemple[13]

– Les durées entre une série d'erreurs sont les suivantes : 4, 3, 5, 6, 4, 6, 7.

Les taux d'erreur instantanés sont : 0,25, 0,33, 0,20, 0,17, 0,25, 0,17 et 0,14. La représentation graphique de ces données montre une courbe descendante (Figure 21). Ce graphe indique que le taux d'erreur réel est en état de diminution.

Durée	4	3	5		6	4	6	7
Taux	0.25	0.33	0.20		0.17	0.25	0.17	0.14

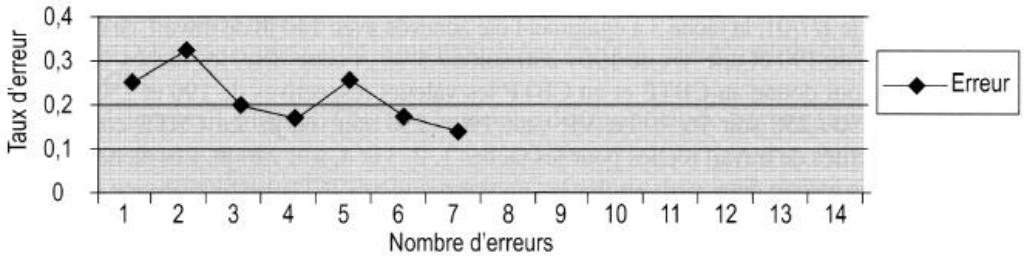


Figure 21 graphe de taux d'erreurs[13]

Si on continue le traçage de cette courbe, on trouve qu'elle va couper l'axe des abscisses au point x= 11 erreurs environ. Ce qui veut dire que le taux d'erreur devrait donc être nul après le 11^{eme} erreur et puisque on 7 erreurs ont été trouvées, normalement il ne restera pas plus que 4 erreurs.

7.6 - **Exercices**

Exercice 7.1

Votre plan est de développer un logiciel dans une durée de 1e année. Le budget prévu de ce projet est de 12500 DA /mois. Après 6 mois vous avez trouvé que vous n'avez réalisé que 20% du travail et vous avez dépensé 90000DA.

1. Calculer : l'écart de cout, l'écart de délai l'IPC, l' IPD,.
2. Quel est alors l'état de votre projet à cette date ?

3. Tracer la courbe S qui montre l'état de projet à cette date ?
4. Calculer l'écart à l'achèvement si on considère que ces écarts sont accidentels

Exercice7.2

Soit le projet suivant :

1 jour=10 dinars algériens.

Tâche	Durée en jours	Dépendances
init	4	
a	8	init
b	10	init
c	8	a,b
d	9	a
e	5	b
f	3	c,d
g	2	d
h	4	f,g
i	3	e,f

1) Si Après 4 jours de travail, on a réalisé 10% du travail et on a dépensé 1000DA.
2) Calculer : L'écart de coût, L'écart de délai l'IPC, l' IPD,.

2.1 Décrire l'état de votre projet à cette date ? (2 pts)

Chapitre 8 - **Dimension humaine d'un projet**

8.1 - **Introduction**

La réussite de projet requiert une organisation effective en équipes de différentes personnes ayant différentes compétences. Ce chapitre définit le terme équipe puis présente les différentes approches d'organisation d'une équipe.

8.2 - **Équipe**

Une équipe est un « groupe de personnes unies dans une tâche commune » (Dictionnaire Robert).

L'équipe projet est composée de différentes personnes de différentes directions de l'entreprise et de personnes de sociétés de prestations de service ou de fournisseurs. Elle est géré par le chef de projet.

Les fonctions de l'équipe projet sont :

• la réalisation de travaux

• Rend compte de leur avancement au chef de projet.

8.3 - **Approches d'organisation d'équipe**

Organiser un groupe de personnes en équipes efficaces et efficientes est une tâche difficile.

Laisser une équipe développer son propre paradigme peut être une solution risquée. Organiser l'équipe en se basant sur le projet et les membres de l'équipe peuvent aider à éviter les risques.

Un aspect important d'une équipe est son degré de structuration. Alors que certains groupes des programmeurs peuvent travailler de manière très indépendante, d'autres groupes ont besoin d'une structure solide pour progresser[13].

Dans une équipe fortement structurée, de petites tâches sont affectées à chaque membre.

Dans une équipe faiblement structurée, les tâches sont généralement d'une durée longue et ouverte. Il existe plusieurs méthodes courantes d'organisation d'équipes impliquées dans le

développement logiciel - équipes fonctionnelles, équipes de projet et chef programmer team[13].

8.3.1 - **Equipe fonctionnelle**

Dans une organisation fonctionnelle, tous les membres de l'équipe effectuent le même type de travail, de sorte qu'il y a une équipe de programmeurs, une équipe d'analystes, une équipe de testeurs, etc. Les membres d'une équipe, à tout moment, peuvent travaillées sur différents projets ; le travail d'un seul projet est réalisé par des personnes de différentes équipes.

Un problème majeur avec ce mode d'organisation est que la communication entres membres devient difficile et en plus le chef de projet ne peut pas avoir autant de contrôle qu'il souhaite sur l'équipe.

8.3.2 - **Equipe de projet**

Les membres de l'équipe sont engagés dans le développement du même projet. L'équipe est composée de personnes qui effectuent tout le travail du développement : analyse des besoins, programmation, tests, etc.

L'inconvénient majeur d'une équipe de projet est que sa composition augmente et décroît à mesure que les exigences du projet changent.

8.3.3 - **Chief-Programmer Team**

Ce concept est développé par IBM. Il attribue des rôles spécifiques aux membres de l'équipe. Le chef des programmeurs est le programmeur le plus expérimenté et il dirige l'équipe. Les non-programmeurs sont utilisés dans l'équipe pour la documentation et les fonctions non techniques. Les programmeurs juniors font partie de l'équipe et sont encadrés par le chef des programmeurs.

8.4 - **Exercices**

Exercice 8.1. Comparer les différentes approches d’organisation en équipe.

Exercice 8.2 discuter le problème de communication au niveau de l’équipe.

Chapitre 9 - **Métriques logicielles**

9.1 - **Introduction**

You can’t control what you can’t measure. —Tom DeMarco

What is not measurable, make miserable — Galileo

« When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the state of science. » (William Thompson, Lord Kelvin, Popular Lectures and Addresses [1891-1894], Bartlett's Familiar Quotations, 14th edition, 1968, p. 723a)

La science est basée sur la mesure (i.e., métrique) et les scientifiques ont beaucoup travaillé pour rendre les choses mesurables. La mesure joue aussi un rôle important dans l‘ingénierie logicielle. La mesure est utilisée pour mieux gérer le processus de développement, assurer la qualité, planifier le travail, prédire les couts, améliorer les processus, contrôler la progression...etc. les meilleurs développeurs utilisent aussi les mesures pour avoir une idée sur la qualité du logiciel, savoir si les besoins sont consistants et complets, est ce que la distribution code est prête …etc.

9.2 - **Processus de mesure**

Mesurer est un processus qui consiste à affecter des nombres ou des symboles à des attributs d'entités du monde réel afin de les décrire attributs selon des règles précises[10].

Un attribut est une caractéristique ou une propriété d'une entité.

Exemples : âge, couleur, poids, volume, taille ou prix d'un objet.

9.3 - **Types de mesures**

- Mesures directes [25]: La valeur est attribuée directement à l'attribut. Exemple la taille du programme peut se mesurer par le nombre de lignes de code.
- Mesure indirecte [13, 25]: La valeur est résultat d'une transformation de mesures directes. beaucoup de 'ilities'' (maintainability, readability, testability, quality, complexity, etc sont indirectemnt mesurables.

