Sommaire

1. COURS : Introduction au Génie Logiciel

1.1. / Données économiques

1.1.1. / Evolution de la demande en logiciels

La demande en développement logiciel est en constante évolution car tout ce qui est analogique devient numérique. C'est l'essor de l' informatique et du traitement des données par des calculateurs.

Exemple : gestion électronique de documents, multimédia, traitement de l'image de synthèse (médecine), compression de données, appareils photo, télévision, bureautique, jeux, …

Les services en informatique (maintenance et pérennisation) sont en augmentation constante. On a de plus en plus d'automatisation et de gestion de gros systèmes. Afin de structurer, planifier, quantifier efficacement les travaux de développement logiciel, ont utilise les méthodes définies par le génie logiciel.

1.1.2. / Volume, délai, durée de vie, effort

Dans le domaine du développement logiciel, on utilise 3 critères dont les définitions sont les suivantes :

Le volume : c'est la taille du code source mesuré en KLS (Kilolignes de Code Source).
Le délai : c'est le temps de réalisation.
L'effort : en homme.année ou homme.mois. 3 personnes pendant 18 mois = 54 hm = 4,5 ha.

Le tableau suivant présente différents exemple d'applications avec leurs critères :

Type d'applications	*Exemple*	*Effort (ha)*	*Volume (KLS)*	*Délai (année)*
COMPILATEURS	Pascal, C Cobol, Fortran	10 80	20 à 30 100 à 200	1 à 2 2 à 3
BASE DE DONNEES	Oracle	300 à 500	300 à 500	5
TEMPS REEL	Navette spatiale	> 1000	2200	6 ans (Langage HAL)
	SafeGuard (1970) (système de défense antibalistique US)	5000	-	7 (Langage TLE)
	SABRE (système de réservation Am. Airlines (IBM))	955	960	10 (MTTF = 55 heures.)
Système d exploitation.	MVS, VMS, GCOS 7	2500 à 5000	5000 à 15000	5 (1ere version) Durée de vie prévue : 15 à 20.
Graphisme 2D, 3D.	Développements en Fortran	50 à 150	> 200	-
Intelligence Artificielle.	Lisp, Prolog Systèmes experts	10 à 20 20 à 30	-	-
Atelier de Génie Logiciel (c/c++)	-	150 à 200	-	-

Exemples d'applications avec leurs coûts.

1.1.3. / Evolution des coûts

Le prix du hardware descend et le prix du logiciel monte.
Augmentation de l'investissement en maintenance.
Apparition de la ré ingénierie.

} Le coût le plus important est le coût humain.

1.1.4. / Nature du risque logiciel

La généralisation de l'informatisation des données peut présenter plusieurs risques plus ou moins majeurs :

- Risque majeur : la sûreté du fonctionnement.
- Risque humain : le pilotage d'un avion (A320), contrôle aérien, contrôle ferroviaire.
- Risque économique : les transactions boursières, lignes téléphoniques (AT&T côte Ouest).
- Risques sociaux : le logiciel Socrate (SCNF), la confidentialité des cartes bancaires, les virus.

Voici quelques exemples de dysfonctionnements informatiques avec leurs causes et leurs conséquences :

***Temps réel embarqués* :**

* Mission Vénus : affichage de la distance : 500 000 kms au lieu de 5000 kms. La ',' était remplacée par un '.'.
* F16 déclaré "sur le dos" au passage de l'Equateur.
* Columbia : faux départ. Manque de synchronisation entre deux calculateurs.
* Exocet argentin non déclaré ennemi par la Marine anglaise lors de la guerre des Malouines.
* Fusée russe qui allait à Hambourg au lieu du Pôle Nord. Problème de signe ➔ virage à 180°.

***Temps réel au sol* :**

* Fusée à l'eau. Mauvaise unité.
* Métro fantôme (San Francisco).
* Décès d'un malade. Erreur de monitoring.
* Lever de lune considérée comme un OVNI ➔ Déclenchement du système de défense balistique US.
* Vallée du Colorado inondée. Mauvaise modélisation du temps d'ouverture d'un barrage.

Selon une enquête auprès de 55 entreprises on a :

53 % du budget informatique qui est dédié à la maintenance dont :

34% maintenance évolutive.
10% maintenance adaptative (portage, …)
17% maintenance corrective.
16% maintenance perfective.
6% assistance utilisateurs.
6% contrôle qualité.
7% organisation et suivi.
4% divers.

63% des entreprises ont une maintenance "dédiée".
56% des entreprises appliquent des méthodes formelles pour les priorités de maintenance.
45% des entreprises ont des méthodologies de maintenance.

1.2. / Typologie et criticité

1.2.1. / Classification

On classe les programmes informatiques sous 3 types qui sont les S, les P, et les E programmes.

1.2.1.1./ Les S-programme

Les spécifications parfaitement définies et stables c'est à dire qui n'évoluent pas.

Exemple : codage d'une spécification : $\sqrt{A} = \lim_{n\to\infty}(X_{n+1} = \frac{1}{2}(X_n + \frac{A}{X_n}))$

L'innovation est nulle et les tests sont réduits. Il n'y a pas de validation. Le gain est de 2 à 5 par rapport à un cycle normal. Les spécifications sont définies et ne peuvent pas bouger.

1.2.1.2./ Les P-programme

Ils correspondent à la recherche d'une solution optimale d'un problème. C'est l'adhérence à des éléments qui rendent les spécifications fluctuantes et instables.

Exemple : un compilateur. Il permet de compiler une infinité de programmes écrits dans un certain langage à destination d'une certaine architecture ou d'une infinité d'architectures (WINDOWS, UNIX, APPLE, etc...).

Conception et réalisation : trouver le meilleur compromis entre la vitesse d'exécution du compilateur et qualité du code généré.

➔ Optimiser une fonction de coût non triviale.
➔ Forte innovation.
➔ Choix antinomique (vitesse/qualité).

Les tests : Ils sont très complexe car ils doivent vérifier une multitude de scénarios combinatoire On les traitent suivant deux modes de fonctionnement :

- mode "debug" : compilation rapide.
- mode "optimisé" : grande qualité du code généré, d'où exécution rapide de l'application.

Une des spécification est la vitesse d'exécution. L'optimisation du coût est alors complexe. Ce que l'on va devoir résoudre c'est le compromis entre l'espace et le temps.

1.2.1.3./ E-programme (EMBEDDED)

C'est le logiciel embarqué. Il réagi avec l'environnement. Lors de la conception, il faut essayer d'isoler l'environnement strict de l'application. L'expression des besoins n'est jamais close, et les spécifications évoluent constamment.

Cependant, il faut arrêter des choix en sachant qu'ils pourront être reconsidérés plus tard. Il faut concilier stabilité du modèle initial avec son aptitude à s'enrichir vis à vis de l'environnement. D'autre part, il faut également considérer les délais de restitution des différentes versions.

Exemple : réservation de places pour un transport collectif, le contrôle aérien, les Interfaces Homme / Machine (IHM), le poste de pilotage d'un avion de ligne.

1.2.2. / Progiciel et programme "clé en main"

Définition 1 : Le **progiciel** (MULTIUSER) : c'est une application qui fait l'objet d'un (très) grand nombre de copies (logiciel générique). Plusieurs versions successives. Il est soumis à la concurrence.

Contrainte : **Respecter les délais de livraison.**

Définition 2 : Le **logiciel "clé en main"** (SUR MESURE) : C'est une application dont on créé un (très) petit nombre de copie (logiciel spécifique). L'estimation des délais et coûts est hasardeuse. Il y a un grand risque technologique car le matériel client est peu connu ou est indisponible…

Contrainte : **Maîtriser les coûts.**

1.3. / Le problème fondamental du Génie Logiciel

1.3.1. / Le problème de l'erreur

Définition : **Le Génie logiciel** est un ensemble de méthodes, de moyens techniques, industriels et humains à réunir pour construire des logiciels sûrs (comportement sûr et reproductible, erreur ou pas), conviviaux (donc évolutifs), adaptatif (IHM et paramétrages), et économiques. On doit avoir le coût le plus bas en réalisation et en maintenance.

Critère CQFD : **C**oût, **Q**ualité, **F**onctionnalité, **D**élai.

On constate deux faits de causes :

1 / On a de nombreuses erreurs même après de nombreuses vérifications.

Exemple : cas de la navette spatiale : . En embarqué : 0,11 erreur/kls pour 500kls.
. Au sol : 0,4 erreur/kls (1500 kls).
➔ plus de 50 erreurs.

2 / L évolutivité problématique des grands logiciels.

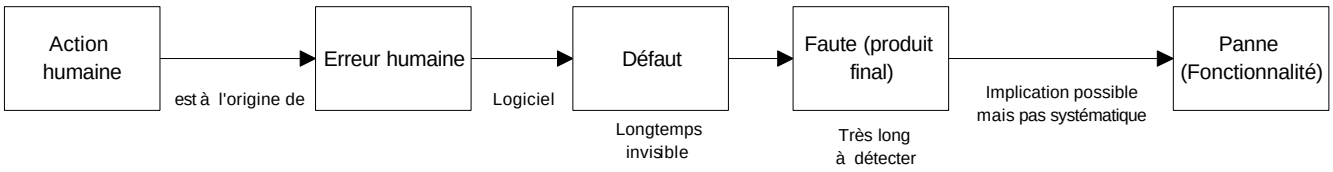
Sur une grande durée, le coût d'évolution approche le coût du développement. Les différentes modifications "détruisent" la structure logique initiale. On a une refontes complètes et prématurées du logiciel, d'où un coût très élevé pour éviter une instabilité et pour pérenniser le programme.

Les moyens humains permettent à l'aide de techniques industrielles associées au génie logiciel de spécifier, concevoir, développer, maintenir, et distribuer un programme.

La fiabilité de la communication humaine implique certaine régles de base :

* Nécessité d'une activité logico-mathématique (opposition logique/bon sens).
* Généralisation et constructions d'abstractions.
* Modélisation dynamique des systèmes.
* Comprendre et être compris.
* Traduction (code).
* Perception de la complexité combinatoire.

Les erreurs humaine suivent une chaîne qui est la suivante :



Il y a plusieurs niveaux à la conséquences d'un défaut :

1/. **Critique** : le programme ne pourra s'exécuter (sa mission est interrompue).
2/. **Sérieux** : réponses fausses.
3/. **Modéré** : perte de temps et gêne de l'utilisateur.
4/. **Tolérable** : peut être contourné.

On désigne par les terminologies suivantes la mesure d'un défaut :

MTTR : Mean time to repair (Temps moyen pour réparer).
MTTF : Mean time to fealure (Temps moyen jusqu'à la panne).

1.3.2. / Nature et fonction du logiciel

Définition : Le **programme** est une pensée exécutable et nouvelle, qui agit directement sur le monde extérieur, et qui diffère de la pensée communiquée à des tiers car le contenu est enfermée dans une machine.

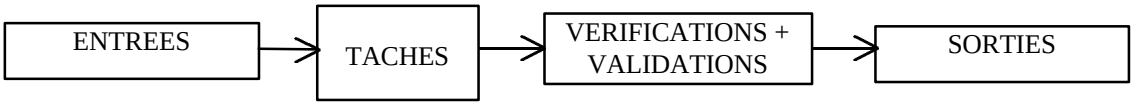
Le cerveau humain fait appel au sens commun, ce qui n'est pas le cas de la machine. Le génie logiciel doit apporter le contrôle de la sémantique du programme, c'est à dire la correspondance entre la syntaxe et le monde réel.

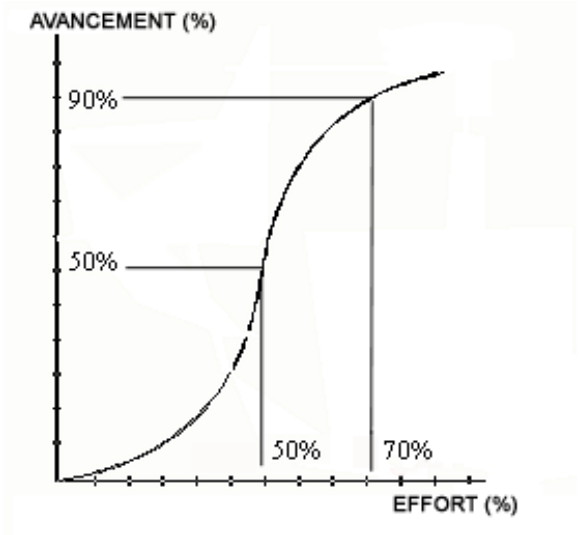
1.4. / Modèle de développement : le cycle de vie du logiciel

1.4.1. / Les différentes phases du cycle de vie - Processus QUALITE

1980 : décomposition des processus de développement en processus élémentaires.

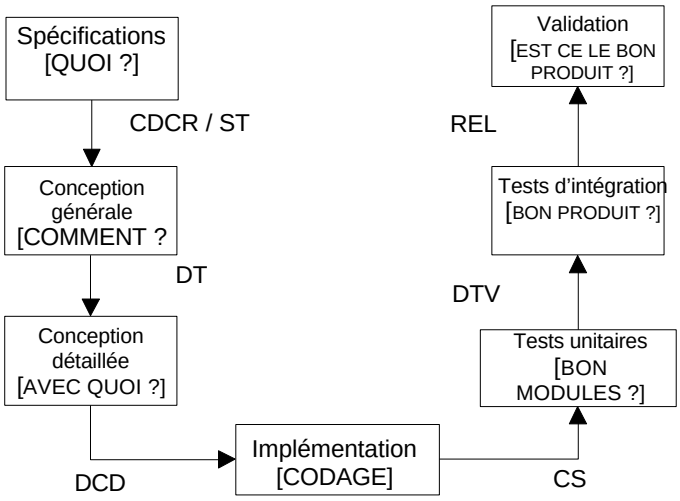
Paradigme ETUS :





Courbe d'avancement d'un projet.

(interprétation : 10% du W restant vont réclamer énormément d effort en fin de projet)



Le cycle de vie du logiciel (V).

Légende :

CDCR/ST : cahier des charges / Spécification Technique.
DT : Documentations techniques.
DCD : Dossier de conception.
CS : code source.
DTV : dossier de test de validation.
REL : Rapport d essais logiciel.

