DI GALLO Frédéric

COURS DE GENIE LOGICIEL

Cycle Probatoire

CNAM BORDEAUX 1999-2000

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

TEST DE LOGICIEL

FIABILITE DU LOGICIEL

GESTION DE PROJET

MAINTENANCE DE LOGICIEL

GESTION DE LA QUALITE

TEST DE LOGICIEL

TEST DE LOGICIEL

GENIE LOGICIEL – CNAM BORDEAUX 1999-2000

TEST DE LOGICIEL

Introduction :

Le test est une activité importante dont le but est d'arriver à un produit « zéro défaut ». C'est la limite idéaliste vers laquelle on tend pour la qualité du logiciel. Généralement 40% du budget global est consacrée à l'effort de test.

I. FONDEMENT DU TEST

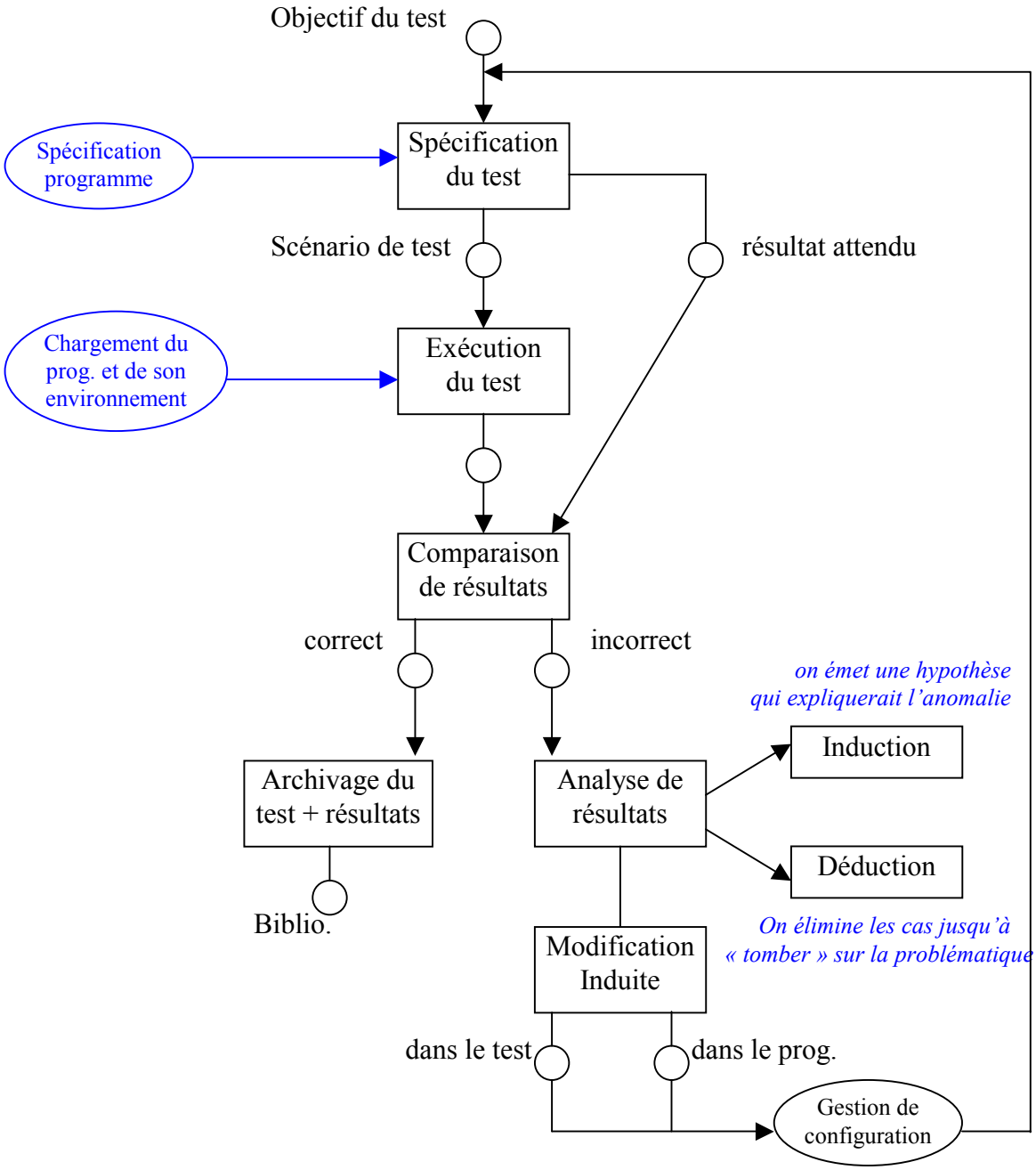
Le test est une recherche d'anomalie dans le comportement de logiciel. C'est une activité paradoxale : il vaut mieux que ce ne soit pas la même personne qui développe et qui teste le soft. D'où le fait qu'un bon test est celui qui met à jour une erreur (non encore rencontrée).

Remarque (difficulté) : il faut arriver à gérer une suite de test la plus complète possible à un coup minimal.

Un test ne peut pas dire « il n'y a pas d'erreur » car il teste le logiciel de façon poussive, plus que dans l'utilisation réelle.