9.4 - **Pourquoi mesurer**

Les objectifs de la mesure sont nombreux et on peut citer :

1. La compréhension
2. L'évaluation
3. La prédiction
4. L'amélioration

9.5 - **La théorie représentationnelle de la mesure**

Pour que la mesure soit valide, elle doit respecter la théorie représentationnelle[13].

Cette théorie fait appel à un *système relationnel empirique* – la réalité – et un système *relationnel numérique*, et un mapping M qui relie les deux structures. Ce lien est exprimé par la notion d'*homomorphisme*.

9.5.1 - **Système relationnel empirique**

Le système relationnel empirique (E, R) est formé de deux parties [13]:

1. Un ensemble des entités E
2. Un ensemble de relations R

La relation est souvent égale ou inférieur de. Cette relation peut être d'ordre partiel.

9.5.2 - **Système relationnel numérique**

Le système relationnel empirique (N, P) est formé de deux parties[13] :

1. Un ensemble des entités N souvent nommé ensemble de réponse. Et généralement formé par des nombres.
2. Un ensemble de relations P. souvent c'est < ou <=

Le mapping M qui préserve la relation, transforme (E, R) en (N, P). La restriction principale sur ce mapping se nomme ***condition de représentation***.

9.6 - **Condition de représentation**

La relation empirique du monde réel doit être préservée quand on fait le choix de la représentation par des nombres ou des symboles [13].

Exemple [26]

• X est plus grand que Y

•Si la taille est mesuré par la valeur en kilogrammes de la personne alors cette mesure ne préserve pas la relation empirique.

Formellement

Soient x et y deux entités de E, la condition de représentation est définie par $R(x, y) \Leftrightarrow P(M(x), M(y))$

Exemple[26]

– Soient E un ensemble de personnes et Omar et amine deux personnes.

– Soit R la relation est_plus_grand_que. On sait que omar est plus grand que amine

– Soit M la mesure de la taille en cm

– Soit P la relation ">"

– Par la mesure, on obtient M(omar) = 180 et M(amine) = 185

– est_plus_grand_que (amine, omar) est vrai (monde réel) et M(amine) > M(omar) est vrai (relation mathématique)

– Alors que est_plus_grand_que (omar, amine) et M(omar) >M(amine) sont tous les deux faux.

Donc La condition de représentation est valide.

On peut utiliser la relation formelle pour raisonner sur le monde réel. Par exemple en mesurant la taille en LOC(ligne of code) on peut définir de nouvelles mesures comme par exemple la taille moyenne d'un ensemble de programmes qui n'ont pas d'équivalent dans le monde réel . La relation ">" définit précisément la relation est_plus_grand_que et rend le raisonnement plus large.

Les critères de validité d'une métrique (mesure) sont [13]:

1. une métrique doit permettre la différentiation entre différentes entités.

2. une métrique doit respecter la condition de représentation.

3. chaque unité d'attribut doit contribuer une quantité équivalente à la métrique.

4. des entités différentes peuvent avoir la même valeur.

9.7 - **Les échelles**

Il est possible de représenter un même système empirique de différentes manières. On parle alors d'échelles. Dans la littérature on trouve 5 types d'échelles : nominale, ordinale, intervalle, ratio, absolue… etc. Une échelle E1 est plus riche qu'une échelle E2 si tout ce qui est faites dans E2 est possible dans E1.

Une échelle est acceptable si elle est dérivée d'une représentation par une transformation admissible qui dépend du type de l'échelle.

Les échelles présentées ci-dessous sont présentées du moins riche à la plus riche[13].

9.7.1 - **Echelle nominale**

Cette échelle est la moins riche. Elle consiste en une classification simple et il n y'a aucun ordre entre les classes même si ce sont des nombres.

Par exemples

-Couleurs : rouge, vert, …

– Langage de programmation, : procédurale, OO, fonctionnelle…

9.7.2 - **Échelle ordinale**

Plus les propriétés de l'échelle nominale, on ajoute la notion d'ordre à l'échelle.

Exemples :

-Taille : petite, moyene, grande.

-Mention :faible, moyen, assez bien, bien…

– Niveaux de maturité • Capability Maturity Model CMM

9.7.3 - **Échelle intervalle**

Plus les propriétés de l'échelle ordinale, on ajoute la notion de distance relative entre deux classes de mesures.

Exemples

– Température en Celsius ou en Fahrenheit

– Date d'apparition/détection d'une faute

9.7.4 - **Échelle ratio**

Ajout de la notion de zéro absolu à l'échelle intervalle.

Zéro absolu = absence de l'attribut mesuré

Exemples

– Température en Kelvin

– Temps entre 2 fautes

9.7.5 - **Échelle absolue**

L'attribut ne peut être mesuré que d'une seule façon qui consiste généralement à compter.

Exemples

-Nombre des étudiants dans une classe

– Nombre de classes dans un packages

– Nombre d'erreurs trouvées lors du test d'intégration

Chaque type d'échelle autorise un certain types de transformations Table 11.

Le type d'échelle d'une mesure détermine quelles sont les opérations possibles sur les données Par exemple, les analyses statistiques utilisent les opérateurs arithmétiques (+, –, ÷, ×). Ces opérations ne sont pas permises sur des données de type échelles nominale et ordinale, les analyses statistiques qui les utilisent ne sont pas significatives[10 ,13].

Choix d'échelles – les mesures objectives sont préférables par rapport aux mesures subjectives (nominale et ordinale). Exemple : pour mesurer le couplage d'une classe, utiliser une mesure quantitative du couplage plutôt qu'une échelle de Likert (subjective) – Les mesures subjectives sont acceptables s'il est difficile de définir des mesures objectives et si l'imprécision est bien comprise.

Transformations admissibles	Types d'échelles	Exemples
$M' = F(M)$ F est une bijection	Nominale	Etiquettes
$M' = F(M)$ F est monotonne croissante $M(x) \geq M(y) \Rightarrow$ $M'(x) \geq M'(y)$	Ordinale	Difficulé, préférence
$M' = \alpha M + \beta \ \ (\alpha > 0)$	Intervalle	Température
$M' = \alpha M$	Ratio	Taille
$M' = M$	Absolue	Comptage d'entités

Table 11 les transformations autorisées par l'échelle.

9.8 - **Mesure en Génie Logiciel**

Classification des mesures : on peut distinguer trois classes d'entités dont les attributs sont intéressants à mesurer :

- Le processus de production de logiciels.
- Les produits : logiciels, spécification ou documents.
- Les ressources.

On distingue pour chaque entité :

- Des attributs internes : dépendent de l'entité seulement.
- Des attributs externes : dépendent de l'entité dans un environnement ou un contexte donné. Ces attributs sont plus difficiles à mesurer : productivité, fiabilité, qualité, etc. ils sont généralement mesurés indirectement.

9.8.1 - **Attributs pour la mesure du processus logiciel[10, 26]**

Construction de spécifications

- Attributs internes : temps, effort, nombre de changements dans les besoins.
- Attributs externes : qualité, coût, stabilité.

Conception

- Attributs internes : temps, effort, nombre de fautes dans les spécifications.
- Attributs externes : coût, coût-productivité

Test

- Attributs internes : temps, effort, nombre de fautes découvertes.
- Attributs externes : coût, stabilité.

9.8.2 - **Attributs pour la mesure des produits[10, 26]**

Spécifications

- Attributs internes : taille, modularité, ré-utilisation,redondance, fonctionnalité, correction syntaxique.
- Attributs externes : lisibilité, facilité de maintenance.

Conceptions

- Attributs internes : taille, modularité, ...
- Attributs externes : qualité, complexité, maintenabilité.

Code

- Attributs internes : taille, ré-utilisation, complexité algorithmique
- Attributs externes : fiabilité, facilité d'utilisation, facilité de maintenance.

Test de données

- Attributs internes : taille, couverture, ...
- Attributs externes : qualité, ...