Spécifications : "Qu'est-ce qu'on veut faire?"

But : définir l'ensemble des caractéristiques externes (côté utilisateur) et internes (côté conception).

Documents : le dossier de spécifications (reprise et re formulation détaillée du cahier des charges), Le cahier de recettes (= description des tests de validation), une esquisse du manuel utilisateur.

Contrôles : la revue de spécifications qui doit permettre de chercher l'erreur(s).
1er tour de table : ce qui a choqué.
2eme tour de table : analyse du document page par page.

Conception générale : "Comment faire?"

But : déterminer l'architecture générale du logiciel et préparation des tests d'intégration, découpe en tâches, fonctionnalités regroupées en modules, interfaces de communication entre les modules, références au dossier de spécification.

Documents : le dossier de conception générale, le cahier d'intégration (description des tests d'intégration), la révision du manuel utilisateur.

Contrôles : la revue de conception générale qui présente les différents choix possibles : détermination des risques, des moyens à mettre en place, avantages et inconvénients.

Conception détaillée : "Comment faire exactement?"

But : s'assurer que le produit peut être testé et en indiquer les moyens,
détailler les algorithmes et structures de données utilisés ou développés,
décrire les autres solutions possibles mais non retenues,
justifier le choix de la solution.

Documents : le dossier de conception détaillée, le dossier de définition des algorithmes, desstructures de données, des interfaces..., la procédure de fabrication, le dossier justificatif des choix.

Contrôles : la revue de conception détaillée

Réalisation :

But : coder le programme.

Documents : les sources du programme.

Contrôles : la revue de codage qui vérifie de la présence et de la pertinence des commentaires, du respect des règles de codage.

Tests unitaires : "Est-ce que les modules sont corrects?"

But : s'assurer de la correspondance entre la réalisation et la documentation de définition (au niveau du module),
faire des tests aux limites (mémoire, arithmétique, performances…).

Documents : le cahiers de tests unitaires complets.

Tests d'intégration : "Est-ce un bon produit?"

But : s'assurer de la conformité des interfaces vis-à-vis de la documentation de conception générale.

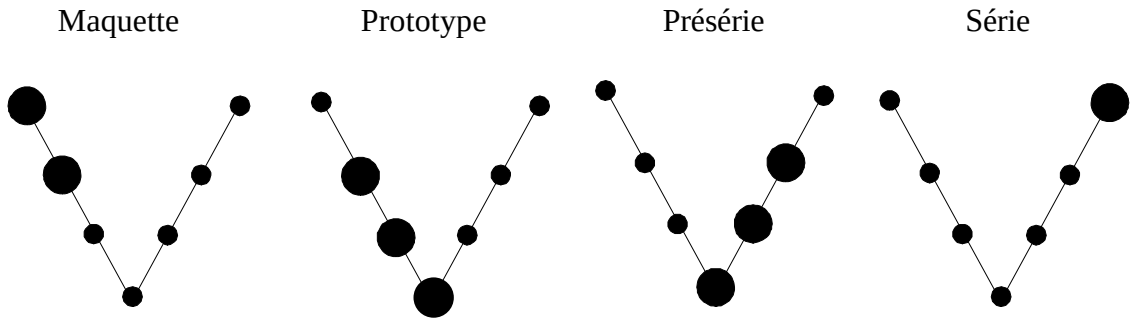
Documents : le cahier de tests d'intégration complets.

Validation : "Est-ce LE bon produit?"

But : consigner les résultats de tests avec le client

Un très bon programmeur crée, teste et documente 20 kls/an.
Une bonne moyenne est 5 kls/an.

Il y a quatre grandes phases dans la production d'un logiciel et chacune d'entre elles comporte ce cycle :



La **maquette** permet de valider les hypothèses et approfondir une phase d'analyse. Elle est très utilisée pour définir l'ergonomie d'une IHM et pour vérifier des hypothèses : Algorithmes, fonctions, bibliothèques, paramétrages. Elle permet de valider les phases de développement.

Le **prototype** est la première version d'un système. Il est généralement incomplet. Il constitue un test "grandeur nature" (Bêta version).

1.4.2. / Maintenance et Evolutions (Lois de Lehman)

Le changement continu : les programmes "P" et "E" sont constamment modifiés. Le processus de maintenance ne s'arrête que si le coût du changement dépasse le coût de refonte complète du logiciel.

La complexité croissante : la modification continue fragilise la structure initiale. On crée de nombreux objets d'où un accroissement de l'entropie et de la redondance. Il y a création de nouveaux objets incompatibles avec les solutions initiales.

L'évolution du programme : la dynamique de croissance d'un système est autorégulée par l'environnement et l'organisation du projet. On amorti toujours les initiatives individuelles.

Le processus évolutif :

	1ere version	2eme version
Spécifications & Conception	60%	15%
Implémentation	15%	25%
Tests unitaires & Tests d'intégration & Validation	25%	60%

La durée de vie du système est à peu près égale à la durée de la première phase de spécificationet de conception. Les techniques de tests sont fondamentales pour le contrôle des coûts de maintenance.

1.4.3. / Ré ingénierie du logiciel

Le coût du logiciel augmente avec son âge :

* Architecture et interfaces saturées,
* Documentation technique rarement mise à jour en parallèle des évolutions,
* Pertinence des tests : ils n'ont pas forcément suivi l'évolution du logiciel,
* Changement de l'effectif de l'équipe de développement (dilution du savoir-faire).
* On rajoute du code avecl' intégration d'erreurs (oscillations).

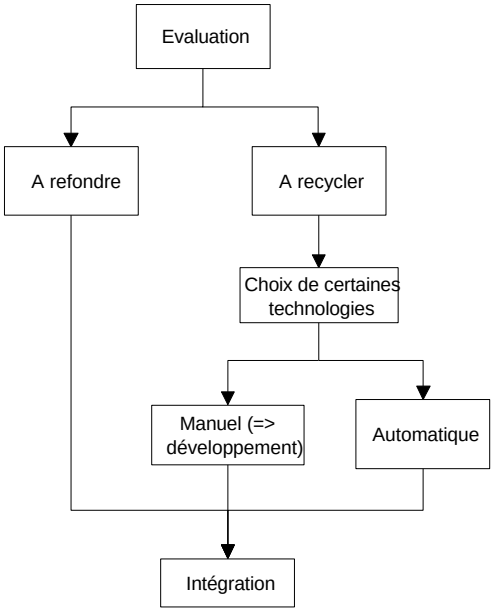
Deux choix possibles : - tout refaire (on retraite le logiciel).
- récupérer en modernisant (on recycle).

Les vieux logiciels contiennent un capital : - conception et architecture éprouvées.
- code source adaptable aux nouvelles technologies.
- tests et documents d'analyse récupérables.

↓

Ces conditions sont vrai si on dispose d une bonne logistique du projet : QUALITE

Les outils : compilateurs, décompilateurs,
traducteurs : éditeurs à syntaxe paramétrable, auto-indentation, mots-clés en valeur, …
traducteurs de programme (Pascal ➔ C, …),
traducteurs de fichiers en base de données,
peuplement automatique de dictionnaire,
réorganisation de base de données….

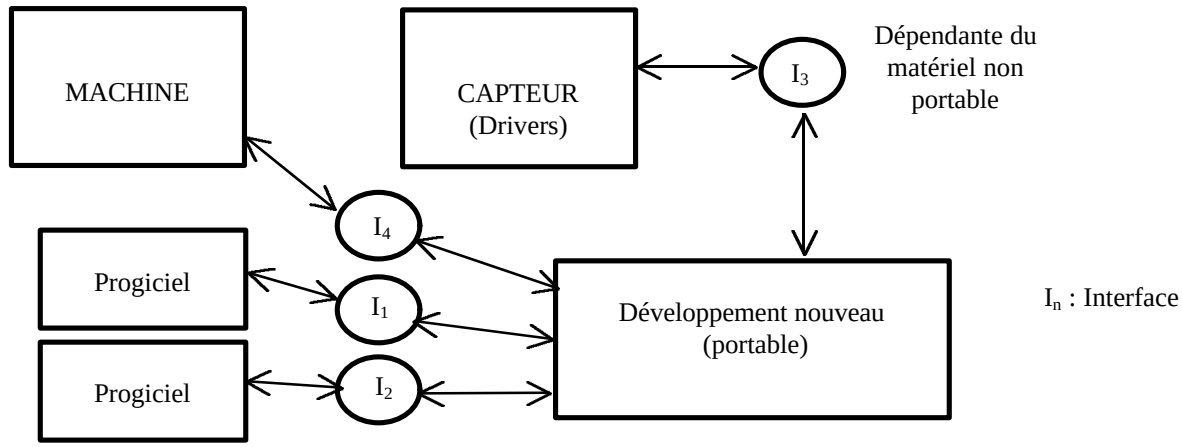


Processus de ré ingénierie d'un logiciel

1.4.4. / Intégration de systèmes à logiciels prépondérants

Systèmes informatiques complets :

Le logiciel à développer est l'élément qui présente le risque principal face aux dysfonctionnements, par rapport aux logiciels achetés et aux matériel et logiciels de base du constructeur qui eux propose des interfaces de communication entre les couches logicielles et matérielles (drivers).



L'intégration de simulateurs permet le développement pendant l'évolution du matériel. Ils Peuvent conduire à un réapprovisionnement lors de la détection d'incompatibilité.

L'homologation des appareils et des logiciels achetés permet d'éviter les interfaçages directs.

1.4.5. / Outils de gestion du cycle de vie

Outils horizontaux : éditeurs, compilateurs (outil fondamental), débuggeurs, bibliothèques de tests.

Outils verticaux (transversaux) : documentation de projet, gestion de projet (plan de développement, d'assurance qualité, fiches de suivi, …), gestionnaire de versions, dictionnaire de données…

Structures d'accueil : organisation de l'échange d'informations entre ces outils :

interface graphique uniformisée,
modèle de données suffisamment riche,
échange de messages.

} **Faciliter le travail en équipe**.

1.5. / Cinétique, dynamique, régulation du cycle de vie

1.5.1. / Le contrôle de processus

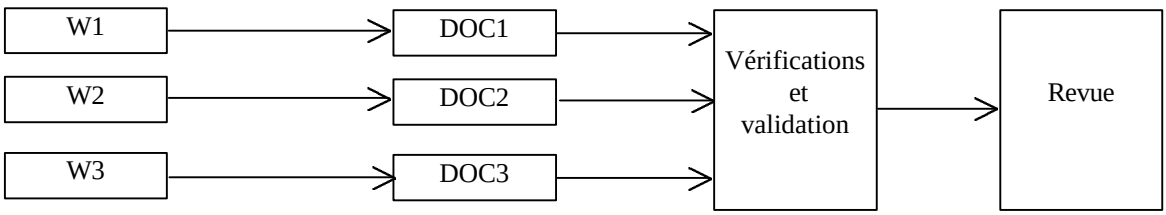
Moyen : Les revues techniques formelles mettent en évidence l'existence inévitable d'erreurs. Elle doivent présenter à des tiers de même niveau hiérarchique les idées et les documents concernant les tâches critiques.

Pourquoi : On a des erreurs humaines.

Comment : Processus de la revue

L'équipe Revue organise la revue.
L'équipe Projet prépare la revue.
L'équipe Revue énonce les recommandations et rédige le rapport de revue.
L'équipe Projet prend en compte les recommandations, conclut et gère le suivi.

On ne juge pas les personnes mais l'adéquation des choix.
Tout ce qui n'est pas écrit n'existe pas.



L'Equipe projet :

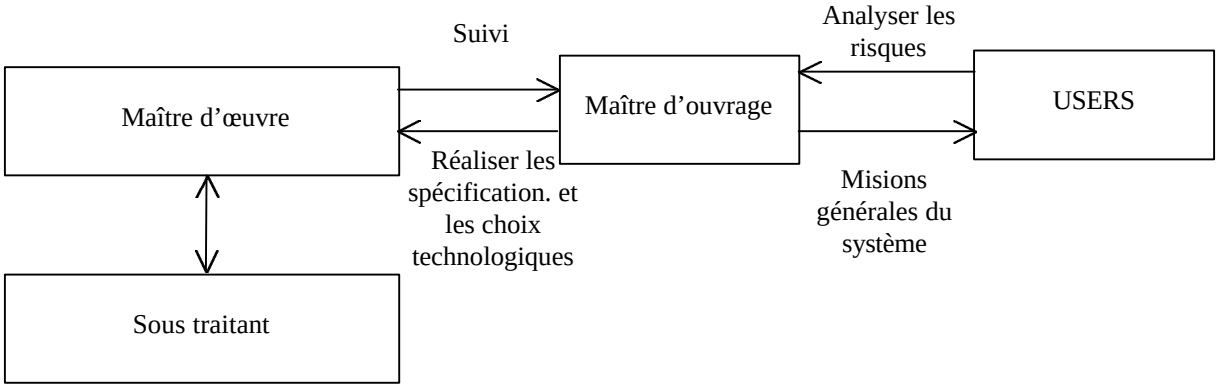
- Doit préparer la revue.
- Prendre en compte des conclusions.
- Doit assurer le suivi.

L'Equipe de revue :

- Doit consulter l'équipe projet.
- Doit énoncer ces recommandations
- Doit organiser la revue.
- Doit rédiger le rapport de la revue.

1.5.2. / Classification par difficulté

Les méthodes organisationnelles :



Nature du problème	Structure de données	Algorithmes	Contrôle du programme
Gestion	Difficile	Simple	Simple
Scientifique	Simple	Complexe/Très complexe	Simple
Temps réel	Simple	Difficile	Difficile/Très difficile
Base de données	Difficile/Très difficile	Difficile	Facile
Compilateurs	Difficile/Très difficile	Très difficile	Facile
Système d'exploitation	Difficile/Très difficile	Difficile	Difficile/Très difficile

Type de difficulté : - Volume et délai de réalisation. A partir d'environ 200 kls, tout devient très difficile.
- Délais trop courts ou trop longs. On a une concurrence, un vieillissement, et de la négligence qui entraîne un empilement des tâches).
- Maturité, résistance au changement.