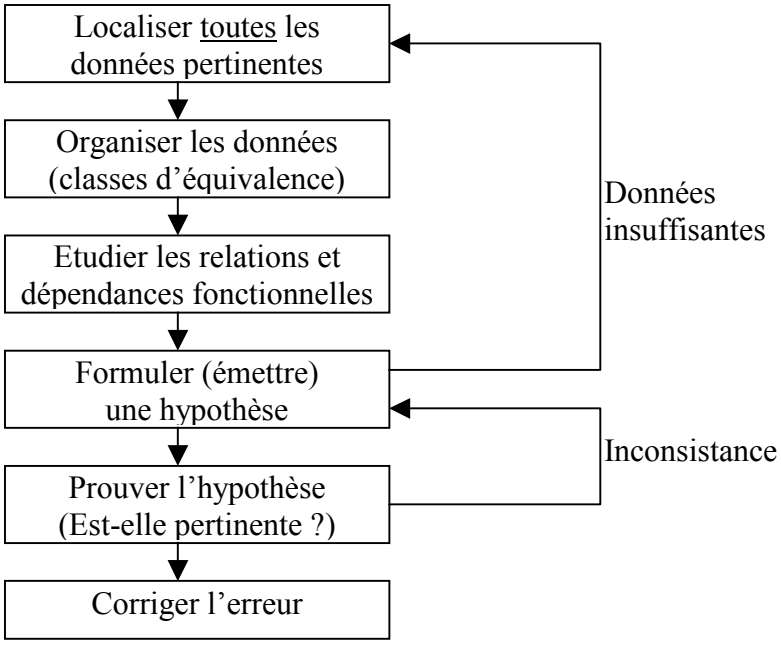
1.1) Cycle de développement de test

Lorsqu'une erreur est détectée alors que commence le débogage, la correction d'une erreur dont la différence avec résultat en du juif est de l'ordre de 0,01% peut prendre… En fait, ce n'est pas fonction de l'importance de l'erreur. Ce qui induit une difficulté concernant la planification du débogage.



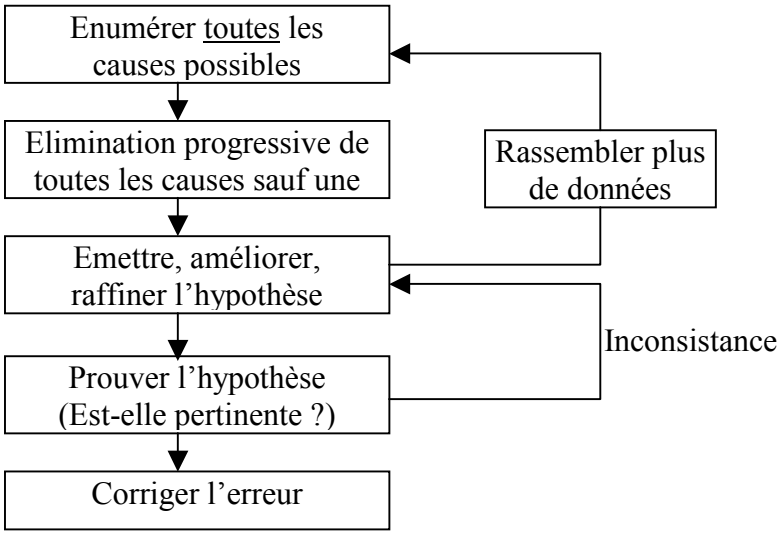
1.2) Mise au point Inductive

On met une hypothèse sur l'ensemble.



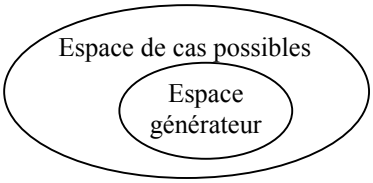
1.2) Mise au point Déductive

On traite chaque cause séparément.



II. TECHNIQUE DE TEST

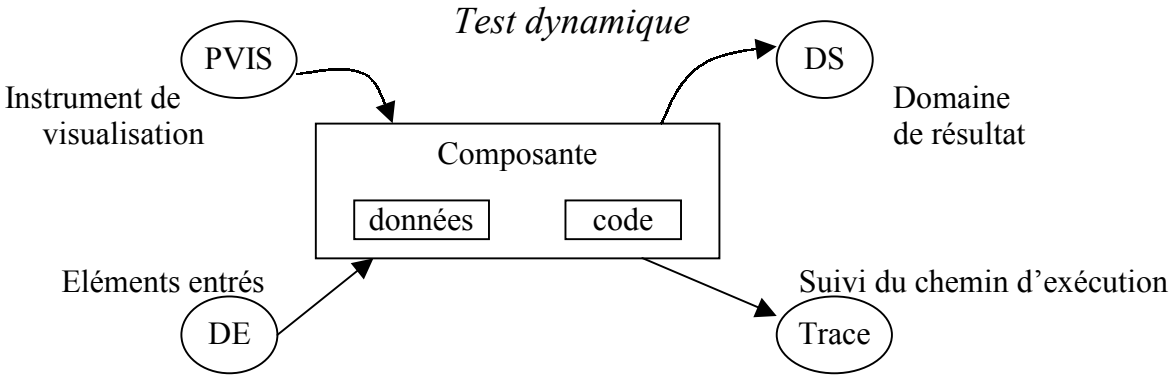
Plusieurs techniques qui dépendent de l'objectif du test. Mais aucune technique ne sera jamais complète. Le problème est de savoir quelle technique nous assure la complétude, car en fait, aucune ne peut le garantir.



Cela revient à échantillonner de façon représentative.

Propriétés recherchées : Si l'espace générateur est couvert alors la probabilité d'une défaillance dans l'espace de cas possible est très faible (inférieure à une limite fixée à l'avance). La difficulté et de faire que l'espace générateur soit consistant et complet.