9.8.3 - **Attributs pour la mesure des ressources[10,26]**

Personnel

- Attributs internes : âge, salaires, ...
- Attributs externes : productivité, expérience, intelligence, ...,

Equipes

- Attributs internes : taille, communication, structuration, ...
- Attributs externes : qualité, productivité, ...,

Logiciels

- Attributs internes : prix, taille, ...
- Attributs externes : fiabilité, facilité d'utilisation, ...

Matériel

- Attributs internes : prix, vitesse, taille mémoire, ...

- Attributs externes : fiabilité, ...

Bureaux

- Attributs internes : taille, température, éclairage, ...
- Attributs externes : confort, qualité, ...

9.9 - Métrique de produit

Ces métriques sont calculées à partir des artefacts sans prendre en considération comment ils ont été produits. Ces métriques sont dérivées du code source, des artefact (e.g., documents) produits durant l'analyse, la conception…etc. Par exemple le nombre des paragraphes composant la spécification pourra être une métrique.

9.9.1 - Métriques de la taille de l'objet logiciel

Historiquement, un des premiers attributs mesurés du logiciel. La taille ici est similaire au concept taille d'un humain (attributs grandeur, poids). On parle de la Taille physique – Fonctionnalité – Complexité

1. Taille

1.1 Taille physique : Nombre de lignes de code

Cette métrique est souvent abrégée LOC « Lines Of Code », ou en unité de milliers des lignes de code KLOC « Kilo Lines Of Code ». Cette métrique mesure la taille physique d'un produit et elle est ambiguë puisque on ne pas définir exactement c'est quoi une ligne. Et on peut compter les lignes de code de différentes façons. Par exemple

1. Est-ce qu'on compte les commentaires et les déclarations ?
2. Comment traiter une ligne qui contient plusieurs instructions ?

La taille physique totale est LOC = NCLOC (**NonComment Lines Of Code)** + CLOC (**Comment Lines Of Code)**

NCLOC (NonComment Lines Of Code)

« Est considérée comme ligne de code toute ligne qui n'est ni un commentaire ni une ligne vide. Le nombre d'instructions ou de parties d'instructions n'est pas important. » Ceci inclut les en-têtes de programmes, les déclarations et les instructions (exécutables et non exécutables)

Le nombre de commentaires est mesuré par le nombre de lignes de commentaires (CLOC) – La taille physique totale est donc LOC = NCLOC + CLOC – On peut ainsi mesurer la densité de commentaires par CLOC/LOC – LOC peut être plus pratique à utiliser que NCLOC

1.2 Le nombre de caractères CHAR

Est une autre mesure de la taille physique. Elle est facile à caculer et on peut l'utiuliser pour calculer le LOC en utilisant la formule suivante : LOC = CHAR/α où α est le nombre moyen de caractères par ligne de texte.

1.3 Le nombre d'octets nécessaires au stockage du programme

C'est une mesure indépendante du type de langage et elle est facile à extraire.

1.4 Le nombre d'instructions livrées (DSI)

C'est une mesure liée au langage de programmation utilisé.

9.9.2 - McCABE'S CYCLOMATIC NUMBER

La complexité logicielle peut être divisée en trois catégories : la complexité structurelle, la complexité algorithmique et la complexité du problème à résoudre. Dans cette section, nos intérêts portent plus particulièrement sur les métriques de complexité structurelle qui tentent de décrire les attributs internes de la conception ou du code source du système.

L'objectif de l'approche est d'établir une relation entre la structure du produit logiciel et la qualité du logiciel comme la fiabilité, la testabilité et la facilité de maintenance du système

McCabe a proposé une métrique pour mesurer la complexité C d'un programme. Il a supposé que la complexité est liée au CFG (**Graphe de flot de données)** du programme.

Graphe de flot de données

Le Graphe de flot de données noté CFG (control flow graph) est un graphe orienté (N,E). Les nœuds N représentent des blocs d'instructions séquentiels. Les arcs E représentent le transfert de contrôle et peuvent être étiquetés avec un attribut représentant la condition du transfert de contrôle. Un arc entre deux nœuds n1 et n2 représente le transfert du contrôle de n1 `a n2. Ce graphe possède un nœud d'entré début et un nœud de sortie fin.

Principes du CFG

La figure suivante montre la représentation du concept ; test, boucle, séquence dans un CFG.

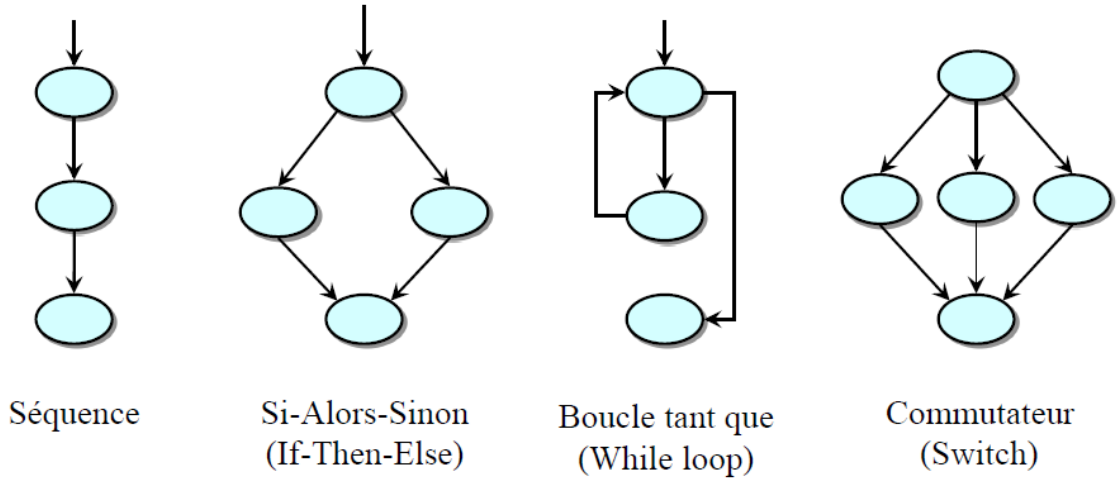


Figure 22 la représentation des concepts test, boucle, séquence dans un CFG

La complexité cyclomatique est calculée à partir de ce graphe en utilisant la formule suivante[10,13] :

C =e-n +2p

e : nombre des arcs qui composent le graphe CFG.

n : nombre des nœuds qui composent le CFG

p : nombre des composant fortement connectés(normalement égale à 1).

Exemple

Soit le programme suivant et son CFG présenté par la figure 22.

```
Read(a)
Signe=’negatif’
If (a>0)  signe=’positif’ ;
If (a==0) signe=’null’ ;
Print(signe)
End.
```

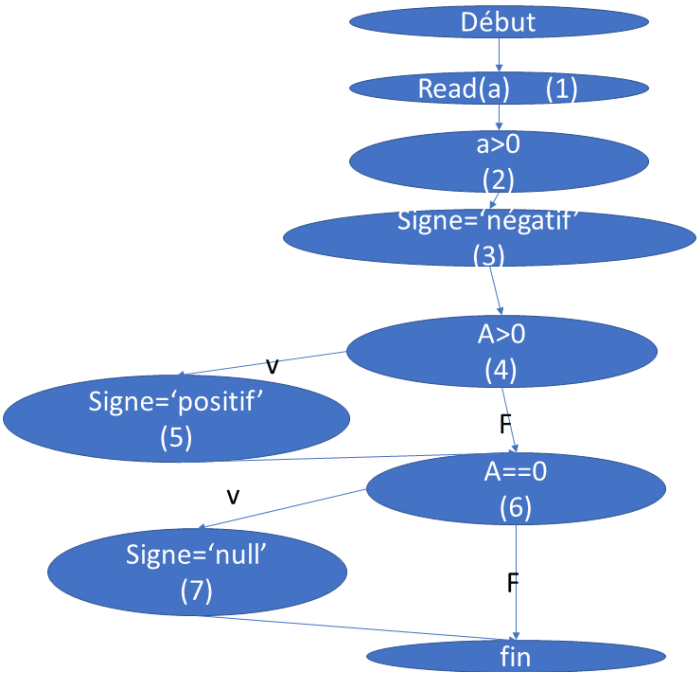


Figure 23 Le CFG du programme précédent

Ce graphe est composé de 9 nœuds et 10 arcs.