Conditions du succès : - Ne pas brûler les étapes.
- Adapter la méthode au problème, à l'environnement et pas l'inverse.
Nécessite l'adhésion de tous.

1.5.3. / Place centrale de la programmation dans le cycle de vie

Un **programme** est un assemblage d'algorithmes et de fonctions dont la représentation est le codage. Il doit gérer les ressources (mémoire, espace disque …) et les différents composants matériels. Il est constitué de différentes structures de données qui permettent la gestion des E/S, des ressources, et de paramétrages.

Le modèle de programmation est la manière dont une machine réelle est vue au travers des langages.

- Cobol : univers plat (accessibilité générale), statique, séquentiel.
- PLE, Algol, Pascal, C : univers hiérarchique, dynamique, asynchronisme (exception sauf en C).
- Ada, C++ : héritage, exceptions, modèle objet.
- client-serveur.

Métrologie de programmation est la mesure de l'activité de programmation dont on a 3 niveaux de productivité :

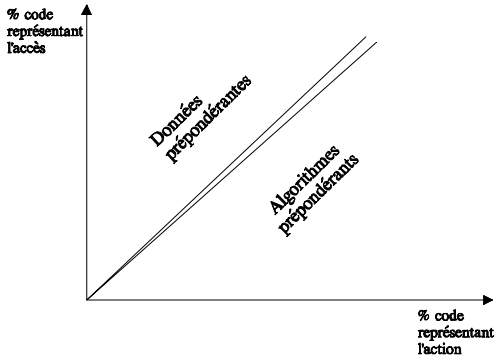
- L'assembleur.
- Les langages évolués : C, Pascal, …
- Les languages de 4eme Génération (L4G) : Smaltalk, C++, Ada, …

La productivité est liée au modèle et pas au langage.

On caractérise le W de programmation en nombre de ligne de source à réaliser : unité '**kls**' ou kilo ligne de source.

Aspect dynamique :

La taille du code réservée à la lecture de la structure de données définie la difficulté de représentation de la structure de données.



Profil d'exécution en terme de volume

1.5.4. / Normalisation, standards

Aspects positifs : image qualitative,
définition d'un langage commun (permet une communication facile),
garantie de la pérennité si la norme est internationale

Aspect négatif : élément de stratégie industrielle profitant uniquement aux groupes qui la contrôle.

1.6. / Méthodes d évaluation de coûts de projet informatique

On utilise différents modèles, principalement basé sur des études statistiques qui ont été estimées à partir d'une expérience passée.

1.6.1. / Introduction

Les modèles statiques (micro-modèle) :

Le **prédicateur** est la variable de départ pour le calcul des autres variables. c'est modèle remontant qui va des plus petit composants vers les plus gros.

On distingue deux types : - les modèles univariables avec un seul prédicateur.
- les modèles multivariables avec des équations qui sont constituées de plusieurs prédicateurs.

Exemple : SEL (Software Engeenerie Laboratory - université du Maryland)

Effort : $E = 1{,}4.S^{0{,}93}$
Durée : $D = 4{,}6.S^{0{,}26}$
Documentation : $NbPages = 30{,}4.S^{0{,}9}$ avec S en KLS.

Le modèle de **COCOMO** (Constructive Cost Model) permet l'estimation du nombre de lignes de code. Il commence au plus bas de l'architecture, c'est à dire au niveau du module (BOEHM) puis il prend en considération les niveaux supérieurs.

Les modèles dynamiques (macro-modèles) :

On définit des équations entre les variables intéressantes du modèle (coût, taille, et temps), sans en privilégier aucune. Ces modèles sont dit "dynamiques" car ils sont non-orientés. On peut calculer n'importe quelle variable à partir des autres. Ils sont appelés aussi des "macro-modèles" car les détails sont ignorés. On assimile cette notion à une méthode descendante. Seules sont conservées les variables principales.

Exemple : le modèle de **Putman** est basé sur les fonctions de Norden-Rayleigh. Effort = f(Temps).

"Un logiciel est un ensemble de sous-problèmes indépendants."

a) Il y a un nombre fini de sous-problèmes.
b) Chaque effort de développement enlève un sous-problème
c) L'effort restant est proportionnel au nombre de sous-problèmes restants.

On suppose que les sous-problèmes sont de complexité équivalente.

Le point de départ est l estimation de la taille du code.

Modèle de **Parr** : Les problèmes ne sont pas indépendants car la résolution d'un problème peut en impliquer de nouveaux (l'inspection du code peut entraîner la découverte d'erreurs).

1.6.2. / Paramètres d'estimation

A partir de la comparaison de 15 modèles (Mohanty) on défini 3 groupes d'attributs relatifs à :

- l'environnement (**36** %).
- la taille (**20** %).
- la complexité (**19** %).

Cela représente 75% des prédicateurs.

Estimation de la taille :

L indicateur : le 'kls' permet de compter les lignes vides et les commentaires. On a une dépendance vis-à-vis de la présentation du code.

Le nombre de lignes exécutables est plus significatif. On le nome NCSS (Non Commentary Source Statement) ou ILS (Instruction ligne source).

Exemple : Avec le langage C, on compte le nombre de '{' et de ';'.

Travaux d'Allan J. Albretch (IBM). Mise en évidence des relation entre NCSS de 24 programmes de traitement de données en fonction de :

- Nombre de types d'entrées utilisateur.
- Nombre de types de sortie utilisateur.
- Nombre de requête (combinaison entrées/sorties).
- Nombre de fichiers utilisés et partagés.
- Nombre d'interface.

On a 3 niveaux de complexité : simple, moyen, complexe.

La mesure : le 'FP' (Fonction point) qui permet l'estimation de la taille et du coût en PL1 et COBOL.

FP = a.Entrées + b.Sorties + c.Requête + d.Fichiers + e.Interfaces

avec a, b, c, d, e des constantes statistiques. Taille = 118,7.FP - 6490 = NCSS.

Classification des projets selon leur taille :

Projet	Petit	Moyen	Grand
Structure autorité	1	2	> 2
Taille (kls)	< 50	50 à 300	> 300
Durée (ans)	< 2	2 à 3	> 3
Nb personnes/an	4 à 5	5 à 10	> 10
Doc (65 lignes/page)	< 100	< 5000	> 5000

Méthodologie : Le Brainstorming avec une équipe expérimentée.

1ére séance : uniquement productive. Pas de critiques sur les nouvelles idées.
2éme séance : mise en forme des idées par le CP et synthèse.

Différents problèmes : - La rotation du personnel.
- La diffusion de l'information.

Exemple :

- SWIFT : réseau international de transfert interbancaire de fonds (200 banques. 1973). Refondu pour faire face au développement et pour mettre en évidence une architecture décentralisée (15 % de croissance/an. 1 million de messages/jour)

- SWIFT 2 (1982) : 1,5 million de messages/jour
978 logiciels
90 ingénieurs pendant 6 ans
coût : 90 MF annuels.
1 Mls (NCSS).

1.6.3. / COCOMO (Constructive Cost Model)

C'est un modèle statique (multivariable). La taille et l'environnement jouent un rôle prépondérant. Il n'y a pas de complexité.

3 modèles : modèle de base : projet < 50 kls (≈ 1000 pages de 50 lignes). Petit projet.
modèle intermédiaire : projet de taille moyenne
modèle détaillé : grand projet

Pour chaque modèle, on a 3 modes de développement :

- Mode organique (**S**) :petite équipe expérimentée et environnement familier (traitement classique).
- Mode semi-détaché (**P**) : équipe assez expérimentée, environnement connu.
- Mode imbriqué (**E**) : projet à contraintes serrées, dues à l'environnement ou une certaine nouveauté de l'application (temps réel, ...)

Modèle de base :

Prédicateur : **S** (kilo ligne de source '**kls**) **La taille.**
Autres variables : E (Hommes.mois '**H.mois**) **L effort.**
T : (mois) **Le temps.**

Mode organique :	$E = 2{,}4.S^{1{,}05}$	$T = 2{,}5.S^{0{,}38}$
Mode semi-détaché :	$E = 3.S^{1{,}12}$	$T = 2{,}5.S^{0{,}35}$
Mode imbriqué :	$E = 3{,}6.S^{1{,}2}$	$T = 2{,}5.S^{0{,}32}$

Exemple : pour un projet de 30 kls.

Mode organique : E = 85 hommes mois, T = 13,5 mois.
Mode semi-détaché : E = 135 hommes mois, T = 13,9 mois.
Mode imbriqué : E = 213 hommes mois, T = 13,9 mois.

ATTENTION : T semble rester constant alors E augmente. Donc, il suffirait de rajouter du personnel quand la difficulté augmente, ce qui n'est pas toujours vrai.

Modèle intermédiaire :

mode organique :	$E = 3{,}2.S^{1{,}05}$
mode semi-détaché :	$E = 3.S^{1{,}12}$
mode imbriqué :	$E = 2{,}8.S^{1{,}2}$

On utilise des directeurs de coûts. On a 15 modèles en plus par rapport modèle de base. On défini l'effort ajusté E_a par :

$$E_a = \prod_{i=1}^{15} C_i . E$$

Il y a 4 catégories de directeurs de coûts :

1/. Caractéristiques du produit :

- fiabilité requise (**RELY**)
- taille de la base de données (**DATA**)
- complexité globale (**CPLX**).

2/. Caractéristiques de la machine cible :

- temps d'exécution (**TIME**)
- mémoire principale (**STOR**)
- stabilité de l'environnement (**VIRT**)
- interactivité des moyens de développement (**TURN**).

3/. Caractéristiques du personnel :

- compétence et cohérence de l'équipe d'analystes (**ACAP**)
- expérience du domaine d'application (**AEXP**)
- expérience des programmeurs (**PCAP**)
- connaissance du système d'exploitation (**VEXP**)
- connaissance du langage de programmation (**LEXP**).

4/. Caractéristiques du projet :

- utilisation de techniques modernes de développement (**MODP**)
- niveau de sophistication des outils de développements (**TOOL**)
- délai de mise à disposition (**SCED**).

Niveau / Coût	Très bas	Bas	Nominal	Haut	Très haut	Très très haut
RELY (Fiabilité)	0,75	0,88	1	1,15	1,40	-
ACAP (compétence)	1,46	1,19	1	0,86	0,71	-
SCED (délais)	1,23	1,08	1	1,04	1,1	-

Modèle détaillé :

a) Il y a un découpage des modèles en phase. L'effort est calculé indépendamment pour chaque phase.
b) L'application est découpée en 3 niveaux de hiérarchie : module, sous-système, et système.

On évalue les directeurs de coûts au niveau où ils varient le plus. Le calcul de la taille **(Sy)** permet la prise en compte de la réutilisation. Par module, le coefficient de multiplication de la taille 'A' avec ces critères, s'écrit comme suit :

D = % modification de design,
C = % modification du codage,
I = % modification de l'intégration.

$$A = \frac{0,4.D + 0,3.C + 0,3.I}{100}$$

Pour passer du modèle intermédiaire au modèle détaillé, le facteur d'ajustement est :

$$Sy = A.S$$

où S est la taille d'origine calculée suivant le modèle intermédiaire.

Phases de développement : **Correspondance avec le cycle en V :**

1/ Planification, spécification (20 %)

6 à 8% de l'effort de développement (Es = 7%) **Phase 1**
10 à 40 % du temps de développement (Ts = 20%)

2/ Conception générale (25%)

16 à 18% de l'effort (Ecg = 17%) **Phase 2**
19 à 38% du temps (Tcg = 26%)

3/ Programmation (50%)

3.1/ Conception détaillée Ecd = 25%	**Phase 3**
3.2/ Ecriture du code + tests unitaires Eet = 33 %	**Phase 4 & 5**
48 à 68 % Effort (Ep = 58%)	
24 à 34% Temps (Tp = 48%)	

4/ Intégration et tests (25%)

16 à 34% Effort (Ei = 25%)	**Phase 6**
18 à 34% Temps (Ti = 26%)	

Attention : La phase 1 est préliminaire, elle vient s'ajouter au temps nominal.

1.7. / Discipline de test

1.7.1. / Constatations

"Les très gros bogues peuvent vivre très longtemps."

"Les environnements de programmation ont des spécificités mal connues du programmeur." Les tests doivent se faire sur la machine cible.

"Les bogues mineurs peuvent avoir des conséquences lourdes." (La sonde Mariner loupe Vénus parce que les "," ont été remplacées par des "." pour les nombres numériques).

"N'importe quel niveau de la couche logicielle de l'environnement de développement, s'il est erroné, provoque la vermination du code final." (Superviseur, compilateur, émulateur, chargeur/déchargeur de codes, spécification de la machine virtuelle, outils de test, librairies, compilateurs, makefile, éditeur des liens, déboguer, noyaux, matériel, CPU.

1.7.2. / Introduction

1950 : Fortran. "On écrit et ensuite on teste." Généralement on confond tester et déboguer.

Solution : il faut les séparer (1957). Conférence sur le sujet en 1972.
Déboguer c'est découvrir un mauvais fonctionnement et corriger en conséquence.

1973 : Tester (1ere définition) : "Procédure qui fait la preuve qu'un programme fait ce qu'il est supposé faire". Cette vue incorrecte du test en entraîne la médiocrité. D'après Meyer, "on teste dans l'optique qu'il y a des erreurs et le jeu de test est là pour les trouver."

Problème : on a une vue incorrecte du test.

1979 : Tester (2^{éme} définition) : "Action de faire fonctionner un programme avec l'intention de trouver des erreurs."

2 problèmes : on teste après l'écriture du code,
le test est lié à la notion d'examen et de recherche de faute de la part du programmeur.

Solution : il faut concevoir le test pour aider à la compréhension du problème initial et améliorer la qualité (remplir les spécifications, facile à utiliser, code documenté, structuré, performances, facilité de modification, réutilisation, ...)

1990 : Tester (3^{éme} définition) : mise en évidence de la qualité du logiciel.