Les Différentes techniques de test



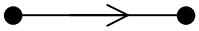
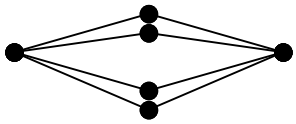
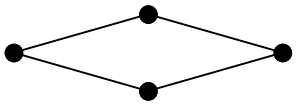
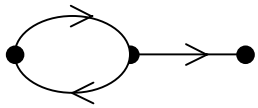
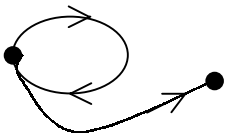
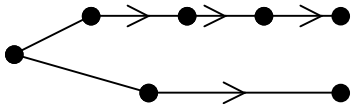
2.1) Test « Boite blanche »

Ce test consiste à analyser la structure interne du programme en déterminant les chemins minimaux. Afin d'assurer que:

- Toutes les conditions d'arrêt de boucle ont été vérifiées.
- Toutes les branches d'une instruction conditionnelle ont été testés.
- Les structures de donne interne ont été testées (pour assurer la validité).

Structures de la représentation de la boîte blanche.

La structures de contrôle se présente sous la forme d'un graphe dit graphe de flot. On représente les instructions comme cela :

➢ Suite linéaire : 	➢ Case : 
➢ If : 	➢ Until : 
➢ While (for) : 	➢ Else : 

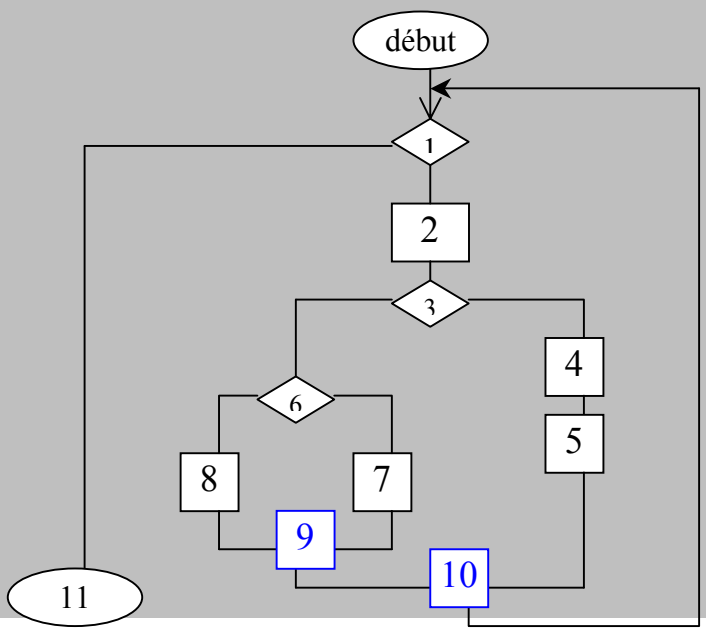
Remarque : If A & B & C ⇔ If A then if B then if C then…

Mesure de complexité de Mac Cabr.

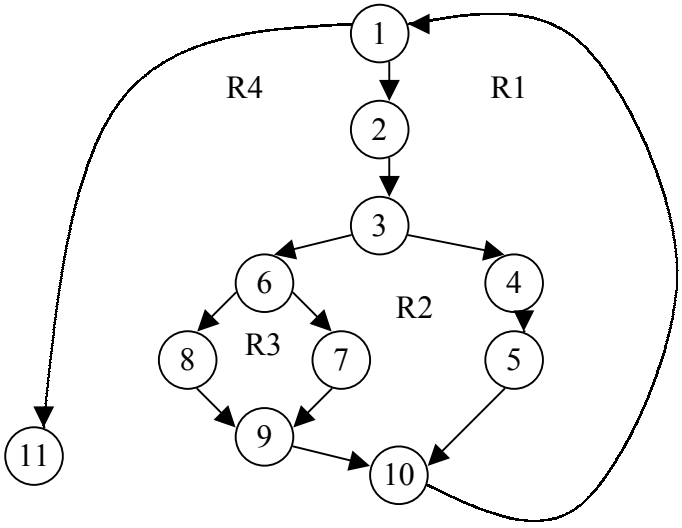
Cette mesure donne le nombre de chemins minimaux. Elle est donnée par la formule suivante qui correspond au nombre de régions du graphe de flot :

Nb.Arcs – Nb.Nœuds + 2 *Nombre cyclomatique*

Exemple : Supposons un programme représenté par l'organigramme suivant:



On en déduit le graphe de flot suivant :



Donc le nombre cyclomatique est : Nb.Arcs – Nb.Nœuds + 2 = 13 – 11 + 2 = **4**

Pour vérifier, on regarde les chemins minimaux (un test par chemin pour tester toutes les possibilités du programme) :

1. 1.11
2. 1.2.3.4.5.10.1.11
3. 1.2.3.6.7.9.10.1.11
4. 1.2.3.6.8.9.10.1.11

Exercice : soit le programme « recherche dichotomique » en langage C:

```
void recherche_dico (elem cle, elem t[], int taille, boolean &trouv, int &A)
  {   int d, g, m;
      g=0; d=taille -1;
      A (d+g) /2;
      if (t[A]= =cle)  trouv=true;
      else trouv=false;
      while (g <=d && !trouv)
          {     m= (d+g) /2;
                if (t[m]= =cle)
                  {     trouv=true;
                        A=m;
                  }
                else  if (t[m]> cle)  g=m+1;
                      else d=m-1;
          }
  }
```

Calculer le nombre cyclomatique de cette procédure.

2.2) Test « Boite noire »

On ignore la structure de codage du logiciel

Principe :

1. On considère le programme dans son aspect fonctionnel et non plus structurel.
2. On partitionne le domaine (DE) en classes.
3. On génère des cas de test aux limites de classe.