La complexité cyclomatique de ce programme est :

C=10-9+2=3

Deux autres méthodes peuvent être utilisées pour obtenir C[10,13] .

C=π+1

π est le nombres de décisions trouvées dans le programme.

C=r (r est le nombre de régions de CFG)

Pour le programme précédent on a π=2 donc c=3 aussi.

La mesure de la complexité cyclomatique donne une indication sur la testabilité et la facilité de maintenance du module. Plus la valeur est élevée, le module demande plus d' effort pour le comprendre, le tester et le maintenir.

9.9.3 - **Mesures issues de la science logicielle d'Halstead**

Maurice Halstead [10,13] en 1976 a proposé un ensemble de métriques basées sur des modèles mathématiques pour mesurer la longueur du logiciel, le volume du système logiciel et la complexité du logiciel pour prédire le temps et l'effort de développement. Un programme est

vu comme un ensemble d'unités lexicales appartenant à deux catégories : opérateurs et en opérandes[10,13] .

n_1 =le nombre d'opérateurs uniques n_2 = le nombre d'opérandes uniques

n =le nombre total des tokens distincts

n^* est le nombre des Opérandes potentiel. Ce nombre représente le nombre de l'ensemble de valeurs minimales necessaire en cas de n'importe quelle implementation.

1.1 La longueur du logiciel

La longueur du logiciel est identifiée par le nombre N =N1 + N2
N1 : le nombre total d'opérateurs
N2 : le nombre total d'opérandes

Exemple[13] : soit le programme suivant

```
z =0;
while x >0
z = z + y;  x = x -1;
    end-while
    print(z);
```

Les Opérateurs sont :
= ; while/end-while > + -print()
Donc $\eta 1$ =8
Les Opérandes sont : = z 0 x y1
Donc $\eta 2$ =5

Le nombre total d'opérateurs $N1$ = 14 puisque on a 3 occurrences de =, ;(5) , >(1), + (1), - (1) , print (1), () (1) while et endwhile (1).

Le nombre total d'opérandes N2 = 11 puisque on a 4 occurrences de z, 0(2), x(3), y (1), 1 (1).
A partir de ces deux informations on trouve que la longueur du logiciel N =25

1.2 Estimation de la longueur

L'estimation de N à partir de $\eta 1$ et de $\eta 2$ est faite on utilisant la formule suivante :

$$N^{est} = n_1 + log_2(n_1) + n_2 + log_2(n_2)$$

Le calcul de N^{est} de l'exemple précédent donne 35.6.

1.3 Volume

Une estimation du nombre de bits nécessaires pour coder le programme.

$$V = N * log_2(n_1 + n_2)$$

Le volume du programme précédent selon cette mesure est :92,5

1.4 Volume potentiel V*

Représente la Taille minimale de la solution et elle est calculée comme suit : $V^* = (2 + n_2^*) * log_2(2 + n_2^*)$=

1.5 Niveau d'implémentation

Représente la distance entre l'implémentation actuelle et l'implémentation minimale.

1.6 Effort E

E=V/L et mesure l'effort intellectuel nécessaire pour implémenter le programme.

Temps T

Le temps requis pour implémenter le programme.

T=E/S avec s est le nombre de straud.

9.9.4 - **La mesure de flux d'information**

Henry et Kafura [10,13] modélisent la complexité HK_i d'un module i en fonction de nombre de flux d'information entrant «In_i» , de nombre de flux d'information sortant « Out_i » et de sa taille en nombre de lignes de code.

Elle est décrite par la formule suivante: $K_i = poids*(In_i*Out_i)^2$ où M est la complexité du flux d'information,. Le poids est fonction de la taille du composant et sa complexité structurelle.

La mesure totale *HK d'un programme composé d'un ensemble de modules correspond* à la somme des HK_i

Par exemple [13]

Si on suppose que le poids de chaque module=1. La mesure du programme présenté par la table suivante (Table 12) est HK=1110

Module	A	B	C	D	E	F	G	h
In_i	4	3	1	5	2	5	6	1
Out_i	3	3	4	3	4	4	2	6
HK_i	144	81	16	225	64	400	144	36

Table 12 Un exemple d'un programme [13] .

9.10 - **Les métriques proposées par Chidamber et Kemerer[27]**

Ce sont des métriques ayant pour objectif la mesure de la complexité de la conception des classes d'un logiciel orienté objet.

9.10.1 - **WMPC - Weighted Methods Per Class**

C'est le nombre de méthodes composant une classe.

WMPC est un indicateur de complexité.

Cette métrique peut être utilisé pour prévoir le temps et l'effort nécessaire pour développer et maintenir la classe. Un WMPC élevé indique que la classe peut contenir beaucoup de fautes et quelle est d'une compréhensibilité plus difficile.

9.10.2 - **DOIH - Depth OflnHeritance tree**

DOIH est définit comme le niveau de la classe dans la hiérarchie en posant que le niveau de la racine est 0.

Cette métrique indique la complexité en se basant sur la portée des ancêtres.

9.10.3 - **NOC - Number Of Children**

NOC est définit comme étant le nombre de descendants direct de la classe.

Justification théorique: NOC est un reflet de l'impact potentiel d'une classe sur ses descendants.

9.10.4 - **CBO - Coupling Between Object classes**

Définition : Nombre de liaisons entre une classe et toutes les autres classes du système (invocation de méthodes ou de variables).

CBO indique le degré d'interdépendance entre les composants du système.

9.10.5 - **RFC - Response For a Class**

C'est le nombre de méthodes (de la classe et d'autres classes) qui peuvent être appelées par une classe en cas de réception d'un message.

9.10.6 - **LCOM - Lack of COhesion in Methods**

C'est le nombre de paires de méthodes n'ayant pas des instances de variables de la classe en commun -(moins) le nombre de paires de méthodes ayant des instances de variables de la classe en commun. Une valeur faible indique une forte cohésion (une classe est cohésive si ses méthodes travaillent sur le même ensemble de données) et si elle est négative, LCOM est égale à zéro. Elle indique la qualité de la structure de la classe. Une faible cohésion indique Éventuellement que la classe gère différentes responsabilités.

9.11 - **MétriquesMOOD (Metrics for Object Oriented Design)[27]**

Ensemble de métriques pour mesurer les attributs suivants : Encapsulation, Héritage, Couplage et Polymorphisme

9.11.1 - **MHF - Method Hiding Factor**

Cette métrique évalue l'encapsulation et elle définit comme le pourcentage de méthodes cachées.

Valeurs recommandées : [10%,30%]

9.11.2 - **AHF - Attribute Hiding Factor**

Cette métrique évalue l'encapsulation et elle est définit comme le pourcentage d'attributs cachés.

Valeurs recommandées : [70%, 100%]

9.11.3 - **MlF - Method Inheritance Factor**

Cette métrique évalue l'abstraction et la fonctionnalité et elle est définit comme le pourcentage de méthodes héritées.

Valeurs recommandées : [65%, 80%]

9.11.4 - **AIF - Attribute Inheritance Factor**

Cette métrique évalue l'abstraction et elle est définit comme le pourcentage d'attributs hérités.

Valeurs recommandées : [50%,60%]

9.11.5 - **PF - Polymorphie Factor**

Cette métrique évalue la flexibilité et elle est définit comme le pourcentage de méthodes polymorphes par rapport au nombre total de méthodes potentiellement polymorphes.

Valeurs recommandées : [3,5%, 10%]

9.11.6 - **COF - Coupling Factor**

Cette métrique évalue L'interdépendance et elle est définit comme pourcentage de classes couplées aux autres classes autrement que par l'héritage.