1.7.3. / Facteurs de qualité

Qualité fonctionnelle (extérieure) :

- Produit les résultats attendus dans les spécifications,
- Sécurité de fonctionnement (pas de plantage, pas de perte de données, sauvegarde, point de reprise à chaud (redémarrage), à froid (coupure de courant))
- Facilité d'utilisation,
- Intégrité (ne perturbe pas l'environnement).

Qualité de réalisation (intérieure)

- Performances,
- Tests des composants (tests unitaires),
- Tests des assemblages (tests d'intégration),
- Documentation interne,
- Structure du code.

Qualité future (capacité d'adaptation)

- Flexibilité (facilité de modification),
- Réutilisation,
- Facilité de maintenance,
- Evolution facile.

1.7.4. / Méthode de test du logiciel

Les questions qu'il faut se poser avent le test d'un logiciel sont les suivantes :

1/ Que faut-il tester ?
2/ Quand faut-il tester ? Quand commencer ? Quand arrêter ?
3/ Qui fait les testes ?

On doit trouver le compromis entre coût et temps minimaux, et qualité maximale.

1.7.5. / Pratiques courantes

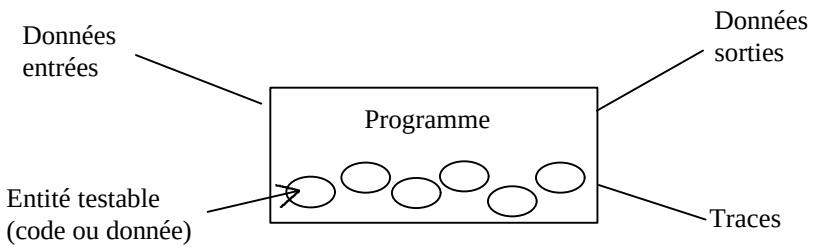
"Chacun fait ce qu'il pense." Il y a moins d'une entreprise sur dix qui a une méthodologie systématique. Les procédés individuels sont Ad hoc.

1.7.5.1. / Le test unitaire (ou des unités)

Que doit on tester ? ➔ Les programmes individuels pendant leur écriture.
Quand doit on tester ? Qui doit tester ? ➔ C'est le programmeur qui décide.

Il n'y a pas de rapport d'activité de test. Le test unitaire est informel.

Parfois on s'oblige à rédiger un plan de test (ou Test Résultat). La validation du test est réalisé par le chef de projet.



Le test "**boîte noire**" : dans ce test, on ne prend en compte que les données entrées et les données sorties (éventuellement les traces). On analyse la structure et la correspondance des données entrées et la combinatoire des divers scénarios possibles. Ce genre de test présente les limites suivantes :

. Problématique de la détermination des domaines d'entrée et de sortie.
. Rendement de l'effort de test.
. Si modification des chemins de sauvegarde des sources alors l'outils doit pouvoir retrouver les sources d'origines et modifier en conséquence les chemins qui sont implémentés dans le code.

Le test "**boîte blanche**" : dans ce test, on prend en compte les données entrées, les données de sorties, les traces, ainsi que la structure du programme et son environnement d'exécution. Il y a différents niveaux de granularité : les instructions élémentaires, les régions connexes, les fonctions et procédures. Ce genre de test présente les limites suivantes :

. Quantité et pertinence de l'information produite.
. Stabilité des tests (dépend du code).
. Nécessité d'utiliser des outils d'analyse de chemin (une modification du code entraine une modification du chemin).

1.7.5.2. / Le test d intégration (ou du système)

"Est-ce un bon produit ?"

On assemble les modules et on applique l'approche du test de la 'boîte noire'. Les essais sont les suivants :

1/ Test de communication entre les interfaces des modules.

2/ Test de fonctionnement : vérification du respect des spécifications et vérifications des assemblages.

La personne qui teste est le chef de projet ou un groupe désigné pour l'occasion. A l'issue, on rédige un rapport qui signale les anomalies détectées avec le contexte, les outils utilisés et l'environnement. Il énumère tout ce qui est nécessaire à la reproduction des anomalies.

1.7.5.3. / Le test de réception (ou recette)

"Est-ce **LE** produit attendu par le client ?". Le bon produit est-il prêt à être utilisé ?

Il s'agit de démontrer au client que le produit est prêt pour l'utilisation. Cette phase de test est informelle. Un rapport d'essais se fait en général avec le client qui vise une fiche de recette.

1.8. / Principes pour la conduite de tests

1/ "Il n'est pas possible de faire un test complet."

Exemple :

```
Trouvé := faux;

Tant que (EntréesDansLaTable) et (non Trouvé) faire
.
.
  si (EntréeCourante = "Jean")
     alors Imprimer(EntréeCourante);
        Trouvé := vrai;
  Fin si
Fin Tant Que

si (Trouvé = faux)
     alors Imprimer("Le nom" EntréeCourante "n'existe pas.");
.
.
```

Teste-t-on le cas où le nom est présent ?
Teste-t-on le cas où le nom est absent ?
Tables de taille différente ?
Test avec différentes position des noms dans la table (en particulier les bornes) ?
Type de nom (longueur différente, longueur limite, case inoccupée, ...) ?

Pour éviter cela, il faut être sûr des spécifications. On est jamais sur que les spécifications sont correctes (ce que veut le client) Il n'existe pas de système de test qui identifie les programmes corrects. De plus, on n'est jamais sûr du système de test lui-même. On arrête la procédure quand on juge que la continuation est improductive.

2/ "Tester est un travail difficile et créatif." Il demande de l'imagination.

Fausses idées :
- Tester, c'est facile,
- N'importe qui peut tester,
- Pas d'expérience requise.

Tester n'est pas simple car :
- Il faut comprendre en profondeur le système,
- Ces systèmes ne sont jamais simples.

3/ "Il faut tester pour prévenir l'apparition des défauts."

Depuis 15 ans, on considère les tests comme une activité qui dure tout le long du cycle de vie du logiciel, pour détecter les décalages. Ces décalages viennent d'une mauvaise compréhension du problème, d'une mauvaise structure de données, d'une mauvaise communication des spécifications, d'une mauvaise compréhension ou choix des algorithmes, ...

Les tests doivent aider à découvrir le plus tôt possible les décalages entre ceux que veut l'utilisateur et ce qui est réalisé. Il faut penser à l'avance ce qu'il faut tester car plus on avance, plus le coût de la correction de l'erreur est élevé.

Dans certain cas, on peut tester un logiciel par un utilisateur inexpérimenté qui dégrossit les plus grosses erreurs (Le test du singe !!!).

4/ "Les tests sont basés sur les risques connus."

5/ "Les tests nécessitent l'indépendance du testeur."

La personne qui est la moins indépendante est celle qui connaît le mieux le système. Or, le principe n°2 dit qu'il faut connaître le système à tester. Il faut donc "jouer le jeu" à fond.

6/ "Les tests doivent être planifiés."

On répond aux questions "Que teste ? Quand tester ? Quand s'arrêter ?" en trouvant le meilleur compromis coût/qualité.

Le plan de test est le suivant :

1/ Cadrage de test : objectifs, description de l'approche, tâches nécessaires à l'accomplissement du test (dépendance entre les tâches).

2/ Projet du test : spécification des tests à développer, description du jeu de test.

3/ Documents de test : cahier d'enregistrement des résultats. Il y a qu'un seul document qui décrit l'activité du test. Plusieurs parties de ce document sont associées aux tâches du logiciel à tester.

1.8.1. / Outil de test : la revue de contrôle

1.8.1.1. / Rôle

Définition : c'est l'évaluation technique réalisée par un groupe travaillant sur un projet. Historiquement, la revue servait à compter le nombre de tâches finies par rapport au planning initial. Un projet peut rester achevé à 90% longtemps, puis partir à la poubelle. Ce qui importe, c'est la qualité de réalisation des tâches et non le nombre.

Questions : Le planning a-t-il été respecté ? Quelle est la qualité de ce qui a été produit ?

La revue facilite le degré d'avancement du projet.

Revue formelle : pour laquelle on a un produit et un document.

Revue informelle : on a pas de document. C'est le seul outil de test disponible dans les premières phases du développement logiciel.

1.8.1.2. / Planification des revues

Les revues formelles sont choisies au début du projet. Elles coïncident avec la fin de chaque phase.

① **Revue de spécification** : approbation du document de spécification (compréhension de ce qu'il faut réaliser).

② **Revue d organisation du logiciel** : approbation de la spécification et des principes de conception.

③ **Revue du plan principal de test** : approbation de la statique et des méthodes de test.

④ **Revue de conception générale** : compréhension globale du système, compréhension des tests globaux.

⑤ **Revue de conception détaillée** : validation de l'implantation (structure du code). On démarre le codage. Critiques des choix des algorithmes et des structures des données.

⑥ **Revue de codage** : approbation des modules (structure du code, commentaire, nom des procédures).

⑦ **Revue d'intégration** : validation des interfaces (test de réception : approbation opérationnelle).

⑧ **Revue de Recette** : validation avec le client.

1.8.1.3./ Organisation des revues

- On Planifie d'une date où toutes les personnes concerné seront présentes.
- On Désigne un responsable (planification, organisation, rapport de la revue).
- On Créé un plan : qui participe, quel sont les documents nécessaires, quelles sont les pré conditions requises, vérification de la liste des considérations, énumération des conditions ou post conditions de terminaison de la revue, contenu du rapport.

1.8.1.4. / Efficacité des revues

Les revues doivent pouvoir :

- Evaluer l'avancement par rapport à la planification.
- Mettre en valeur les points forts et les points faibles.
- Découvrir tôt les problèmes.
- Eduquer les participants et améliorer les compétences de la direction.

1.8.2. / Tests des spécifications

1.8.2.1. / Objectif

But : déterminer les problèmes à résoudre et les résultats à obtenir.

Questions : "Les fonctionnalités attendues ont-elles décrites ?" (complétude → checklist).
"La qualité est-elle décrite ? (performances, ...)
"Cohérence, simplification, ... ?"

Il faut donner des spécifications claires, simples et complètes.

1.8.2.2. / Méthodes de test

1/ A partir d'exemples de comportement dans des configurations précises. Cela permet de mettre en évidence des contextes non imaginés lors de la conception (configurations flous et incomplètes, cas d'erreurs).
2/ Par la conception de prototypes, modèles, schémas (expérimentations : permet de vérifier que l'on résout bien les problèmes).
3/ Utilisation de langages de spécification (Z, B).

Nota : L'élément essentiel des méthodes de test est la revue.

1.8.3. / Tests de conception

1.8.3.1. / Objectif

Définition : **Concevoir** c'est trouver une solution satisfaisant les données du problème."

Test de conception : trouver **LA** meilleure solution.

Questions : - Est-ce que la solution est bien choisie ?
- Existe-t-il plus simple ?
- Réalise t'on les fonctionnalités demandées ? (risques d'échec)

1.8.3.2. / Méthode

Analyse d'alternatives : c'est une analyse critique de l'ensemble des solutions proposées (existe t'il d'autres solutions de faire avec quels avantages et quels inconvénients).

Exemple : le "forcing" : faire une revue 15 jours après l'expression des besoins pour que chacun propose une solution possible. Mettre les équipes en compétition.

Pour un projet de type p, des résultats statistiques ont montrés qu'à partir de 4 études de conception (1 étude de base + 3 études supplémentaires) avec 6 semaines supplémentaires dans le cycle de conception (2 semaines par étude + quelques jours de revue), les résultats des études peuvent varier de 1 à 5 sur les points suivants :

- Le coût escompté.
- Le nombre de ligne de code.
- La sécurité.
- Le temps de développement (6 mois à 3 ans).

1.8.4. / Tests individuels de programme

1.8.4.1. / Introduction

Question : "Le code est-il correct ?"

1/ Fait-il tout ce qui est prévu par les spécifications ?
2/ Fait-il plus que ce qui est prévu ?
3/ Peut-il échouer et que fait-il alors ? (erreur possible et comportement).

1.8.4.2. / Importance de la motivation et planification

La motivation du programmeur pour le test est un gage de garantie de la qualité de son programme. Or le testeur doit être indépendant du code à tester !!! Il faudrait une personne complètement extérieure. Cela coûte trop cher !!! Il faut aider le programmeur à être indépendant en l'obligeant à documenter ce que vont être ses tests.

On est assuré que : - Un certain temps est enlevé au codage pour planifier les test.
- L'effort principal est réalisé avant le codage d'où une motivation au moment des tests.
- La conception des tests devient un document donc peut être consultée par autrui plus tard.

Important : On doit planifier le temps passé à la sélection des tests et à leur degré.

1.8.4.3. / Tests unitaires sur les spécifications

Intérêt : vérifier qu'il ne manque rien au niveau des fonctionnalités de base demandées (vérifier la complétude).

Inconvénient : On ne vérifie pas les fonctionnalités étendues.

Etapes :

1/. Sélectionner les cas décrits dans les spécifications ayant un lien quelconque avec la partie du code à tester.
2/. Créer un ensemble minimum de tests mettant en œuvre tous les types d'entrées/sorties.
3/. Décomposer les tests complexes en cas simples.
4/. Sélectionner les combinaisons possibles de cas simples.
5/. Supprimer les tests inclus dans d'autres (redondance des tests).

1.8.4.4. / Tests unitaires sur la conception

Ces test sont construits sur les algorithmes et les structures de données du programme. La méthode consiste à parcourir la totalité du code (appelée C0). Ils peuvent être automatisés. Ils assure la non-régression, i.e. "une fonction qui marche à l'instant t marche à t+x".

Exemple : le programme 'tcov' sous UNIX.

Soit les lignes de commandes suivantes :

```
cc –a –o prog prog.c
prog < test
tcov prog.c
```

On obtient le fichier 'prog.tcov' qui contient sous forme de fichier texte, un détail en % du code exécuté.

1.8.4.5. / Tests unitaires de robustesse (ou tests des extrêmes)

Pour bien vérifier les cas d'erreurs on réalise des tests aux limites de fonctionnement.

Exemple : le parcours d'un tableau. Que se passe-t-il si :

On a 0 éléments ?
Le nombre maximum d'élément est atteint ? (le tableau est plein).
On est au 1er élément ?
On est au dernier élément ?

Les outils : les générateur de données aléatoires, les trace, l'extraction de données de fichiers d'entrées des précédentes versions.