Exemple : Soit P un programme. Supposons que les données de P soient des nombre de cinq chiffres. Alors les classes de nombre à cinq chiffres s'obtiennent de la manière suivante:

1. $x < 10\,000$ 2. $10000 \leq x \leq 99\,999$ 3. $X \geq 100\,000$

Les cas de test aux limites de classes sont donc 00 000 et 09 999 pour la première classe, 10 000 et 99 999 pour la deuxième classe et 100 000 pour la troisième.

On a donc à tester les nombres suivants :

<10 000	10 000	50 000	99 999	99 999 <

Exercice : programme « recherche dichotomique » en langage C:

```
void recherche_dico (elem cle, elem t[], int taille, boolean &trouv, int &A)
  {   int d, g, m;
      g=0; d=taille -1;
      A (d+g) /2;
      if (t[A]= =cle)  trouv=true;
      else trouv=false;
      while (g <=d && !trouv)
          {     m= (d+g) /2;
                if (t[m]= =cle)
                  {     trouv=true;
                        A=m;
                  }
                else  if (t[m]> cle)  g=m+1;
                      else d=m-1;
          }
  }
```

Pré-condition :
Taille >= 1 ;
Quelquesoit i : $0 \leq i \leq$ taille – 1
$T[i] \leq t[i+1]$

Post-condition :
(vrai et t(i)=clé) ou
(faux et $0 \leq i \leq$ taille t(i)=clé)

4. Proposer un partitionnement de D.E.

D'après la pré-condition, on en déduit que le programme manipule les tableaux triés non vides (ils contiennent au moins un élément).

1ère classe: tableau de taille = 1 ;

2ème classe: tableau de taille > 1.

5. Proposer un jeu de test

Tableau	Elément	Table	Clé	Sortie (trouv, A)
1 seule valeur	Dans le tableau	[17]	17	(true, 0)
1 seule valeur	Pas dans le tableau	[17]	27	(false, ???)
Plus d'une valeur	1er élément dans le tableau	[3,17,33,42,58]	3	(true, 0)
''	dernier élément dans le tableau	''	58	(true, 4)
''	médian dans le tableau	''	33	(true, 2)
''	non présent dans le tableau	''	1	(false, ???)

6. Donner un exemple, basé sur votre expérience, qui montre l'incomplétude du test B. Noir par rapport au test B. Blanche.

Par exemple, un pointeur non initialisé ne sera pas détecté par le test B. Noire alors qu'il est par le test B. Blanche.

```
Typedef struct cplx                    nombre complexe (partie réelle + imaginaire)
    {    int reel,
         int img ;   } cplx ;

    CPLX * add-cpl (cplx a, cplx b)          addition de nombres complexes
    {    resultat = malloc (sizeof (*cplx)) ;
         resultat.reel = a.reel + b.reel ;
         resultat.img = a.img + b.img ;
         return resultat ;    }
```

Ici, il est possible que se pose un problème d'allocation mémoire. La B. Noire ne le verra pas contrairement à la B. Blanche. Certains disent que l'incomplétude est réciproque, d'autres disent qu'il n'y a pas de raison puisque les DE sont les mêmes.

Logiciel	L	
Spécification	S	
Conception	D	
Codage	C	
	U	Test unitaire (boîte blanche)
	I	Test d'intégration (ici, on s'intéresse à l'architecture d'un module, on vérifie l'adéquation entre les fonctions appelées et les fonctions appelantes : Boîte Noire).
	V	Test de validation (vérifier est-ce que le système construit correspond bien aux besoins exprimés par le client ? Les moyens mis en œuvre sont des notions mathématiques : preuves formelles du programme).
	T	Test du logiciel (environnement réel du logiciel, avec les données du client, sa plate-forme, etc. : test de fiabilité).

III. TD: Test, Vérification et Validation

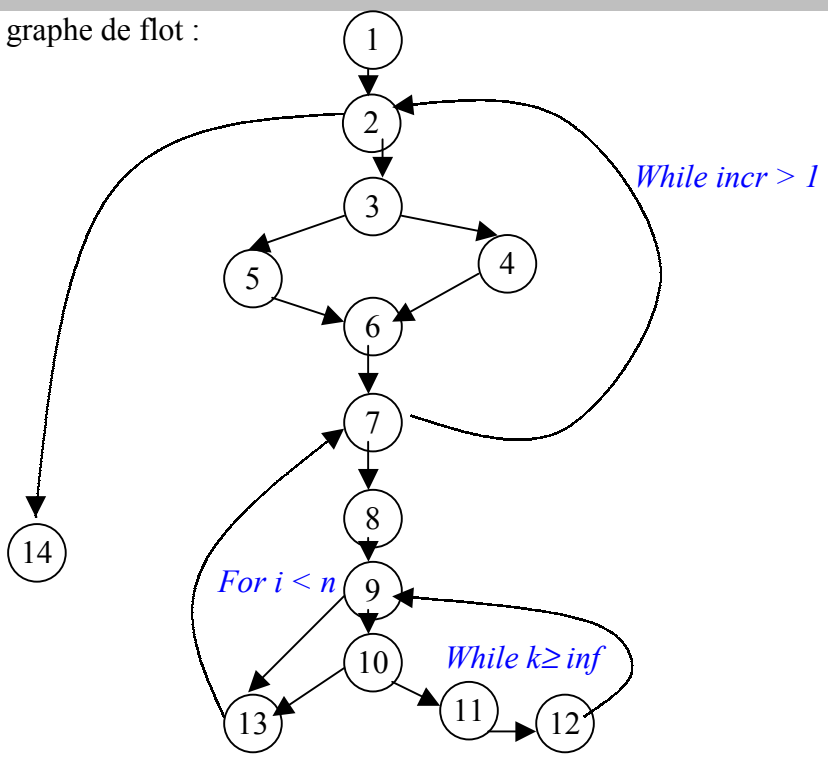
Exercice 1: Test Boîte Blanche

Trouver le nombre cyclomatique du graphe de contrôle associé au programme suivant et donner un ensemble de tests:

```
void tri_shell (tableau t ;int n) {
    element  inf= t [0];
    int incr=n;
    element L;
    int i, k;

    while (incr > 1) {
        if (incr < 5)
            incr=1;
        else incr = (5 * incr-1)/11;
        for ( i = inf; i < n ; i+incr) {
            k = i - incr;
            while( k >= inf) {
                if (L < t [k]) {
                    t [k+incr] = t [k];
                    k= k - incr;    }
                else exit;  }
            t [k+incr]=L;
        }
    }
}
```