Valeurs recommandées : [4%, 20%]

Comment choisir les mesures:

9.12 - GQM: (Goal Question Metric)

GQM est une approche guidée par les buts proposé par Basili pour mesurer les systèmes. Elle définit un modèle de mesure à trois niveaux[13].

1er niveau : ce niveau est un niveau conceptuel dans lequel les objectifs principaux d'un projet sont définit et énumérer (général ou de développement, de maintenance ou d'expérience)

2eme niveau : ce niveau est un niveau opérationnel. Le travail à ce stade consiste à produire pour chaque objectif, les questions permettant de savoir si cette objectif est atteint

3eme niveau : ceci est un niveau quantitatif ou on va décider de ce qui doit être mesuré pour répondre convenablement aux questions.

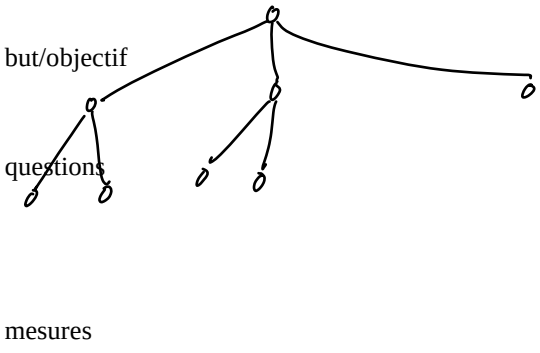


Figure 24 Le modèle GQM

9.13 - **Exercices**

Exercice 1 **:** Expliquer sous quelle condition une métrique est considérée comme valide.

Exercice 2 : Citez et expliquez une métrique de produit et une métrique de processus.
Exercice 3 : Citer quelques inconvénients de la métrique LOC.

Exercice 4 : quelle est l'échelle des métriques vues en cours.

Chapitre 10 - **Test logiciel**

10.1 - **Introduction**

Le test est une activité fondamentale pour améliorer la qualité du logiciel. On distingue deux types de test. Le test statique et le test dynamique.

Le test dynamique consiste à exécuter un programme pour s'assurer du fonctionnement correct du logiciel. Le test statique est un test qui consiste à analyser le code source afin de trouver les fautes par exemple : Lectures croisées / inspections. Le coût moyen de cette activité est :

• 1/3 en phase de développement du logiciel

• 2/3 durant la phase de la maintenance du logiciel

On va étudier dans ce chapitre le test dynamique. Nous commençons par la présentation du vocabulaire de ce domaine.

10.2 - **Vocabulaire :**

Selon le IEEE Standard Glossary of Software Engineering Terminology [IEEE, 1990]:

Défaillance : déviation de la sortie du système par rapport à l'attendu (décrit par la spécification).

Erreur : état interne erroné du système (i.e., valeurs prise par les variables, affectées aux registres, existant dans les tampons…etc) qui peut produire une défaillance. Normalement on ne doit pas arriver à cet état.

Faute : un défaut dans la conception du système. Sa source est humaine et peut Causer l'apparition d'une erreur.

Une erreur est la cause d'une ou plusieurs fautes. La défaillance est causée par la présence d'une erreur ou plusieurs.

Noter que l'implication entre ces concepts est potentielle et elle schématisée comme suit :

faute ⇒(peut produire) erreur (peut causer) ⇒ défaillance

Exemple[28]

Soit le code suivant :

int a, b, x, y, k, ...

if (a == b) {

... x = a - b;

}

if (k == 1) y /= x;

Si l ’instruction `x = a - b;‘ est fautive (par exemple le correct est, `x = a + b;’).

• Si a et b sont différent en cas d’une exécution, il n’y aura peut-être pas d’erreur(malgré la faute).

• Dans le cas contraire, si k != 1 lors de cette exécution, il n’y aura pas de défaillance (malgré l’erreur).

• Dans le cas contraire, on divise par 0 et on a effectivement une défaillance.

10.3 - **Définitions**

« Le test est l’exécution ou l’évaluation d’un système ou d’un composant par des moyens automatiques ou manuels, pour vérifier qu’il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus. » IEEE (Standard Glossary of Software Engineering Terminology)

« Tester, c’est exécuter le programme dans l’intention d’y trouver des anomalies ou des défauts. » G. Myers (The Art of Software testing)

«Testing can reveal the presence of errors but never their absence.» Edsgar W. Dijkstra (Notes on Structured Programming).

Les données utilisées pour tester un programme sont appelées jeu de test et consistent en un ensemble de cas de test.

Un cas de test est une paire (I, O, C). Le premier élément est l ’entrée du programme , l’élément O est la sortie attendue et C représente le contexte.

Par exemple pour tester un programme qui calcule la somme de deux entiers, on peut utiliser les cas de test suivants :

((3,4),7) ici 3 et 4 sont les entrées et 7 est la sortie attendue.

((-10,10),0)...etc.

L'objectif du test est de trouver les défaillances

10.4 - **Vérification and Validation(V&V)**

L'objectif du test est de faire la vérification et la validation du logiciel[1].

Vérification : est-ce qu'on est en train de développé correctement le produit.

Validation : est-ce que le produit développé est correct (par rapport à la spécification)

10.5 - **Les différents niveaux de test**

• **Tests de recette [1]:** test effectuer chez le client pour voir si le système est lui satisfaisant ou non.

• **Tests d'intégration système[1]:** : test de l'intégration du logiciel avec d'autres Logiciels

• **Tests système[1]:** : tester pour s'assurer que le système délivré fonctionne correctement.

• **Tests d'intégration [1]:**: test effectuer pour s'assurer que les composants interagissent correctement.

• **Tests Unitaires [1]:**: tests élémentaires des composants logiciels (une fonction, un module, ...).

•**Tests de régression [1]:**: réexécution des tests sur la version modifiée du logiciel pour vérifier que la modification n'a pas causé de problèmes.

10.6 - **Difficulté de test**

Un test exhaustif est un test qui consiste à exécuter tous les cas de test possible. Ce type de test est généralement impraticable. Par exemple [13]le test exhaustif d'un programme qui calcule la somme de deux entiers de 32 bits chacun, demande l'exécution de $2^{32}*2^{32}$ cas de test. Si chaque cas de test prend 1ms alors le processus de test prendra 2^{64} ms . Par conséquent le test consiste à exécuter une partie de tous les cas de test possibles.

10.7 - **Psychologie de test**

Le test est plus effectif s'il est conduit par des personne qui n'ont pas écrit le code. Parce que généralement le producteur d'un produit le voit comme parfait. La programmation est une activité constructive alors que le test est une activité destructive (on cherche le maximum des défaillances on supposons que le programme est imparfait).

Un bon testeur est un testeur capable de concevoir des cas de test effectifs. Ces cas de test on le potentiel de produire le maximum de défaillances possibles. Par exemple pour le programme qui calcule le produit de deux nombres entiers en utilisant l'addition. Le jeu de test (2,3,6), (-3,-7,21), (0,5 ,0), (-9,3,27) est efficace par rapport au jeu de test suivant malgré que le dernier est large par rapport au premier ensemble.

(1,2,2) , (3 ;4 ;12),(4 ;8 ;32), (100,200,20000),(40,10,400),(1000,0,0)

10.8 - **Types de test**

Deux questions font face aux testeurs[9,8] :

Comment choisir les cas de test(sélection)

Quand est ce qu'on arrête le test c.à.d. quel est le nombre des cas de test nécessaire pour tester le programme (critère d'arrêt).

La sélection des cas de test (jeu de test) peut être effectuer en se basant sur la spécification et on parle de black box testing.

En se basant sur le code source et on parle du white box testing.

En peut aussi combiner les deux, white box testing.

10.9 - **Black box testing**

Ce test est aussi nommé test fonctionnel[9,8] . Les cas de test sont conçus à partir de la spécification de programme.