1.8.4.6. / Tests par construction d'une base de chemins

Ces tests permettent de représenter le programme sous forme de graphe de contrôle. On construit un jeu minimal de tests de tous les chemins possibles (appelé couverture C1). On parcourt toutes les branches indépendantes du programme (par opposition à C0).

Définition : une base de chemin est une combinaison linéaire d'élément de la base.

Au préalable :

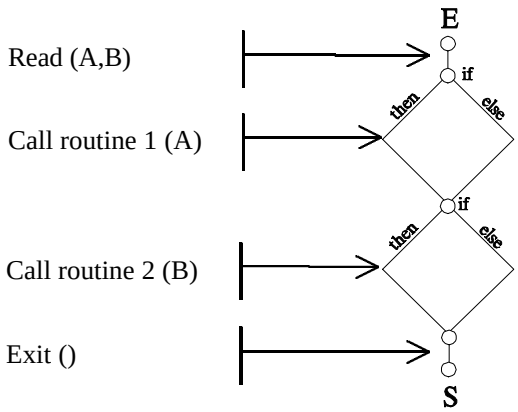
1/. Représenter le programme sous la forme "une entrée, une sortie".
2/. Construire le graphe de contrôle associé à cette forme. Les formes sont les suivantes :

Point de décision → nœud du graphe
Suite d'instruction → arc du graphe

Exemple : Soit le code source suivant :

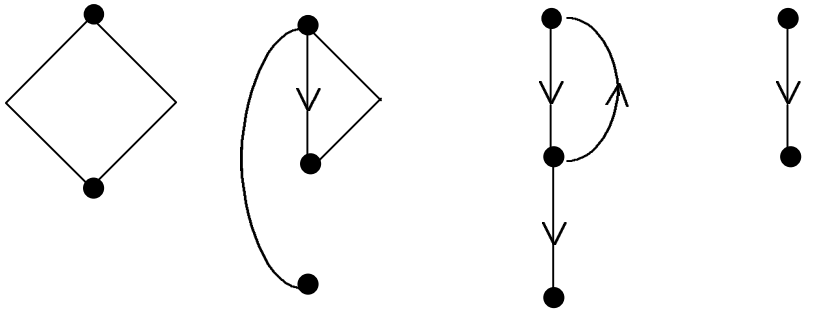
```
main ()
    {
    read (A, B);
    if (A > 0) then
            call routine1(A);

    if (B < 0) then
            call routine2(B);
    exit ();
    }
```



Le nombre cyclomatique d'un graphe est $C = NbArcs - NbNoeuds + 2$. Il permet de mesurer la complexité du programme. C'est le nombre de chemins indépendants, il donne le nombre de tests minimum pour effectuer une couverture C1. Dans l'exemple précédent, on a C=3.

Représentations :



IF THEN ELSE **DO WHILE** **REPEAT UTIL** **GOTO**

Type de couverture : C0 : nœuds (on doit passer par tous les nœuds).
C1 : arcs (on doit passer par tous les arcs).
C∞ : Chemins (on doit passer par tous les chemins).

Méthode de couverture :

- Choisir un cas définissant un chemin et le marquer.
- Marquer les arcs couverts par le cas.
- Réaliser le cas suivant en : partant du même chemin
- Modifier/parcourir au moins un autre arc et le(s) marquer → répéter jusqu'à la valeur de C calculé.
- Définir les résultats attendus.

Soit pour l'exemple précédent :

1/ A > 0, B > 0
2/ A > 0, B < 0
3/ A < 0, B < 0

On obtient une couverture C1. On ne peut détecter les nœuds "isolés".

1.8.4.7./ Test pour la modification d un logiciel (Maintenance)

1.8.4.7.1./ Introduction

Lors d'un changement important, on fait une revue.

Questions : - La complétude est elle respectée (tous les changements) ?
- Les changements ont ils été appliqué à toutes les parties du système ?
- Y a t'il des effets négatif (autres endroit affectés…) ?
- La compatibilité est elle ascendante ?

La documentation de changement et de test des modifications doit être écrite avant. On réalise l'évolution après l'écriture.

Méthodes : - La description du changement.
- La liste des modules affectés : modules à modifier, modules pouvant être concernés mais sans changement.
- Le plan de test du changement (comment le tester).
- La revue d'analyse de la description du changement et du plan de test.

1.8.4.7.2./ Réalisation et tests des modifications

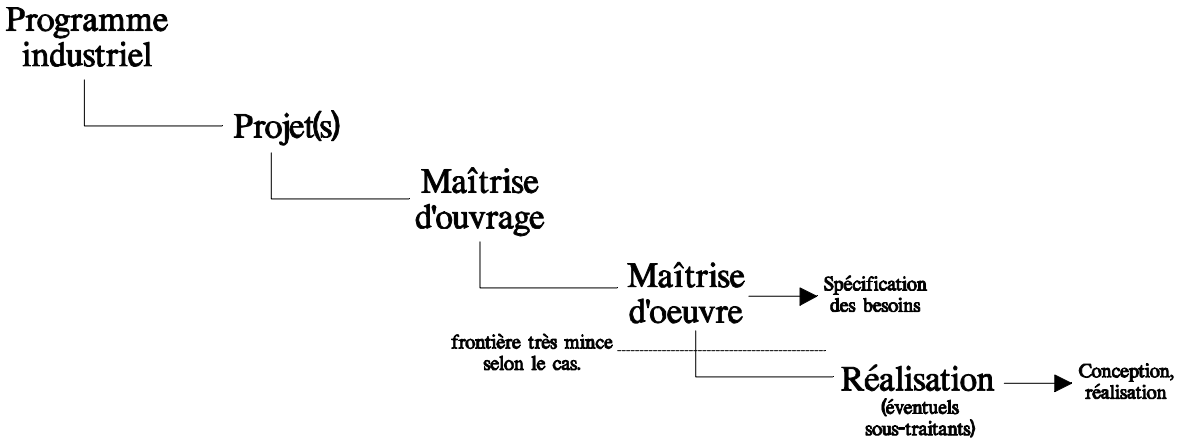
1/. Copier tous les programmes à modifier dans un espace privé.
2/. Réaliser les changements dans les modules concerner dans l'espace privé.
3/. Réaliser les test unitaires dans l'espace privé.
4/. Réaliser les tests de librairies d'intégration avec l'accord du chef de projet.
5/. Réaliser les tests d'intégrations.
6/. Réaliser les tests en vrai grandeur (tests du système complet ou test décrit dans la spécification de changement).
7/. Installer les modifications dans les librairies de production avec sauvegarde de l'ancienne.

1.9. / Gestion de projet et qualité

Définition : Le Projet est une réalisation, une étude, une réponse à un besoin et des moyens. C'est un ensemble d'activités techniques dont le but est de fournir un service ou un produit à coûts et délais fixés.

Exemple : Le programme spatial Français.

La topologie des différentes phases d'un projet est la suivante :



1.9.1. / Le chef de projet

Dans une équipe, on estime le nombre de chef de projet à 1 pour 4 à 5 personnes.

1.9.1.1. / Ces devoirs

Le chef de projet à des obligation de résultats d'un point de vue techniques, sur le plan financier, et dans le respect des délais. Il doit mettre en place les moyens nécessaires pour y parvenir :

1/. La détermination des moyens.
2/. La mise en place et suivi.
3/. Le lancement d'actions spécifiques à un problème.

1.9.1.2. / Ce qu il doit être capable de faire

- Entretenir le relationnel et la communication.
- Pendre des décisions.
- Déléguer le travail à son équipe.
- Etre organisé et méthodique.
- Coordonner les différentes tâches techniques.
- Faire preuve d'innovation (avec le contrôle des risques).
- Motiver son équipe dans ce sens.
- Etre pragmatique (voir l'essentiel).
- Etre réaliste.

} optimise les coûts

- Souci d'informations sur :
  - les compétences des équipes (internes ou externes).
  - l'état de l'art (veille technologique).
  - l'état du marché (concurrents, prix, ...).

- Défendre son projet vis-à-vis de :
  - sa hiérarchie.
  - son client.
  - son équipe (sous-traitants compris).

- Défendre son équipe (y compris les sous-traitants) vis-à-vis de la hiérarchie.

1.9.1.3. / Ce qu il ne doit pas faire

- S'impliquer durablement dans une tâche technique spécifique qu'il s'est attribuée ou attribuée initialement à un membre de son équipe.
- Dépasser ou ne pas remplir le cadre du contrat.
- Prendre du retard sur les prises de décision.

1.9.1.4. / Ces rôles

- Identifier les tâches du plan logiciel.
- Mise en place des moyens techniques et humains.
- Encadrement technique.
- Gestion financière.
- Gestion de configuration et gestion des versions.
- Liaison avec le client, les sous-traitants, la hiérarchie.

Objectif de sa prestation : qualité, dans les délais, coûts.

1.9.2. / Les outils de base en gestion de projet

- La documentation : "toute activité doit conduire à un document".
- Les réunions.
- La découpe en tâches.

1.9.2.1./ La documentation

1 / Elle est une trace (mémoire de l'entreprise).
2 / Elle est un support de base pour la communication des idées.
3 / Elle oblige à clarifier ses idées.

ATTENTION : Eviter la paperasse : un document doit traiter d'un sujet précis. Il doit être concis, géré (nomenclature, ...), mis à jour et correctement diffusé. Un document qui est bien structurée ne doit pas être volumineux.

1.9.2.2. / Les réunions

Attention à la réunionnite !!!

Une réunion doit avoir :
- Un but : « Etre claire sur ce qui est flou ».
- Sélectionner les participants.
- Un directeur de la réunion.
- Un durée limitée dans le temps, "commence une minute avant l'heure et termine cinq minutes avant l'heure"
- Une préparation conçue à l'avance : ordre du jour diffusé correctement, documents lus, questions préparées, réunion préparée par chacun,
- Une fin qui entraîne la prises de décision.

1.9.2.3. / La découpe en tâches

- Documentation : Le Plan de Développement Logiciel (PDL)

- Définition : La **tâche** est :

  - Un travail précis.
  - Un examen précis de ce qui est à faire.
  - Une modifications.
  - Une approbations.
  - Un retour au début.

ATTENTION : Trop de communication entre les tâches → **gestion ite** !!! "A consommer avec modération!"

1.9.3. / L organisation du projet

Les questions du chef de projet sont les suivantes :

- Caractéristiques du projet ? (contraintes, objectifs visés, ...)
- Produits à développer ?
- Tâches nécessaires (organigramme) ?
- Comment relier les tâches entre elles ?
- Planification des tâches ? ←
- Moyens matériels et humains nécessaires ?
- Produits à sous-traiter ou a réaliser ?
- Mise en place des moyens de suivi ?
- Points durs et conséquences ?

PERT : montre l'enchaînement des tâches. "Celle-ci commence après la fin de celle-là."
GANTT : montre le positionnement dans le temps.

Le Plan de Développement Logiciel est réalisé par le chargé de produit et il est analysé par :

- La hiérarchie.
- Les responsables des tâches → débouche sur la revue d'approbation.
- Le service qualité.

Pour la documentation du projet, il faut établir un classement type.

Avantages : le chef de projet a une check-list de tous les documents concernant le projet.
les membres du projet ont facilement accès à l'ensemble des documents.
le changement de chef de projet est aisé.

Les six types de documents et leur contenu sont nommé comme suit :

1/ L'organisation :

L'historique.
Le PDL.
Le Planning.
Les fiches de tâches.
La liste des documents techniques à créer.

2/ La Gestion financière :

L'historique.
La comptabilité analytique.
La facturation.

3/ Le client : documents issus des réunions avec le client.

4/ Les sous-traitants et fournisseurs

5/ Le suivi technique :

Les compte-rendus des réunions de suivi.
Les notes techniques.

6/ Documents applicables :

La notification du marché.
Le cahier des clauses techniques particulières (ou cahier des charges).
Les normes en vigueur à respecter.

Avantages : facilité d'accès, check-list, pérennité des compétences techniques.

1.9.4. / Le suivi du projet

- Encadrement de l'équipe technique :

. Information.
. responsabilités.
. Choix des solutions techniques.
. Coordinations des équipes.
. Règles de travail en entreprise.
. Veillez au respect du planning.
. Suivre la spécification technique.
. Dialoguer avec le service qualité.

- Gestion financière :

. Evaluation et ré-orientation.
. Prévoir des réserves.
. Respecter les coûts.
. Facturation.
. Liaisons administratives.

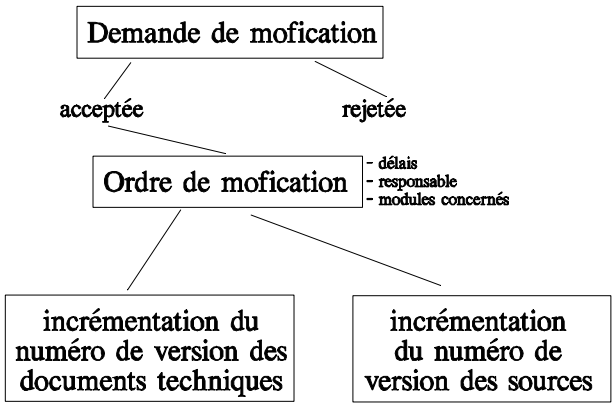
- Liaison avec le client.

- Liaison avec les sous-traitants.

- Liaison avec la hiérarchie.

- Gestion de configuration :

. Totalité des produits conçus et réalisés et documentation technique associée.
. Gestion et organisation de la documentation.
. Identification des produits (versions).
. Gestion stricte des modifications.



- Structure de la documentation technique :

. But : faire vivre les documents et les utiliser simplement.
. Règles : un document doit avoir une référence et une seule le tout structuré avec une arborescence.

1.9.5. / Assurance qualité et gestion de projet

La Qualité : c'est satisfaire les besoins, facilité l'utilisation, sécuriser et fiabiliser les fonctionnement, rendre compatible avec l'environnement, et faire de l'innovation.

L'Assurance Qualité :

- C'est donné confiance à l'équipe projet, au client, et de la hiérarchie vers le chef de projet.
- Respecter les coût et les délais.