On en déduit le graphe de flot :



On a donc 14 noeuds et 18 arcs différentes ce qui donne **Nb Cycl** = 18 – 14 + 2 = **6**

Après avoir trouvé le nombre cyclomatique, il suffit de donner six tableaux correspondants à chaque chemin minimal:

1. un tableau ayant un seul élément
2. un tableau trié avant moins de 5 éléments
3. un tableau non trié ayant moins de 5 éléments
4. un tableau trié ayant plus de 4 éléments
5. un tableau non trié ayant plus de 4 éléments
6. un tableau partiellement trié ayant plus de 4 éléments

Exercice 2: Test statistique

1. Certaines classes de système sont conçues pour supporter une certaine charge. Par exemple, un système de gestion du contrôle aérien, peut être conçu pour traiter cent transactions par seconde. Il a fallu imaginer des tests pour s'assurer que le système supportait bien la charge pour laquelle il était conçu. Ces tests sont appelés « tests de surcharge » et consistent à aller au delà de la charge maximale du système. Le principe: on prévoit une série de tests où la charge augmente progressivement jusqu'à ce que le système tombe en panne. Donner et expliquer deux intérêts du test du surcharge.

a) On teste le comportement du système en cas de panne.
En effet, i1 se peut que dans des circonstances extraordinaire, le système soit plus chargé que prévu. Dans de telles circonstances, il vaut mieux tomber en panne *"doucement"* plutôt que *"brutalement"*. Ces test permettent de vérifier que le système surchargé n'occasionne pas de dégâts irréparables.

b) En surchargeant le système, on peut faire apparaître des défauts qui ne se seraient pas manifestés autrement. Bien que l'on puisse répondre que de tels défaut ont bien peu de chances de causer des pannes en fonctionnement normal, ils peuvent correspondre à des combinaisons peu habituelles qui sont, par coïncidence, semblables aux tests de charge.

c) On peut aussi se servir du test de surcharge pour déterminer une mesure de fiabilité.

2. Décrire comment on peut utiliser un analyseur dynamique pour le test structurel d'un programme. Rappel: les analyseurs dynamique sont des programmes que l'on utilise pour recueillir des informations sur la fréquence d'exécution de chacune des instructions d'un programme.

- D'abord, il faut identifier toutes les instructions de test et d'itération, et ensuite instrumenter chaque instruction du programme.
- Via un pré-processeur, on ajoute ces instructions au langage de haut niveau dans lequel est écrit le programme.
- On compile le langage avec un compilateur standard.
- Lors de l'exécution, les instructions rajoutées viennent stocker les données sur le comportement du programme dans un fichier jouant le rôle d'historique.

L'analyse de ce fichier permet d'identifier les parties du programme qu'il faut optimiser et surtout celles qui n'ont pas été exécutées. On en déduit de nouveaux tests qui exécuterons ces parties. D'autre part, on peut aussi utiliser le résultat pour vérifier l'adéquation du jeu de test avec le programme testé.

FIABILITE DU LOGICIEL

FIABILITE DU LOGICIEL

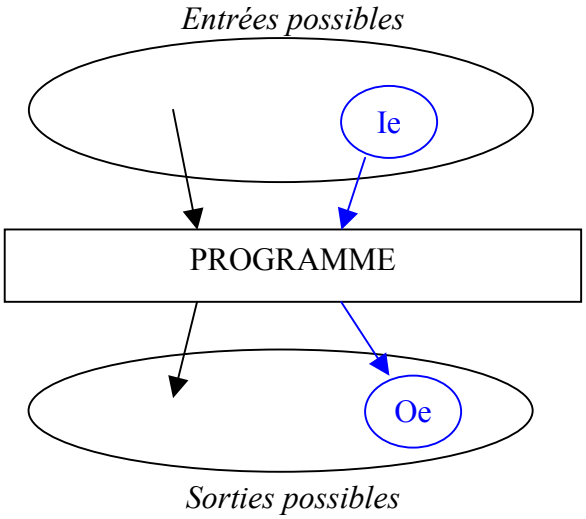
GENIE LOGICIEL – CNAM BORDEAUX 1999-2000

FIABILITE DU LOGICIEL

C'est la probabilité de faire une opération sans panne sur une durée fixée et pour un contexte donné. La fiabilité est subjective : elle dépend de l'utilisateur et du contexte d'utilisation. Elle donne une mesure du degré de confiance et elle mesure les conséquences d'une faute.

I. DEFAUT & FAUTE

Un défaut est due à la présence d'une faute. Il a une caractéristique essentiellement dynamique. Une faute est une caractéristique statique du logiciel qui provoque un défaut à l'exécution. *Exemple : pour un log. de saisie, une faute serait de ne pas vérifier la mauvaise saisie. Un défaut serait que le logiciel plante suite à la mauvaise saisie.*



Il est clair que toute faute ne provoque pas nécessairement un défaut, c'est possible si et seulement si la donnée est prise dans la partie fautive.