Les deux principales approches utilisées pour concevoir ces cas de test sont [9,8] :

1-Partitionement en classes d'équivalence

2. Analyse des valeurs frontières

10.9.1 - **Partitionnement en classes d'équivalence**

Dans cette approche[9,8] , le domaine des entrées est divisé en un ensemble de classes équivalentes. La division est faite de telle sorte que le programme se comporte de la même manière pour toutes les entrées appartenant à la même classe. Les classes sont construite en examinant les entrées et aussi les sorties de programme. Dans cette approche on sélectionne de chaque partition une valeur représentative. Cette valeur assure la couverture de toutes les valeurs de la même classe.

Guide pour construire les classes :

On détermine deux types de classes : classe des entrées valides et classe des entrées invalides.

Si les éléments de la classe valide ne sont pas tous traités de la manière par le programme, on partitionne cette classe en des petites classes.

Par exemple pour le programme qui calcule la valeur absolue d'un entier on peut diviser l'entrée en 3 classes

1. Classe des valeurs valides [minint, maxint] et cette classe est divisée en 3 classes
   - Classe des valeurs valides positives de]0, maxint]
   - Classe des valeurs valides négatives de [minint, 0[
   - Classe qui contient la valeur {0}
2. Classe des valeurs invalides -infini, minint[, et]maxint, +infini[.

Un exemple de jeu de test sera (-5,5), (6,6),(200000000,valeur erronée) ,(-9999999999999,valeur erronée), (0,0)

10.9.2 - **Analyse des valeurs frontières**

Appelé aussi test aux limites[9,8] . A partir de l'expérience nous savons que les valeurs aux limites (valeurs de bord) qui se trouve entre deux classes ne sont pas gérées adéquatement par les programmeurs. Le programmeur peut utiliser improprement < à la place de <= ou <= à la place < . Un exemple de ces valeurs :

•Valeurs très grandes

•Valeur minimale de boucle, nulle, négative, maximale

•Données non valides

•Etc…

Pour créer des cas de test efficaces on doit ajouter ces valeurs dans le jeu de test. Par exemple

Si une variable d'entrée e entière prend des valeurs dans le domaine [1..30], on doit tester le programme pour e={0 ,1,30,31}

10.10 - **Test structurel**

La première question à répondre est : pourquoi un autre type de test et est-ce que le test fonctionnel ne suffit pas ?. La réponse à cette question est oui. Les études empiriques montrent que le test fonctionnel ne touche (exécute) que 30% du code[27].

Ce test est un test basé sur la structure de code source pour concevoir les cas de test. On utilise les informations sur comment le système est conçu et implémenté pour dériver les cas de test.

Dans ce test des critères de couverture sont utilisés pour arrêter le processus de test.

On distingue deux types de test structurel :

Test basé sur le flot de contrôle (control flow testing) et le test basé sur le flot de données (data flow testing)[8 ,9].

10.10.1 - **Test basé sur le flot de contrôle**

Ce test est basé sur un modèle de code dit un graphe de flot de contrôle.

La figure 24 présente le CFG du programme suivant (noter que ce n'est la meilleure écriture) : le programme prend en entré un entier et affiche son signe (+ ;- ou null). Les nœuds contiennent les numéros de chaque instruction.

Read(a)

Signe='negatif'

If (a>0) signe='positif' ;

If (a==0) signe='null' ;

Print(signe)

End.

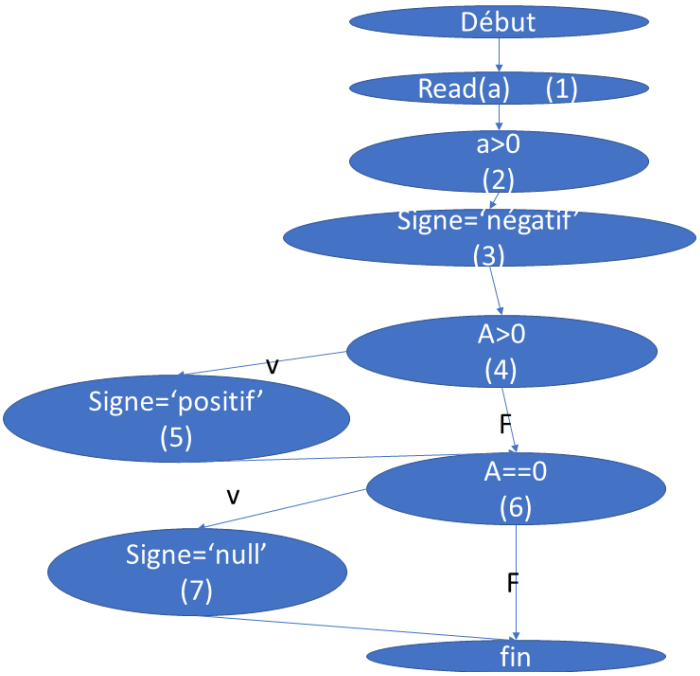


Figure 25 Le CFG du programme précèdent.

C0 : Couverture des instructions

Puisque chaque instruction est source potentiel de problème, il faut concevoir les cas de test de telle sorte que chaque instruction soit exécutée au moins une fois.

De point de vue CFG, il faut passer par chaque nœud du graphe CFG.

Exemple :

Cas de test|instruction	1	2	3	4	5	6	7
(3 ,+)	*	*	*	*	*	*	
(0 ,null)	*	*		*		*	*

C1 : Couverture des décisions (branches)

C'est un test plus fort que C0. Le principe est de concevoir des cas de test de telle sorte que chaque décision soit évaluée à true et à false. Ceci veut dire qu'on doit passer par chaque arc du CFG au moins une fois.

Cas de test|arc	Début-1	1-2	2-3	3-4	4-5	4-6	5-6	6-7	6-fin	7-fin
(8 ,+)	*	*	*	*	*		*		*	
(0 ,null)	*	*	*	*		*		*		*

Couverture des chemins

Ce test est plus fort que C1. Le principe est de Concevoir les cas de test de telle sorte qu'on passe par chaque chemin qui compose le CFG au moins une fois. Un chemin représente une exécution possible du programme. Mais il y 'a des limites :

1. Le nombres des chemins peut être trop élevé ce qui rend l'application pratique impossible.
2. Le problème de détection des chemins non exécutables est théoriquement un problème indécidable. Un chemin non exécutable est un chemin qu'on ne peut pas le sensibiliser (on ne peut pas passer par lui) quel que soit le cas de test utilisé. Par exemple le chemin début-1-2-3-4-5-6-7-fin est non exécutable parce qu'on ne peut pas avoir un nombre qui est à la fois >0 et =0.

Exemple

Le graphe de la figure contient 4 chemins. 3 chemins exécutables est un chemin non exécutable qui est début-1-2-3-4-5-6-7-fin.

Les chemins exécutables sont :

début-1-2-3-4-5-6-fin

début-1-2-3-4-6-7-fin

début-1-2-3-4-6-fin

On peut sensibiliser ces chemins par les cas de test suivants : (5,+), (0,null),(-8, négatif)

10.10.2 - **Test basé sur le flot de données (Data Flow Testing)**

Ce test utilise le CFG pour détecter les anomalies liées aux circulations de données. Ces anomalies sont détectées en analysons l'association entre valeurs et variables. Par exemple :

On peut analyser pour savoir :

1. Quelles sont les variables utilisées mais non initialisées.
2. Quelles sont les variables initialisées mais non utilisées.

Définition et utilisation d'une variable

Une occurrence de variable dans un programme est considérée comme **une définition** de cette variable si une valeur est affecté à cette occurrence.

Une occurrence de variable dans un programme est considérée comme **une utilisation** de cette variable si sa valeur est référencée à cette occurrence.

Une utilisation est une utilisation de type p-use si la variable est utilisée dans un prédicat et sa valeur est utilisée pour décider du chemin à exécuter.