Le Manuel Qualité : est à la disposition de l'entreprise pour détailler les méthodes de travail.

Le Plan d'Assurance Qualité Logiciel (PAQL) : Il contient une partie Qualité spécifique au projet et il est rédigé par le responsable qualité.

Le responsable Qualité : Il rédige le PAQL, assiste le chef de projet pour son application, et est un interlocuteur responsable Qualité entre le client et le fournisseur.

1.10. / Le langage Z

On va spécifier ce que l'on veut obtenir d'un logiciel par une vulgarisation mathématique.

1.10.1./ Introduction

Ce sont des techniques (les méthodes formelles) qui vont appliquer les services mathématiques pour développer les systèmes informatiques.

Le rôle des mathématiques :

- C'est la précision, ce que comprends le client et ce que l'on comprend.
- C'est la concision, dire beaucoup avec peu de mot et de manière précise.
- C'est la clarté et l'abstraction, on va se focalisé sur les aspects essentiel du problème.
- C'est l'indépendance par rapport au langage naturel, c'est universellement écrit pour toutes les populations (Prouver).

Les mathématiques discrètes s'intéressent beaucoup aux ensembles et à la logique par rapport à l'opposition aux nombres.

Créé par Raymond ABRIAL en France, puis développé à Oxford par l'équipe du Pr. HOARE. C'est une méthode formelle, c'est-à-dire un ensemble de techniques qui appliquent des principes mathématiques pour spécifier des systèmes informatiques.

1.10.2. / Les types

Ensemble : collection d'éléments d'un même type. Le type évite le paradoxe mathématique et permet de vérifier que les énoncés fait sur un ensemble ont un sens. Outils logiques. Vérification de la cohérence syntaxique.

Exemple : ensemble N : 0 ;1 ;2 ;3 ;4 ;...
ensemble N_1 : 1 ;2 ;3 ;4 ;...-{0}

Type construit : type prédéfini.

Un seul type construit : le type entier relatif, noté Z. N est inclus dans Z.

$N_1 = N - \{0\}$
$Z_1 = Z - \{0\}$

On a les 5 opérateurs de base : +, -, /, *, mod (modulo : le reste d'une division).

Type de base (ensembles donnés) : on le déclare sans s'occuper de ce qu'il est vraiment.

```
[IMMATRICULATION] ensemble de toutes les immatriculations possibles.
[CONDUCTEURS] désigne tous les conducteurs de véhicules.
```

Type libre (énumératif) :

Tous les éléments constituent un ensemble (type Libre ::= élément 1 | élément 2 | élément 3...).

```
[REACTION] ::= oui/non
[STATUT] ::= EnService/Libre/Réservé/HorsService
[UE] ::= F/GB/B/NL/L/I/D/P/E
```

Déclaration de variable : `chauffeur : CONDUCTEUR`

Affectation : `résidence = F`

1.10.3. / Ensemble de valeurs

`Benelux` : $^{|P}$`UE` = sous-ensemble de UE
UE ::= B | NL | F | L | D ... | GR

1.10.4. / Ensemble de constante

`Benelux= { B, NL, L } = { NL, L, B } = … = { B, NL, L, L } = … =` $^{|P}$`UE // Benelux.`

L'ordre et le dupliquât n'ont pas d'importance.

Ensemble vide : ∅ ou {}

Singleton : `{ E } (ensemble à 1 élément)`

1.10.5. / Opérateurs

```
 =, ≠,<, ≤, >, ≥, ∈, ∉, ⊄, ⊂, ⊆, ∪, ∩, \
E ⊂ F => E ≠ F  == inclusion stricte
E ⊆ F => E = F possible
```

`F∉{B, NL, L}` vrai
`USA∈{B, NL, L}` incorrect car USA n'est pas du type UE mais pas faux

Union distribuée : `∪{{B, NL, L}, {F, D, L, I}, {B, E, P}} = {B, NL, L, F, D, I, E, P }`

Ensemble de parties : P{B ;NL ;L} = {0;{B};{NL};{L};{B;NL};{B;L};{NL;L};{B;NL;L}}

nota : #(PE) = $2^{\#E}$

Equivalence : ≙. Ex : A+B ≙ B+A

Inclusion : $\subset et \subseteq$ (strictement inclus). Ex : $A \subseteq A$ et $A \not\subset A$

Différence : {B ;D ;DK ;FI}\{B ;D ;GB}={DK ;F ;I}

Union : {B ;D ;DK} ∪ {D ;DK ;I} = {D ; B ; DK ; I}

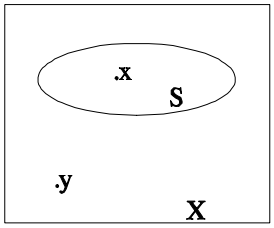
Intersection : {B ;D ;DK} ∩ {D ;DK ;I} = {D ; DK}

1.10.6. / Diagramme de Venn

[X]

s : ℙX
x : X
x ∈ S
y ∉ S

➔



1.10.7./ Intervalle de nombre

n..p

Exemple : 5..8 = {5 ;6 ;7 ;8}

1.10.8./ Disjoint

Exemple : Disjoint <E ;F ;G> -> EF n'ont pas d'élément commun ; FG n'ont pas d'élément commun ; EG n'ont pas d'élément commun.

1.10.9./ Partition

Exemple : <A ;B ;C> Partition <E> -> disjoint <A ;B ;C> -> A U B U C = E

1.10.10. / Taille, cardinalité

Nombre d'éléments distincts d'un ensemble. Noté #.

ℙBenelux = ℙ{{}, {B}, {NL}, {L}, {B, NL}, {B, L}, {NL, L}, {B, NL, L}}

#ℙBenelux = $2^{\#Benelux}$

Exemple : #{B ;NL ;L}=3 ; #{F} = 1 ; #{} = 0 ; #F = NULL car F n'est pas un ensemble.

1.10.11. / Définition en compréhension

Evite l'énumération. Syntaxe : `{ déclaration | contrainte.expression }`= ce que l'on recueille.

Entiers naturels pairs : `{ n:N | Pair(n).n }`
Carré d'entiers naturels pairs : `{ n:N | Pair(n).n² }`
Carré d'entiers naturels : `{ n:N.n² }`

L'ensemble des entiers compris entre n et p compris : `{ x:Z | n≤x≤p.x }`

Partition : A, B, C sont une partition de D si et seulement si ∩{A, B, C}=∅ et ∪{A, B, C}=D

Exemple 1 : $\{\cap:N|Pair(n).n\}$: ensemble de nombre n paire de N.
Exemple 2 : $\{n:N|Pair(n).nxn\}$: ensemble de nombre n de N dont les carrés sont paires.
Exemple 3 : $\{n:N.nxn\}$: ensemble des carrés de N.

1.10.12. / Calcul propositionnel (Algèbre de Boole)

Deux valeurs : 0, faux, false …
1, vrai, true, …

Opérateurs :

Négation : ¬ Conjonction : ∧ (et) Disjonction : ∨ (ou) Implication : =>

P	¬P
0	1
1	0

P	Q	P∧Q
0	0	0
0	1	0
1	0	0
1	1	1

P	Q	P∨Q
0	0	0
0	1	1
1	0	1
1	1	1

P	Q	P=>Q
0	0	1
0	1	1
1	0	0
1	1	1

```
P∧(Q∧R) ⇔ (P∧Q)∧R
P∨(Q∨R) ⇔ (P∨Q)∨R

P∧(Q∨R) ⇔ (P∧Q)∨(P∧R)
P∨(Q∧R) ⇔ (P∨Q)∧(P∨R)

¬(¬P) ⇔ P
P∨¬P ⇔ vrai
P∧¬P ⇔ faux

P∨P ⇔ P
P∧P ⇔ P

P∨vrai ⇔ vrai
P∨faux ⇔ P
```

```
P∨(P∧Q) ⇔ P
P∧(P∨Q) ⇔ P

P∧vrai ⇔ P
P∧faux ⇔ faux
```

Lois de De Morgan :

```
¬(P∧Q) ⇔ ¬P∨¬Q
¬(P∨Q) ⇔ ¬P∧¬Q
```

$\overline{(P\&Q)} \Leftrightarrow \overline{P} ou \overline{Q}$

$\overline{(P ou Q)} \Leftrightarrow \overline{P}\&\overline{Q}$

Priorité décroissante : ¬, ∧, ∨, =>

Relations calcul propositionnel/Z :

```
[X} tout ensemble
S, T : PX
S∪T == {x:X | x∈S ∨ x∈T.x}
S∩T == {x:X | x∈S ∧ x∈T.x}
s\T == { x:X | x∈S ∧ x∉T.x }
```

1.10.13. / Les schémas

Ils permettent de faire un découpage modulaire des définitions construite pour pouvoir les réutiliser plus facilement. Les déclarations et prédicats d'un schéma restent locaux au schéma.

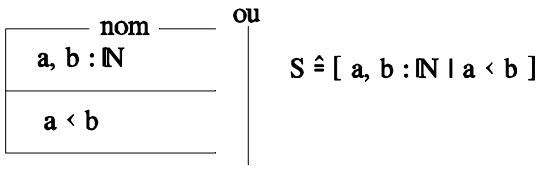
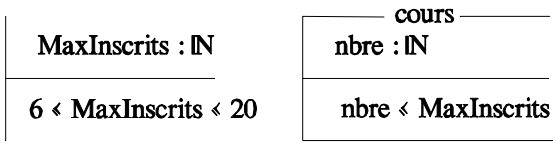


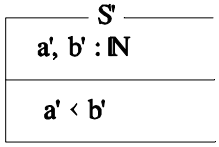
Schéma axiomatique : il est anonyme et considéré comme global :



Les ornement :

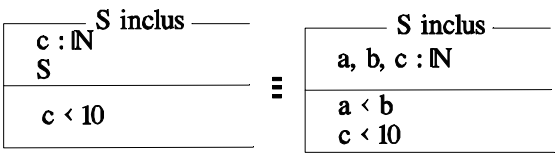
C'est la désignation de la valeur du schéma après une opération.

Si la partie déclarative contient plusieurs lignes, on considère qu'elles sont terminées par ";".
Si la partie prédicative contient plusieurs lignes, on considère qu'elles sont terminées par "∧".

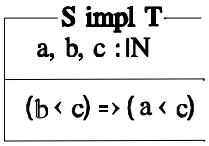


Calculs de schéma :

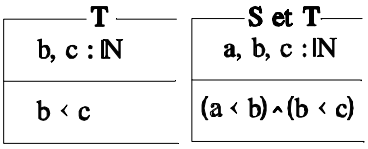
Inclusion :



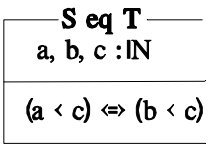
Implication :



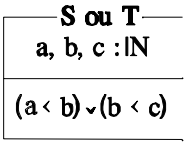
Conjonction :



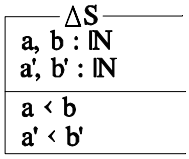
Equivalence :



Disjonction :

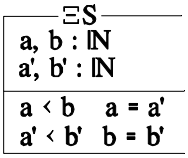


Convention delta : Les propriétés des variable en entrée et en sortie sont identique :



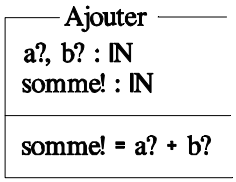
$\Delta S \triangleq S \wedge S'$

Convention KSI : Les propriétés et les valeurs des variables sont invariantes :



Conjonction avec son ornement :

"!" et "?" font partie du nom.
"!" = sortie.
"?" = entrée.



2. TRAVAUX DIRIGES

2.1. Exercice 1

Une manifestation annuelle a lieu le dernier week-end de septembre. Elle débute le vendredi et finit le dimanche. A quelle date débute-elle si le 30 est un lundi, dimanche, samedi, vendredi ? Proposer une formulation non ambiguë de la date de cette manifestation.

Solution : le 30 est un — Elle débute le

lundi	27
dimanche	28
samedi	22? 29?
vendredi	23? 30?

Il faut spécifier ce qu'est le dernier week-end. dernier dimanche de septembre $\subset$ week-end.

2.2. Exercice 2

Une personne passe à son bureau le vendredi 1er janvier. Elle laisse comme message "Je suis absent jusqu'à mercredi prochain." Un collaborateur arrive lundi 4 janvier et lit le message. A quelle date fixe-t-il le retour ?

Solution :

1ere ambiguïté : mercredi inclus dans le congé ou pas ?
2eme ambiguïté : "prochain" ➔ mercredi 6
jeudi 7
mercredi 13
jeudi 14

2.3. Exercice 3

"Les enregistrements d'entrée d'un programme devront être triés dans l'ordre croissant des clés." Que peut-on spécifier ?

Solution :

Rien sans savoir si les enregistrements seront triés avant le traitement ou par le programme.

2.4. Exercice 4

Programmation d'un magnétoscope : date, heure départ, heure fin, n° de canal. Date : mm/jj. Temps : hh.mm. Gestion du jour suivant. Programmes mémorisables.

a / la date de début n'existe pas. (31 avril).
b / 29 février non traité.
c / les plages se chevauchent.
d / Date début = 31/12.
e / Programmes non dans l'ordre chronologique.

Qu'est-ce qui se passe à l'exécution ?

Solution :

a / ça ne marche pas.
b / ça ne marche pas. (pas d'année dans la date).
c / ambigu.
d / Ok.
e / Ok.