II. AMELIORATION DE LA FIAIBILITE

Est-ce que la fiabilité est fonction de la correction de faute logicielle ?

Non !!! Mais il faut quand même corriger les fautes, surtout les fautes graves.

Paradoxe : « Plus on augmente la fiabilité, plus on réduit l'efficacité ». Pour assurer la fiabilité, on fait des test, on rajoute du code (redondance, vérification…). Ceci entraîne le fait que le logiciel sera plus lourd, plus lent donc moins efficace. En général, on privilégie la fiabilité car l'efficacité devient de moins en moins nécessaire vu les prix des machines actuelles. Cela est plus facile à améliorer.

III. METRIQUE DE LA FIABILITE

3.1) Probabilité d'une panne

C'est la probabilité que le système se comporte de manière non prévue (non souhaitée) lorsqu'une requête est effectuée.

Exemple : Système non stop : P.F.=0,001 ⇒ sur 1000 requête, on a une proba. d'1 défaut.

3.2) Taux de panne

C'est la fréquence d'apparition d'un défaut.

Exemple : Système d'exploitation ou transactionnel…

T.F.=0.02 ⇒ sur 100 unités de temps, on a 2 défauts.

3.3) Temps moyen entre deux pannes

C'est la mesure de temps entre deux apparitions de défauts.

Exemple : Réseau (essentiellement échange de gros fichiers)…

T.M.P.=500 ⇒ le temps moyen entre deux défauts est de 500 unité de temps.

3.4) Disponibilité

C'est la probabilité que le système soit opérationnel. Elle prend en compte le temps de réparation éventuel.

Exemple : Centrale nucléaire, commande de refroidissement du noyau. Deux métriques principales : disponibilité et probabilité. On peut avoir aussi les transmission par un réseau concernant le temps moyen de panne, systèmes de communication…

Dispo = 0,998 ⇒ sur 1000 unités de temps, le système est disponible et utilisable pendant 998 unités de temps.

Unité de temps :

Elle dépend du système utilisé…

- Horloge interne pour le système « non-stop »
- Temps calendaire pour le système activité régulière
- Nombre de transaction pour le système fonctionnant à la demande.

IV. CLASSIFICATION DE DEFAUT

Classe de panne	**Description**
Transitoire	Ne se produit qu'avec certaines entrées.
Permanente	Se produit avec toutes les entrées.
Réparable	Ne nécessite pas d'intervention humaine.
Irréparable	Nécessite une intervention de l'opérateur.
Non corruptrice	Ne détruit, ni corrompt les données.
Corruptrice	Corrompt les données. (INACCEPTABLE !!!)

Exercice : Distributeurs automatique de billets.

- Chaque distributeur est utilisé 300 fois par jour,
- Une banque possède 1000 distributeurs,
- La vie d'une version de distributeur est de deux ans,
- Chaque distributeur traite environ 10 000 transactions par jour.

Classer les pannes et proposer les métriques qui vont avec.

Classe du défaut	**Exemple**	**Métrique**
Permanente et non corruptrice	Le système n'est plus opérationnel quelque soit la carte	Dispo : 1 par 1000 jours
Transitoire et non corruptrice	Les données sur la bande magnétique ne peuvent être lue sur certaines cartes (non endommagées)	Taux de panne
Transitoire et corruptrice	Le montant n'est pas correctement reporté sur le compte.	Inqualifiable (ne devrait jamais arriver).

GESTION DE PROJET

GESTION DE PROJET

GENIE LOGICIEL – CNAM BORDEAUX 1999-2000

GESTION DE PROJET

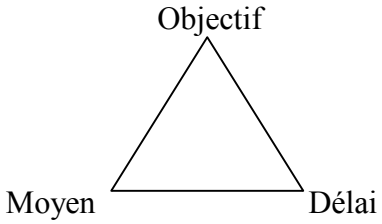
I. RAPPELS

"Qu'est-ce qu'un projet ?"

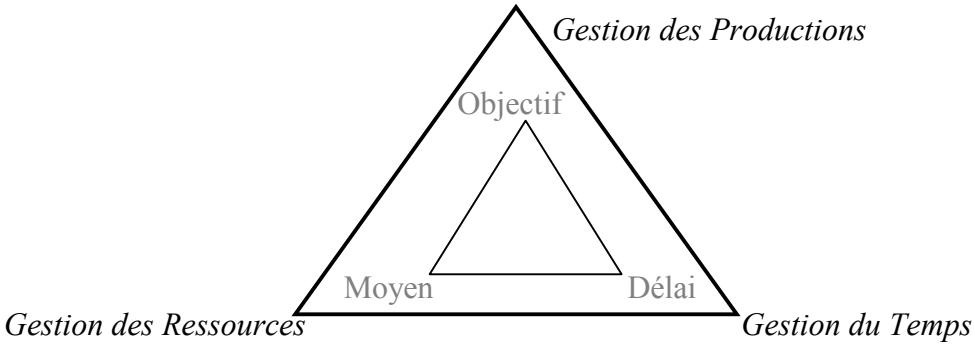
C'est une intention, plus ou moins floue, dont la réalisation est (peut-être) lointaine.

1.1) Définitions

Du point de vue scientifique: l'image d'un futur, qu'on espère atteindre.
Du point de vue du génie logiciel, c'est à triangle contraint:



La gestion du projet logiciel a pour but de le mener à son terme, en tenant compte de contraintes qui lient chacun des aspects du triangle projet.