Une utilisation est une utilisation de type c-use si la variable est utilisée pour calculer une valeur à affecter à une variable ou comme une valeur de sortie.

Par exemple pour l' instruction

S: a=b+c;, DEF(*S*)={*a*}, c-use(*S*)={*b, c*}, p-use{s}={}

Un chemin def-free : est un chemin sans définition, commence de la définition d'une variable à une utilisation sans passer par d'autres définition de cette variable.

10.10.3 - **Critères de test flot de données[13, 22]**

dcu : chaque définition doit être reliée à un c-use en aval par un chemin sans définition
dpu : chaque p-use doit être reliée à une définition en amont par un chemin sans définition
du : combinaison des deux précédents
all-du-paths : le plus exigeant, qui veut que tous les chemins possibles entre une définition et une utilisation doivent être sans définition

Exemple

Soit le problème de type de triangle suivant :

Le programme prend 3 valeurs entières, les interprète comme longueur de côtés de triangles et affiche son type.

Un exemple de code de ce problème[13] :

noued	Instructions
a	read a,b,c
b	type=‘‘scalène’’
c	if(a==b||b==c||a==c)
d	type=‘‘isocèles’’
e	if(a==b&&b==c)
f	type=‘‘équilatéral’’
G	if(a>=b+c||b>=a+c||c>=a+b)
h	type=‘‘not a triangle’’
I	if(a<=0||b<=0||c<=0)
J	type=‘‘bad inputs’’
K	print type

Le CFG de ce code est présenté par la figure 25.

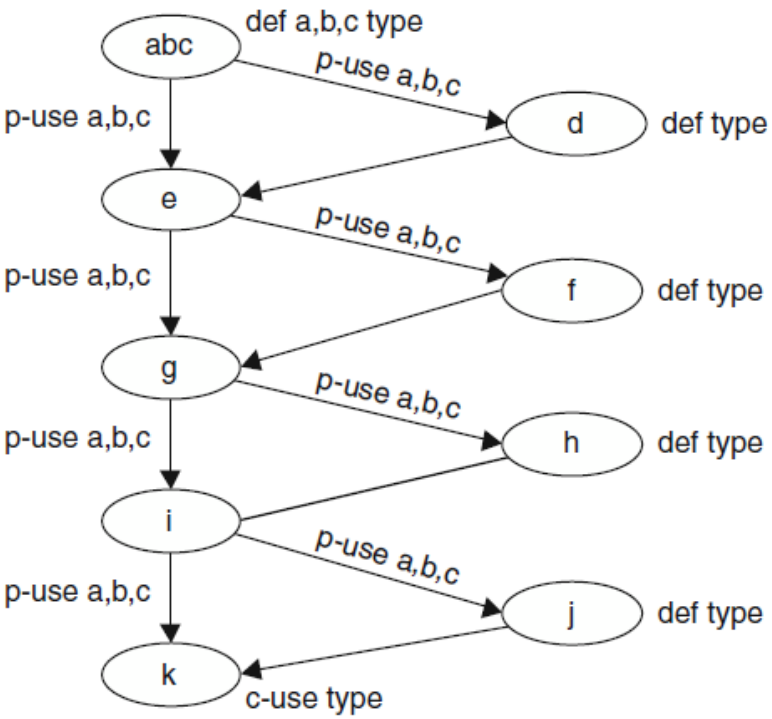


Figure 26 Le CFG du programme du triangle[13].

dcu [22]: le seul c-use porte sur type et se trouve au nœud K

– Depuis ABC jusqu'à K : chemin ABCEGIK

– Depuis D jusqu'à K : chemin DEGIK

– Depuis F jusqu'à K : chemin FGIK

– Depuis H jusqu'à K : chemin HIK

– Depuis J jusqu'à K : chemin JK

– dpu [22]: les p-use portent sur les variables a,b et c qui ne sont définies que dans le nœud ABC

– Depuis ABC jusqu'à l'arc ABC-D

– Depuis ABC jusqu'à l'arc ABC-E

– Depuis ABC jusqu'à l'arc E-F

– Depuis ABC jusqu'à l'arc E-G

– Depuis ABC jusqu'à l'arc G-H

– Depuis ABC jusqu'à l'arc G-I

– Depuis ABC jusqu'à l'arc I-J

– du : tous les tests de dcu et de dpu combinés

– all-du-paths : mêmes tests que pour du

10.11 - **Test des programmes orienté objets**

Un programme développé selon l'approche OO est formé d'un ensemble d'objets qui interagissent pour réaliser des objectifs bien définis. Cette approche est basée sur certains concepts tels que l'encapsulation, abstraction, héritage, polymorphisme… etc pour produire un logiciel de qualité. Mais le test de ces programmes est plus difficile et couteux s'il est comparé avec le test les programmes procédurales.

Pour les programmes OO, on peut concevoir différents types de cas de test en se basant sur leurs modèles de conception. Ce type de test est généralement nommé greybox testing.

Test basé sur le modèle d'état (State model-based testing)[27]

Couverture d'état(State coverage):chaque méthode d'un objet est testé à chaque état possible de cet objet..

Couverture de transition (**State transition coverage):** on test est-ce que toutes les transitions présenté par le modèle d'état fonctionnent correctement.

Couverture des chemins de transition (State transition path coverage) :on test tous chemins de transition .

Test basé sur les cas d'utilisation[27]

Couverture de scenario pour chaque cas d'utilisation, la séquence principale et toutes les alternatives doivent être testées

Test basé sur le diagramme de classe[27]

Test des classes dérivées : toutes les classes dérivées doivent être instanciées et testée. Les méthodes de la classe dérivée et les méthodes héritées doivent aussi être testées.

Test des associations : toutes les associations doivent être testées.

Test des agrégations : on doit créer et testet les agregation.

Test basé sur le diagramme de séquence[27]

Couverture de méthodes : on doit couvrir toutes les méthodes figurant dans un diagramme de séquence.

Couverture du chemin du message : tous les chemins de messages formés à partir du diagramme de séquence doivent être testés.

10.12 - **Exercices**

Exercice 1

Quelle(s) est (sont) la (les) différence(s) entre test boite noire et test boite blanche

Exercice 2

Construire les cas de tests fonctionnels pour tester le programme qui prend en entrée un entier x et affiche la valeur de y / y= $\sqrt{\frac{1}{x^2-1}}$

Exercice 3

Construire les cas de tests fonctionnels pour tester le programme qui prend deux entiers et un caractère c∈{*,+} et affiche le produit de ces deux nombres en cas ou c='*' et la somme en cas ou c='+'.(utiliser l'approche partitionnement en classes d'équivalences et l'approche test aux limites)

Exercice 4

Construire les cas de tests fonctionnels pour tester le programme qui recherche un élément donné par l'utilisateur dans un tableau de dimension n.

Exercice 5

Soit le programme suivant :

```
//le programme calcul  le nombre des entiers impairs ou positifs
//Dans un tableau

Public static  int  impairsoupositifs(int [] x){
Int count=0;
For (int i=0; i<x.lenght;i++)
{
If (x[i]%2==1 || x[i]>0){
Count++;}
}
Return count
}
//test : x=[-3,-2,0,1,4]; sortie attendue =3
```

1. Corriger la faute.

2. Si possible, donnez

- Un cas de test qui n'exécute pas la faute.
- Un cas de test qui fait l'exécution la faute mais ne produit pas une erreur.
- Un cas de test qui produit une erreur mais pas une défaillance
- Un cas de test qui produit une erreur et une défaillance

Exercice 5

Soit le programme suivant qui calcule qui calcule le plus grand commun diviseur (pgcd) de deux entiers positifs ou nuls m et n.

```
int pgcd(int m, int n)
 { int temp; while(n > 0)
{     if(n > m) {
```

```
 temp = n;
n = m;
m = temp; }
temp = n;
 n = m-n;
m = temp; }
return m;
}
```

9.1 Calculer la complexité cyclomatique de ce programme.(1pt)

Construire les cas de tests qui satisfait les critères c0, c1 , dcu.