2.5. Exercice 5

Un système informatique. Une personne est soit connectée, soit déconnectée. Décrire avec Z cette situation.

```
[PERSONNE] ensemble des personnes identifiées de façon unique

user, connected : ℙPERSONNE
connected ⊆ user
```

Limite max entre 32 et 128 utilisateurs connectés. On a deux groupes d'utilisateurs : employés et clients.

```
Limite : N
limite∈32..128
#connected ≤ limite

clients, employés : ℙuser
clients ∪ employés = user
clients ∩ employés = {}
```

Tous les connectés sont des employés. Il y a plus de clients que d'employés.

```
connected ⊆ employés
#clients > #employés
```

#connected ≤ #users? Oui car connected ⊆ users

Définir l'état initial :

```
connected = ∅
user = ∅
```

Définir la création d'un user, la création ne connecte pas :

```
P : PERSONNE
users' = users ∪ {P}
connected' = connected
Pré condition : P∉users
```

Suppression, l'utilisateur n'est pas connecté :

```
P : PERSONNE
P ∈ user
P ∉ connected
user' = user \ {P}
connected' = connected
```

Connexion/Deconnexion :

a/
```
P ∈ user
P ∉ connected
#connected < limite
connected' = connected ∪ {P}
user' = user
```

b/
```
P ∈ connected
connected' = connected \ {P}
user' = user
```

2.6. Exercice 6

Enregistrement de passagers à bord d'un avion. Places non numérotées. "1er arrivé, 1er servi". Capacité fixe. Type prédéfini [PERSONNE], identifiées par le n° carte d'identité. Trouver les propriétés invariantes du système.

```
[PERSONNE] : numéro carte d'identité
capacité : N

embarqués : |PPERSONNE
#embarqués ≤ capacité
```

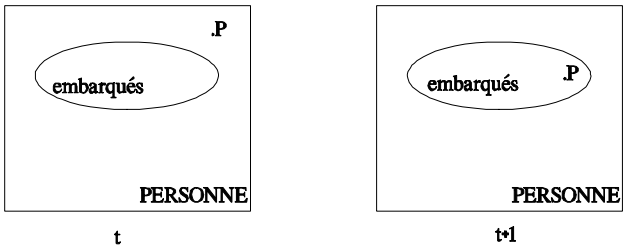
Propriétés invariantes :

Etat initial : embarqués=∅ vrai car capacité∈N (et pas Z)

Embarquement :

P : PERSONNE
embarqués' = embarqués∪{P}

Pré conditions : #embarqués<capacité
P∉embarqués



Débarquement : embarqués' = embarqués\{P}
Pré conditions : P∈embarqués

Interrogations :
n : N
n := #embarqués
embarqués' = embarqués (pour dire que l'interrogation n'a pas modifié l'ensemble embarqués.)

REPONSE ::= oui/non
réponse : REPONSE
((P∈embarqués ET réponse=oui) OU (P ∉ embarqués ET réponse = non))

2.7. Exercice 7

Démontrer : $(P \vee Q) \wedge \neg(P \wedge Q) \Leftrightarrow (P \wedge \neg Q) \vee (\neg P \wedge Q)$

Solution :

$(P \vee Q) \wedge \neg(P \wedge Q)$
$\Leftrightarrow ((P \vee Q) \wedge \neg P) \vee ((P \vee Q) \wedge \neg Q)$
$\Leftrightarrow ((P \wedge \neg P) \vee (Q \wedge \neg P)) \vee ((P \wedge \neg Q) \vee \neg Q)$
$\Leftrightarrow (\varnothing \vee (Q \wedge \neg P)) \vee ((P \wedge \neg Q) \vee \varnothing)$
$\Leftrightarrow (Q \wedge \neg P) \vee (P \wedge \neg Q)$

2.8. Exercice 8

Démontrer $(P \Rightarrow Q) \wedge (Q \Rightarrow P) \equiv (P \Leftrightarrow Q)$

Solution :

P=>Q	Q=>P	(P=>Q) ∧(Q=>P)	P⇔Q
1	1	1	1
0	1	0	0
1	0	0	0
1	1	1	1

2.9. Exercice 9

Simplifier ⌉(P∉embarqués ∧ #embarqués<capa)

Solution :

⌉(P∉embarqués ∧ #embarqués<capa)
⇔⌉(P∉embarqués) ∨ ⌉(#embarqués<capa)
⇔ P∈embarqués ∨ #embarqués≥capa

2.10. Exercice 10

(a∧b)∨(a∧c)∨(a∧⌉c) ⇔ a∧(b∨c∨⌉c) ⇔ a

2.11. Exercice 11

Si P∈connected => P∈user, montrer que P∈connected ∧ P∈user ≡ P∈connected

Solution :

P=>Q $\overset{?}{\Leftrightarrow}$ P∧Q≡P

P∧Q≡P si Q est vrai donc si P=>Q

2.12. Exercice 12

On définit le type REPONSE modélisant les réponses à une opération comme suit :

```
REPONSE :: = OK | ABord | PasABord | Plein | DeuxErreurs
```

Donner les spécifications des procédures d'embarquement et de débarquement en tenant compte de cette déclaration.

Solution :

```
P : PERSONNE
r : REPONSE
```

Embarquement :

```
((P ∉ embarques) ∧ (#embarques < capacité) ∧ (embarques' = embarques U {P}) ∧ (r = OK))
∨
((P ∈ embarques) ∧ (#embarques = capacité) ∧ (embarques' = embarques) ∧ (r = DeuxErreurs))
∨
((P ∈ embarques) ∧ (#embarques < capacité) ∧ (embarques' = embarques) ∧ (r = ABord))
∨
((P ∉ embarques) ∧ (#embarques = capacité) ∧ (embarques' = embarques) ∧ (r = Plein))
```

Débarquement :

```
((P ∈ embarques) ∧ (embarques' = embarques \ {P}) ∧ (r = OK))
∨
((P ∉ embarques) ∧ (embarques' = embarques) ∧ (r = PasABord))
```

2.13. Exercice 13

Définir les réponses possibles ainsi que les procédures de création, destruction, connexion et déconnexion d'un utilisateur du système informatique.

Solution :

Les réponses :

```
REPONSE ::= OK | NotUser | AlreadyUser | NotConnected | AlreadyConnected |
NotDeconnected | Connected

P : PERSONNE
r : REPONSE
```

Création d'un user :

```
(((P ∉ users) ∧ (users' = users U {P}) ∧ (r = OK)) ∨
((P ∈ users) ∧ (users' = users) ∧ (r = AlreadyUser))) ∧ (connected' = connected)
```

Destruction :

```
(((P∈ users) ∧ (users' = users \ {P}) ∧ (P ∉ connected) ∧ (r = OK)) ∨
((P ∉ users) ∧ (users' = users) ∧ (r = NotUser))
∨ ((P∈ users) ∧ (P ∈ connected ) ∧ (r = connected ))) ∧ (connected' = connected)
```

Connexion :

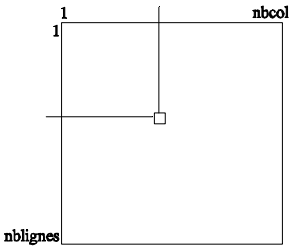
```
(((P ∈ users) ∧ (P ∉ connected) ∧ (r = OK) ∧ (#connected < Limite) ∧ (connected' =
connected U {P}))
∨
((P ∈ connected) ∧ (r = AlreadyConnected) ∧ (connected' = connected))
∨
((P ∉ users) ∧ (r = NotUser) ∧ (connected' = connected))) ∧ (users' = users)
```

Deconnexion :

```
(((P ∈ connected) ∧ (r = OK) ∧ (connected' = connected \ {P}))
∨
((P ∉ users) ∧ (r = NotUser) ∧ (connected' = connected))
∨
((P ∈ users) ∧ (P ∉ connected) ∧ (connected' = connected) ∧ (r = NotConnected))) ∧
(users' = users)
```

2.14. Exercice 14

1/ Définir, à l'aide des schémas, la gestion d'un écran et du curseur. L'écran est défini comme suit :



Avec les définitions suivantes :

[TOUCHE] : ensemble des touches du clavier.

nblignes, nbcol : ℕ
nblignes ≥ 1 nbcol ≥ 1

gauche, droite, haut, bas, debut, retour : TOUCHE
nblignes ≥ 1 nbcol ≥ 1

Curseur
nblignes, nbcol : ℕ
Ligne ∈ 1..nblignes Col ∈ 1..nbcol

2/ Donner par schéma qui spécifie le nombre de lignes qu'il reste sous le curseur par rapport au bas de l'écran, quand on appuie sur une touche.

Solution :

1/

ToucheDebut
Δcurseur touche ? : TOUCHE
Touche ? = debut Ligne' = 1 Col' = 1

ToucheBas
Δcurseur touche ? : TOUCHE
Touche ? = bas Ligne < nblignes Ligne' = ligne + 1 Col' = col

ToucheBasEnBas
Δcurseur touche ? : TOUCHE
Touche ? = bas Ligne = nblignes Ligne' = 1 Col' = col

ToucheRetour
Δcurseur touche ? : TOUCHE
Touche ? = retour [(Ligne < nblignes ∧ ligne' = ligne + 1) ∨ (ligne = nbligne ∧ ligne'=1) col' =1

ToucheDroite
Δcurseur touche ? : TOUCHE
Touche ? = droite col < nbcol Ligne' = ligne Col' = col + 1

ToucheBas ≙
ToucheBas ∨ ToucheBasEnBas

ToucheHaut ≙
ToucheHaut ∨ ToucheHautEnHaut

ToucheGauche
Δcurseur touche ? : TOUCHE
Touche ? = gauche col > 0 Ligne' = ligne Col' = col - 1

Touchehaut
Δcurseur touche ? : TOUCHE
Touche ? = haut Ligne > 0 Ligne' = ligne - 1 Col' = col

TouchehautEnhaut
Δcurseur touche ? : TOUCHE
Touche ? = haut Ligne' = 1 Col' = col

2/

LignesRestantes
Ξ curseur ligne ! : ℕ
Ligne ! = nblignes -- ligne

2.15. Exercice 15

On définit un système informatique avec les schémas suivants :

ordi

user, connected : ℙ PERSONNE

connected ⊆ user

Init

ordi

connected = ∅
user = ∅

REPONSE ::= Ok | DéjàUser | NonUser | DéjàConnected | NonConnected

Définir l'opération de création d'un nouvel utilisateur en utilisant le même formalisme.

Solution :

CréationSansErreur

Δ ordi
r : REPONSE
P? : PERSONNE

P? ∉ user connected' = connected
r! = Ok user' = user ∪ {P?}

CréationSansErreur

Ξ ordi
r : REPONSE
P? : PERSONNE

P? ∈ user
r! = DéjàUser

Création ≙ CréationSansErreur ⌄ ErreurCréation

2.16. Exercice 16

Une société veut revoir sa politique de tests à l'occasion de la sortie d'un nouveau produit dont la taille est de 120 kls et la complexité moyenne (type P = semi détaché).

Calculer :

1/. L'effort global de test (50% TU + 50% Codage)
2/. La durée de la période de test
3/. L'effort de test moyen par ligne (1h.m ≈ 21 H.jours)
4/. L'effort par test unitaire
5/. Nombre d'essais un essai = 0,5 j
6/. L'effort par test d'intégration (1 test pour 1 kls)
7/. Nombre d'essais (1 essai = 3 H.jours)

Solution :

1/. $E=3\times 120^{1,12}=639,44$ H.mois ≈ 640 H.mois. **$E=3,0x(kdsl)^{1.12}$**

suivant le §3.1 des annexe de COCOMO sur le caculs des "Efforts" on a :
Intégration = 28 %,
Code & Unit Test = 31% => Unit Test = 15,5% (car 50%).

E_gTest = 640 (15,5 + 28) / 100 = 278,4 H.mois.

$T=2,5\times 640^{0,35}=24$ mois. **$T=2,5x(kdsl)^{0.35}$**

2/. Au §3.3 l'annexe COCOMO sur les temps de développement, on a :

le temps pour le codage + TU = T x 0,44 = 10,56 / 2 = 5 mois.
Le temps pour les TI = 24 x 0,29 ≈ 7 mois
donc le temps pour TU + TI = 12 mois.

3/. L'effort de test moyen par ligne est : (278x21)/120000 =0,0487 H.j/ls

4/. L'effort pour les TU : 640 x 0,32 / 2 = 99,2 H.mois (50%)
soit (99,2x21)/120000 = 0,0173 H.j/ls d'où 1,73 H.jours pour 100 ls

5/. 100 ls => 1,73j => (1, 73 / 0,5) x 1200 = 4000 essais.

6/. L'effort pour les TI : 640 x 0,28 = 179 H.m
soit (179,2x21)/120000 = 0,0314 H.j/ls d'où 31,4 H.jours pour 100 ls.

7/. 1 kls => 31,4 H.j => (31,4 / (1,73/0,5)) x 1200 = 12000 essais.

2.17. Exercice 17

Le logiciel LOG de la société SOC a fait l'objet du devis suivant :
CG (conception générale) : 4 mois, Codage : 12 mois, Qualification (tests) : 2 mois

Le directeur technique (qui veut optimiser la rentabilité) estime qu'en moyenne, 10 personnes sont amplement suffisantes. Durée de développement : 18 mois. La montée en puissance de l'équipe de développement est prévue comme suit :

T_0 : 4 analystes dont le chef de projet
T_0+4 : 10 personnes
T_0+6 : complément d'effectif

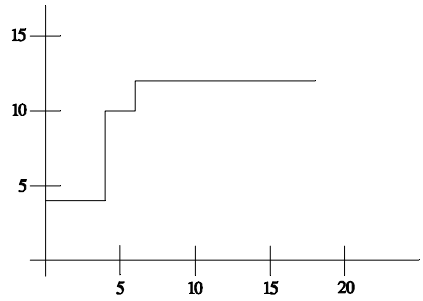
A/. Calculé l'effort alloué (en H.mois) pour le projet
B/ Tracer le diagramme de la montée en puissance de l'effort en fonction du temps.
C/ Le logiciel LOG est de type S et de taille 32 Kls. Calculer l'effort E et le temps de développement Tdev pour chaque phase.
D/ Calculer le nombre de ligne de code prévisible.

Solution :

A/. L'effort brut : 180 H.mois.
On enlève un mois de vacances par personne => budget net = 170 H.m (les gens partent en congé en fin de contrat lors de la 2eme année).

Soit : $4 \times 4 + 10 \times 2 + 12 \times x = 170 \Leftrightarrow x = 11{,}16 \cong 12$ H.mois.

B/.