La gestion de projet logiciel s'intéresse aux activités qui assurent que le projet commandé sera livré dans les temps en accord avec les contraintes des organismes commanditaires et réalisateurs. Il se dégage donc quatre activités principales: la structuration, l'estimation, la planification, et le suivi.

1.2) Définitions des types de Gestion

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

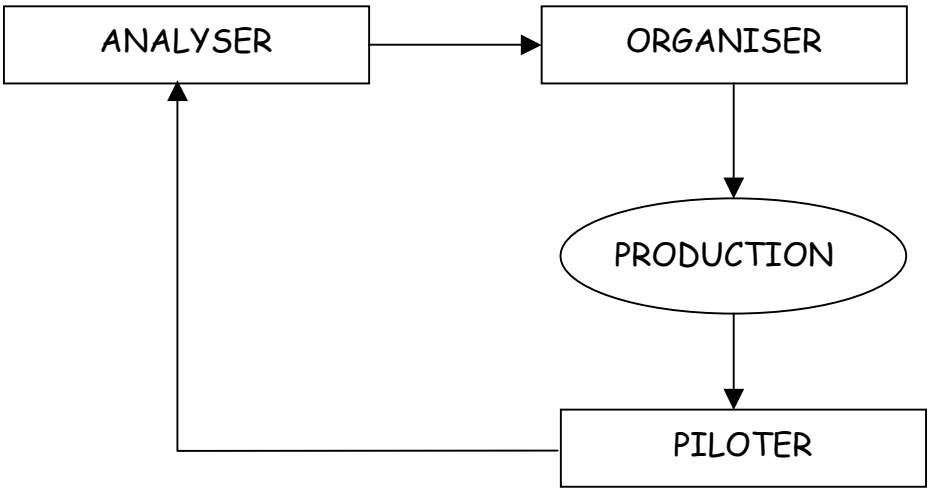
Gestion de ressources: le moyen constitué du budget du projet donc il s'agit de transformer le budget en travail, locaux, matériels, déplacements dans ce but.

Gestion des productions: l'objectif d'un projet doit à son terme être concrétisée par une ou plusieurs fournitures. Il faut s'assurer que ce qui est produit se rapproche du but final.

Remarque: la solidarité entre les sommets du triangle de gestion doit être permanente.

1.3) Activités de Gestion

"A chaque stade du développement ou étape de la production."



II. ESTIMATION DE CHARGE

2.1) Définitions

Définition: c'est la quantité de travail qu'une personne peut réaliser.

Unité: en jour / homme, mois / homme, année / homme.

Remarques: mois / homme (charge sur un mois): en général 20 jours.

Taille du projet: la taille du projet se mesure à sa charge.

Ordre de grandeur: *selon les normes ISO*:

Charge < 6 M/h	⇒ très petit projet
6 M/h ≤ charge ≤ 12 M/h	⇒ petit projet
12 M/h ≤ charge ≤ 30 M/h	⇒ projet moyen
30 M/h ≤ charge ≤ 100 M/h	⇒ grand projet
100 M/h ≤ charge	⇒ très grand projet

Durée: dépend de la charge et du nombre de personnes infectées.

Exemple: 60 M/h peut être 1 personne pendant 5 ans ou
10 personnes pendant 6 mois ou
60 personnes pendant 1 mois.

2.2) Différentes méthodes d'estimation de charge

1) La non méthode

Exemple: répondre à une offre avec un prix bas pour être sur de l'avoir, mais sans être sûr d'y gagner quelque chose en définitive (au point de vue financier).

2) Méthode Delphi *"Basée sur l'expérience des experts du domaine."*

Principe:

- Chaque expert propose une estimation basée sur son expérience.
- On publie le résultat (anonyme).
- Les experts sont invités à modifier ou à maintenir leurs estimations.
- On publie les résultats nominaux.
- Les experts refont la troisième étape.
- On analyse les disparités, on calcule la moyenne.

3) Méthode de répartitions proportionnelle

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

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

4) Méthode COCOMO

"Proposée par B.W. Boehm en 1981 (Construct Cost Model)"

En fonction des hypothèses:

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

Il propose une méthode basée sur la corrélation entre la taille d'un projet et sa charge.

Formule:

$$\text{Charge} = a \,.\, (K\ isl)^b$$

$$\text{Délai} = c \,.\, (\text{Charge})^d$$

$$\text{Taille moyenne d'équipe} = \text{Charge} / \text{Délai}$$

Avec: $K\ isl$ nombre de milliers de lignes sources.
Et les paramètres a, b, c et d qui dépendent de la catégorie du projet.

Classification:

Projet simple: < 50 000 lignes
Projet moyen: 50 000 ≤ lignes ≤ 300 000
Projet complexe: > 300 000 lignes

Type de projet	**Charge en M/h**	**Délai en M**
Simple	a = 3.2 b = 1.05	c = 2.5 d = 0.38
Moyen	a = 3 b = 1.12	c = 2.5 d = 0.35
Complexe	a = 2.8 b = 1.2	c = 2.5 d = 0.32

Facteurs: *pris en compte pour calculer la "charge nette".*

Fiabilité, complexité, taille de la mémoire, temps d'exécution...
Tous ces paramètres dépendent de l'entreprise dans laquelle est développé le logiciel.

Charge net = Produit (facteurs) x Charge (brute)

EXERCICES

Exercice 1: Estimer la taille moyenne de l'équipe qui faudrait prévoir pour développer un logiciel estimé à environ 40 000 instructions sources.