Chapitre 11 - **Solutions de quelques exercices**

11.1 - **Solutions de quelques exercices**

Solution d'exercice 1.1

Analyse du marché, planification de projet, estimation des coûts, spécification des besoins relecture des besoins, conception architecturelle, conception détaillée, revue de la conception, Implémentation, test système, test d'acceptation et mise en œuvre.

Solution d'exercice 2.1

Projet : 1 ,2 ,4,6,9,11, 12

Opération :3, 5,7,8,10

Solution d'exercice 3.1

Application de la formule : effort $=2.4*(3)^{1.05}=7.61$ h/m

Delai$=2.5*(7.61)^{0.38}=14.46$

Solution d'exercice 3.3

Entrée : la saisie d'un nombre, enregistrement de successeur

Sortie : affichage de successeur

Fichier : fichier de successeur

Et on n'a pas des fonctions de type Interface et interrogation.

Charge=4*4=16 PFs.

Solution d'exercice 3.5

Charge=4*taille+5

Solution d'exercice 4.1

La programmation (WBS), 34 niveaux de jeu (PBS), conception graphique (WBS), 4 joueurs (PBS), bons éléments graphiques (PBS), test (WBS), compatible avec les machines MAC et PC(PBS), une bataille de type « boss battle » au niveau final du jeu (PBS).

Solution d'exercice 4.2

a)

Tâche	Durée	Dépendances	Date début		Date fin		Critique ?
			Tôt	Tard	Tôt	Tard	
init	4		0	0	4	4	*
a	8	init	4	5	12	13	
b	10	init	4	4	14	14	*
c	8	a,b	14	14	22	22	*
d	9	a	12	13	21	22	
e	5	b	14	21	19	26	
f	3	c,d	22	22	25	25	*
g	2	d	21	23	23	25	
h	4	f,g	25	25	29	29	*
i	3	e,f	25	26	28	29	

Solution d'exercice 5.2

VMA= 0.5*20000+0,5*0=10000

VMA=0.25*20000+2000=7000

Recruter l'expert est un bon choix.

Solution d'exercice 5.3

Risque	Réduction de prob	Réduction d'impact
Matériel non disponible	Accélérer le développement du matériel	Concevoir un simulateur
Spécifications incomplètes	Approfondir les contrôles des spécifications	
Utilisation de méthodes spécialisées	Former l'équipe, engager des experts	
Départ d'une personne clé	Augmenter les salaires	Recruter d'autres personnes
Sous estimation des efforts	Faire appel à des experts	Estimations fréquentes. Respecter les délais
Le seul client fait faillite		Chercher d'autres clients

Solution d'exercice 6.2

Le taux des erreurs trouvées par KLOC et faults-found/reviewer-hour

Solution d'exercice 7.2

BAA=56*10=560, VA=56, CR=1000, VP=4*10=40 ,EC=56-1000 , ED=56-40

Annexe

Contenu de programme du module GL2

UF2 : Génie Logiciel 2

Contenu de la matière :

I- Processus de développement logiciel

1. Motivations

1.1 Qualités attendues d'un logiciel

1.2 Principes du Génie Logiciel

1.3 Maturité du processus de développement logiciel

2. Cycle de vie d'un logiciel

2.1 Composantes du cycle de vie d'un logiciel

2.2 Documents courants

2.3 Modèles de cycle de vie d'un logiciel

2.4 Modèles de processus logiciels

II- Conduite de projets

3. Gestion de projets

3.1 Pratiques critiques de la gestion de projet

3.2 Analyse de la valeur acquise

3.3 Suivi des erreurs

4. Planification de projets

4.1 Organigramme technique

4.2 La méthode PERT

4.3 Autres modèles

4.4 Estimation des coûts (Exp : Modèle COCOMO).

5. Assurance qualité

III- Techniques du Génie Logiciel

6. métriques

6.1 Métriques de Mac Cabe

6.2 Métriques de Halstead

Références

1. Roger Pressman, Bruce Maxim, Software Engineering: A Practitioner's Approach. *McGraw-Hill Higher Education; 8e édition* (2014).
2. Ghezzi, C., M. Jazaueri and D. Mandrioli , Fundamentals of Software Engineering, *Prentice-Hall of India Private Limited, New Delhi.*1994.
3. Gilb, T., Principles of Software Engineering and Management, Reading, Mass: Addison-Wesley.1988.
4. I. Sommerville, Software Engineering, Seventh Edition, *Addison-Wesley Publ. Ltd., Edinburgh Gate, England,* 2004.
5. Pfleeger, S. L. , Software Engineering: Theory and Practice, *Pearson Education, Inc.,Second Edition*.2001.
6. Boehm, B.W., Software Engineering, IEEE Trans. Computers, pp. 1226–1241.1976.
7. Boehm, B.W. and R. Ross, Theory W Software Project Management Principles and Examples, *IEEE Transactions on Software Engineering, Vol. 15, No. 7, pp. 902–916*.1988.
8. Jorgensen, P. C., Software Testing A Craftsman's Approach, Boca Raton: *CRC Press, Second Edition*.2002.
9. Myers, G. J. , The Art of Software Testing, *Wiley-Interscience, NY*.1979.
10. Norman Fenton and James Bieman, Software Metrics: A Rigorous and Practical Approach, Third Edition.2014.
11. Y. Wang and G. King, Software Engineering Processes: Principles and Applications. *CRC PressISBN: 9780429177491*.2000.
12. Chantal Morley, Management d'un projet Système d'Information - 6ème édition : Principes, techniques, mise en oeuvre et outils. *ISBN 9782100535514, Dunod,* 2008
13. David Gustafson: schaum's outline software engineering. ISBN-10: 9780071377942 *Schaum's Outlines,* 2002.
14. Rita mulcahy : PMP Exam Prep: Accelerated Learning to Pass PMI's PMP Exam, *RMC Publications*, 2013.
15. Afitep : Dictionnaire de management de projet. *Edition AFNOR, Paris,* 2004.
16. André Claude : La gestion financière des chantiers. *Edition Le moniteur, Paris,* 1996.
17. Jean le Bissonnais : Management de Projet de A à Z. *Edition AFNOR, Paris,* 2003.
18. Project Management Institute : Guide du corpus des connaissances en management de projet. *Editeur, PMI Publication, quatrième édition, Newtown Square, Pennsylvania,* 2008.
19. Anne-Marie Hugues : Génie Logiciel, 2002 http://www.polytech.unice.fr/~hugues/GL/

20. Jennifer Greene, Andrew Stellman Head First Pmp: A Learner's Companion to Passing the Project Managemen Professional Exam. *O'Reilly Media, Inc, USA; 4th ed. Edition.*2018.
21. https://agilemanifesto.org/
22. Piere gérard. Génie Logiciel : principes, méthodes et techniques. https://lipn.univ-paris13.fr/~gerard/docs/cours/gl-cours-slides.pdf
23. NadiaTawbi http://www2.ift.ulaval.ca/~tawbi/Cours/Metrics/chapitre1/chapitre1.html
24. Yann-Gaël Guéhéneuc IFT3903 Qualité du logiciel et métriques http://igt.net/~ngrenon/UdeM/cours/IFT3913/Notes/03_qualite.pdf
25. Rajib Mall. Fundamentals of Software Engineering, PHI Learning; 2014.
26. G. Rubino, 2006 --- Introduction à la sûreté de fonctionnement. http://www.irisa.fr/armor/lesmembres/Rubino/myPages/MATOS%20TEACHING/intro%20a%20la%20SdF.pdf
27. Rex Black, Erik Van Veenendaal, Dorothy Graham. Foundations of Software Testing – *ISTQB® Certification, 3rd ed., (2012), Cengage Learning*.
28. Luis Fernández Sanz. Software quality: concepts and evidences .2008.