C/. On a un effort total de 170 H.mois.
Or suivant le tableau 6-8 de COCOMO pour les programmes de 32kls de type S on a un total de 106% d'effort (6% en plus pour la phase préliminaire d'expression des besoins).
La répartition en fonction des phases est la suivante :

Expression des besoins (plans & requirements) :	(170/1,06)x6% =	9,6H.mois.
Conception générale (Product design) :	(170/1,06)x16% =	25,7H.mois.
Codage (programing) :	(170/1,06)x62% =	99,4H.mois.
Qualification (Tests) :	(170/1,06)x22% =	35.3H.mois.

D/. On ne considère pas que l'expression des besoins n'appartient pas à la phase de codage.

170-9,6 = 2,4(X)^{1,05} soit (170-9,6)/2,4 = (X)^{1,05} d'où $X = \left(\frac{170-9,6}{2,4}\right)^{1/1,05}$ = 54,5 kls

3. ANNEXES : Les paramètres du modèle COCOMO

3.1. Calcul des Efforts et des Temps de développement

MODE	EFFORT GLOBAL (H.mois)	EFFORT NOMINAL (H.mois)	Temps de développement (mois)
Organique (type S)	$E = 2{,}4.(KLS)^{1{,}05}$	$E_{nom} = 3{,}2.(KLS)^{1{,}05}$	$TDEV = 2{,}5.(KLS)^{0{,}38}$
Semi détaché (type P)	$E = 3{,}0.(KLS)^{1{,}12}$	$E_{nom} = 3{,}0.(KLS)^{1{,}12}$	$TDEV = 2{,}5.(KLS)^{0{,}35}$
Imbriqué (type E)	$E = 3{,}6.(KLS)^{1{,}20}$	$E_{nom} = 2{,}8.(KLS)^{1{,}20}$	$TDEV = 2{,}5.(KLS)^{0{,}32}$

KLS = Kilo Ligne de Sources.

3.2. Détails des besoins pour un Projet logiciel de type S (organique)

Taille du projet	Effort (H.mois)	Productivité (LS/H.mois)	Durées (mois)	Taille de l équipe (H)
Petit (2 KLS)	5,0	400	4,6	1,1
Intermédiaire (8 KLS)	21,3	376	8,0	2,7
Moyen (32 KLS)	91,0	352	14,0	6,5
Grand (128 KLS)	392,0	327	24,0	16,0

3.3. Details des Efforts et des Temps de développement (tous modes)

Nota : toutes les valeurs sont en %.

MODE	Détails des Efforts	Petit (2 KLS)	Intermédiaire (8 KLS)	Moyen (32 KLS)	Grand (128 KLS)	Très grand (512 KLS)
Organique (type S)	*Planification, Spécification*	*6*	*6*	*6*	*6*	-
	Conception générale	16	16	16	16	-
	Programmation	68	65	62	59	-
	conception détaillée	26	25	24	23	-
	Ecriture du code et test unitaires	42	40	38	36	-
	Intégration et tests	16	19	22	25	-
Semi détaché (type P)	*Planification, Spécification*	*7*	*7*	*7*	*7*	*7*
	Conception générale	17	17	17	17	17
	Programmation	64	61	58	55	52
	conception détaillée	27	26	25	24	23
	Ecriture du code et test unitaires	37	35	33	31	29
	Intégration et tests	19	22	25	28	31
Imbriqué (type E)	*Planification, Spécification*	*8*	*8*	*8*	*8*	*8*
	Conception générale	18	18	18	18	18
	Programmation	60	57	54	51	48
	conception détaillée	28	27	26	25	24
	Ecriture du code et test unitaires	32	30	28	26	24
	Intégration et tests	22	25	28	31	34
MODE	**Détails des Temps de développement**	**Petit (2 KLS)**	**Intermédiaire (8 KLS)**	**Moyen (32 KLS)**	**Grand (128 KLS)**	**Très grand (512 KLS)**
Organique (type S)	*Planification, Spécification*	*10*	*11*	*12*	*13*	-
	Conception générale	19	19	19	19	-
	Programmation	63	59	55	51	-
	Intégration et tests	18	22	26	30	-
Semi détaché (type P)	*Planification, Spécification*	*16*	*18*	*20*	*22*	*24*
	Conception générale	24	25	26	27	28
	Programmation	56	52	48	44	40
	Intégration et tests	20	23	26	29	32
Imbriqué (type E)	*Planification, Spécification*	*24*	*28*	*32*	*36*	*40*
	Conception générale	30	32	34	36	38
	Programmation	48	44	40	36	32
	Intégration et tests	22	24	26	28	30

3.4. Détails des phases d un projet logiciel (tous modes)

Notas : toutes les valeurs sont en %.
P : Petit.
I : Intermédiaire.
M : Moyen.
G : Grand.
TG : Très Grand.

Mode : Organique (type S)

Phases	Planification et spécification					Conception générale					Programmation					Intégrations et tests					Développement					Maintenance				
Taille du projet logiciel	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG
Taux global	-	-	6	-	-	-	-	16	-	-	68	65	62	59	-	16	19	22	25	-	-	-	-	-	-	-	-	-	-	-
Planification et spécifications	-	-	46	-	-	-	-	15	-	-	-	-	5	-	-	-	-	3	-	-	-	-	6	-	-	-	-	7	-	-
Conception générale	-	-	20	-	-	-	-	40	-	-	-	-	10	-	-	-	-	6	-	-	-	-	14	-	-	-	-	13	-	-
Programmation	-	-	3	-	-	-	-	14	-	-	-	-	58	-	-	-	-	34	-	-	48	47	46	45	-	45	44	43	42	-
Planification des tests	-	-	3	-	-	-	-	5	-	-	-	-	4	-	-	-	-	2	-	-	-	-	4	-	-	-	-	3	-	-
Vérification et validation	-	-	6	-	-	-	-	6	-	-	-	-	6	-	-	-	-	34	-	-	10	11	12	13	-	10	11	12	13	-
Version finale	-	-	15	-	-	-	-	11	-	-	-	-	6	-	-	-	-	7	-	-	-	-	7	-	-	-	-	7	-	-
Assurance Qualités	-	-	2	-	-	-	-	2	-	-	-	-	6	-	-	-	-	7	-	-	-	-	5	-	-	-	-	5	-	-
Documentation utilisateur	-	-	5	-	-	-	-	7	-	-	-	-	5	-	-	-	-	7	-	-	-	-	6	-	-	-	-	10	-	-

Mode : Semi détaché (type P)

Phases	Planification et spécification					Conception générale					Programmation					Intégrations et tests					Développement					Maintenance				
Taille du projet logiciel	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG
Taux global	7	7	7	7	7	17	17	17	17	17	64	61	58	55	52	19	22	25	28	31	-	-	-	-	-	-	-	-	-	-
Planification et spécifications	48	47	46	45	44	12.5	12.5	12.5	12.5	12.5	4	4	4	4	4	2.5	2.5	2.5	2.5	2.5	5	5	5	5	5	6.5	6.5	6.5	6	6
Conception générale	16	16.5	17	17.5	18	41	41	41	41	41	8	8	8	8	8	5	5	5	5	5	13	13	13	13	13	12	12	12	12	12
Programmation	2.5	3.5	4.6	5.5	6.5	12	12.5	13	13.5	14	56.5	56.5	56.5	56.5	56.5	33	35	37	39	41	45	45	44.5	44.5	44.5	41.5	41.5	41	41	41
Planification des tests	2.5	3	3.5	4	4.5	4.5	5	5.5	6	6.5	4	4.5	5	5.5	6	2.5	2.5	3	3	3.5	4	4	4.5	5	5.5	3	3	3.5	4	4.5
Vérification et validation	6	6.5	7	7.5	8	6	6.5	7	7.5	8	7	7.5	8	8.5	9	32	31	29.5	28.5	27	11	12	13	13.5	14	11	12	13	13.5	14
Version finale	15.5	14.5	13.5	12.5	11.5	13	12	11	10	9	7.5	7	6.5	6	5.5	8.5	8	7.5	7	6.5	8.5	8	7.5	7	6.5	8.5	8	7.5	7	6.5
Assurance Qualités	3.5	3	3	3	2.5	3	2.5	2.5	2.5	2	7	6.5	6.5	6.5	6	8.5	8	8	8	7.5	6.5	6	6	6	5.5	6.5	6	6	6	5.5
Documentation utilisateur	6	6	5.5	5	5	8	8	7.5	7	7	6	6	5.5	5	5	8	8	7.5	7	7	7	7	6.5	6	6	11	11	10.5	10.5	10.5

Mode : Imbriqué (type E)

Phases	Planification et spécification					Conception générale					Programmation					Intégrations et tests					Développement					Maintenance				
Taille du projet logiciel	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG	P	I	M	G	TG
Taux global	8	8	8	8	8	18	18	18	18	18	60	57	54	51	48	22	25	28	31	34	-	-	-	-	-	-	-	-	-	-
Planification et spécifications	50	48	46	44	42	10	10	10	10	10	3	3	3	3	3	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6
Conception générale	12	13	14	15	16	42	42	42	42	42	6	6	6	6	6	4	4	4	4	4	12	12	12	12	12	11	11	11	11	11
Programmation	2	4	6	8	10	10	11	12	13	14	55	55	55	55	55	32	36	40	44	48	42	43	43	44	45	38	39	39	40	41
Planification des tests	2	3	4	5	6	4	5	6	7	8	4	5	6	7	8	3	3	4	4	5	4	4	5	6	7	3	3	4	5	6
Vérification et validation	6	7	8	9	10	6	7	8	9	10	8	9	10	11	12	30	28	25	23	20	12	13	14	14	14	12	13	14	14	14
Version finale	16	14	12	10	8	15	13	11	9	7	9	8	7	6	5	10	9	8	7	6	10	9	8	7	6	10	9	8	7	6
Assurance Qualités	5	4	4	4	3	4	3	3	3	2	8	7	7	7	6	10	9	9	9	8	8	7	7	7	6	8	7	7	7	6
Documentation utilisateur	7	7	6	5	5	9	9	8	7	7	7	7	6	5	5	9	9	8	7	7	8	8	7	6	6	12	12	11	11	11

3.5. Coefficient multiplicatifs pour les calculs des Efforts

Facteurs de coûts	Niveaux					
	Très bas	bas	moyen	grand	très grand	très très grand
Caractéristiques du produit :						
RELY : fiabilité requise	0.75	0.88	1	1.15	1.40	-
DATA : taille de la base de donnée	-	0.94	1	1.08	1.16	-
CPLX : complexité globale	0.70	0.85	1	1.15	1.30	1.65
Caractéristiques de la machine cible :						
TIME : temps d'exécution	-	-	1	1.11	1.30	1.66
STOR : mémoire principale	-	-	1	1.06	1.21	1.56
VIRT : Stabilité de l'environnement	-	0.87	1	1.15	1.30	-
TURN : Interactivité des moyens de développement	-	0.87	1	1.07	1.15	-
Caractéristiques du personnel :						
ACAP : compétence et cohérence de l'équipe d'analystes	1.46	1.19	1	0.86	0.71	-
AEXP : expérience du domaine d'application	1.29	1.13	1	0.91	0.82	-
PCAP : expérience des programmeurs	1.42	1.17	1	0.86	0.70	-
VEXP : connaissance du système d'exploitation	1.21	1.10	1	0.90	-	-
LEXP : connaissance du langage de programmation	1.14	1.07	1	0.95	-	-
Caractéristiques du projet :						
MODP : utilisation des techniques modernes de développement	1.24	1.10	1	0.91	0.82	-
TOOL : niveau de sophistication des outils de développement	1.24	1.10	1	0.91	0.83	-
SCED : délai de mise à disposition	1.23	1.08	1	1.04	1.10	-

3.6. Influence des facteurs de coût dans un développement logiciel

Notas : D = Taille de la base de donnée en octet.
P = Taille du programme en ligne de code.

Facteurs de coûts	Niveaux					
	Très bas	bas	moyen	grand	très grand	très très grand
Caractéristiques du produit :						
RELY (fiabilité requise)	peu d'influence	influence basse, facilement compensable	influence moyenne, compensable	grande influence sur les coûts	risques humain important	-
DATA (taille de la base de donnée)	-	$\frac{D}{P}<10$	$10\leq\frac{D}{P}<100$	$100\leq\frac{D}{P}<1000$	$\frac{D}{P}\geq1000$	-
CPLX (complexité globale)	-	-	-	-	-	-
Caractéristiques de la machine cible :						
TIME (temps d'exécution)	-	-	≤ 50% d'utilisation du temps d'exécution	70%	85%	95%
STOR (mémoire principale)	-	-	≤ 50% d'utilisation de la mémoire	70%	85%	95%
VIRT (Stabilité de l'environnement)	-	au plus mise à jours tous les ans, au moins tous les mois	au plus mise à jours tous les 6 mois, au moins : toutes les 2 semaines	au plus mise à jours tous les 6 mois, au moins : toutes les 2 semaines	au plus mise à jours tous les 2 mois, au moins : toutes les semaine	-
TURN (Interactivité des moyens de développement)	-	interactif	interactivité moyenne < 4h	4 - 12 h	> 12h	-
Caractéristiques du personnel :						
ACAP (compétence et cohérence de l'équipe d'analystes)	15%	35 %	55%	75%	90%	-
AEXP (expérience du domaine d'application)	≤ 4 mois d'expérience	1 ans	3 ans	6 ans	12 ans	-
PCAP (expérience des programmeurs)	15%	35 %	55%	75%	90%	-
VEXP (connaissance du système d'exploitation)	≤ 1 mois d'expérience	4 mois	1 ans	3 ans	-	-
LEXP (connaissance du langage de programmation)	≤ 1 mois d'expérience	4 mois	1 ans	3 ans	-	-
Caractéristiques du projet :						
MODP (utilisation des techniques modernes de développement)	Pas utilisées	utilisation sommaire	utilisation courante	utilisation générale	utilisation complète	-
TOOL (niveau de sophistication des outils de développement)	assembleur / désassembleur	+ compilateur / éditeur de liens	+ Profileur et librairies	+ optimiseurs, générateur auto.	+ librairies spec., docs	-
SCED (délai de mise à disposition)	75% du temps nominal	85%	100%	130%	160%	-

4. INDEX

A

B

C

D

E

F

H

I

K

L

M

N

O

P

Q

R

S

T

V

5. GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.
If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.