Nous appliquons la méthode COCOMO et nous nous apercevons que c'est un projet simple. Nous avons donc pour le calcul de la charge et et du délai, les coefficients suivant:
a = 3.2 et b = 1.05 c = 2.5 et d = 0.38

Donc selon la formule de la charge: **charge = 3.2 $(40)^{1.05}$ ≈ 154 M/h**
délai = 2.5 (154)0.38 ≈17 Mois

Ce qui nous donne: **Taille équipe = charge / délai = 154/17 = 9 personnes**.

Exercice 2: Sachant que la phase d'observation représente environ un tiers de la charge de l'étude préalable, calculer la charge du projet en M/h et sa répartition dans le cycle de développement. Nous supposons que la charge de la phase d'observation a été estimé à 7,5 J/h.

Charge étude préalable = 3 x phase observation ≈ 22 J/H
Charge totale = 10 x charge étude préalable ≈ 22x10 ≈ 220 J/h
≈ 11 M/h

III. PLANIFICATION DE PROJET

3.1) Définition

À partir des résultats de la structuration et de l'estimation, la planification consiste à:

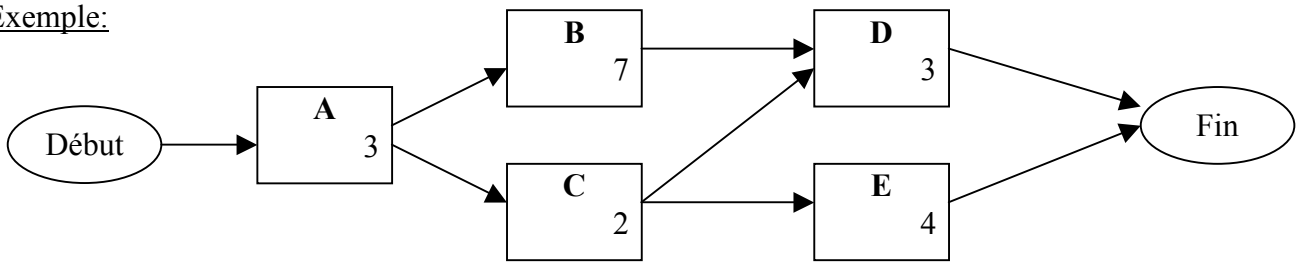
- Constater les deux listes différentes tâches et leur durée,
- Déterminer les relations de dépendance entre les tâches,
- Déterminer les étages critiques,
- Ordonnance ces les tâches dans le temps,
- Proposer à profil partage,
- De tenir compte d'éventuelles intolérables.

Pour cela, le chef de projet a deux principales techniques (complémentaires) à sa disposition.

3.2) Réseau PERT *(Profit Evaluation and Review Technique)*

Elle est basée sur les contraintes d'enchaînement avec pour chaque tâche les dates de début et de fin. C'est un graphe acyclique (orientée et sans cycle) qui permet de représenter l'enchaînement de tâche. Chaque noeud du graphe est un couple (Ti, di).

Exemple:



3.2.1) Les types de liens

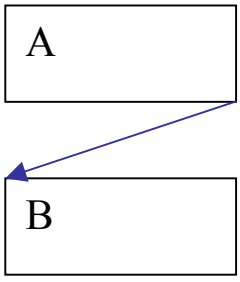
Il existe quatre types de liens pour l'enchaînement des tâches.

a) Fin → début:

C'est une relation de type "Fin début" car des que l'étape A est finie, l'étape B commence.

I délai (en jour)

Exemple:



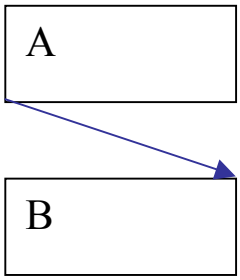
A : programmation
délai : -15 jours
B : tests

Les tests peuvent commencer quinze jours avant la fin de la programmation.

b) Début → Fin:

La tâche B ne peut se terminer que quand A commence.

Exemple:

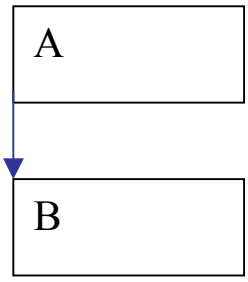


A : Gestion d'une version d'un système.
délai : +30 jours
B : Arrêt de la gestion de l'ancienne version.

On arrête la gestion de l'ancienne version que 15 jours après le début de la nouvelle version (exemple: le temps de former le personnel).

c) Début → début:

La tâche B doit commencer en même temps que la tâche A.



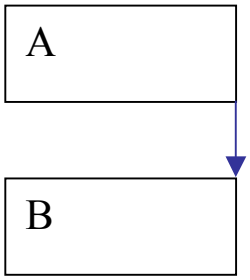
+/- délai

L'étape B doit commencer en même temps que l'étape A.

d) Fin → Fin:

La tâche B doit se finir en même temps que la tâche A.

Exemple:



A : stage

B : encadrement

L'activité d'encadrement ne se termine qu'à la fin du stage.

3.2.2) Paramètres Clés

a) Définition:

Pour déterminer le temps de fin de projet, on utilise des *paramètres clés* (associés à chaque tâche) qui sont les dates au plus tôt (D_tôt et F_tôt) et les dates au plus tard (D_tard et F_tard) ainsi que la marge qui en découle logiquement.

b) Calcul des paramètres:

N.B. Valables pour les liens de type Fin → Début.

- Dates au plus tôt:

Si la tâche Ti est en début du projet (t_o)

Alors D_tôt (T_i) = t_o
F_tôt (T_i) = D_tôt (T_i) + d_i

Sinon D_tôt (T_i) = max {F_tôt (prédécesseur (T_i))}
F_tôt (T_i) = D_tôt (T_i) + d_i

- Dates au plus tard:

De même si Ti est en fin de projet (t_f)

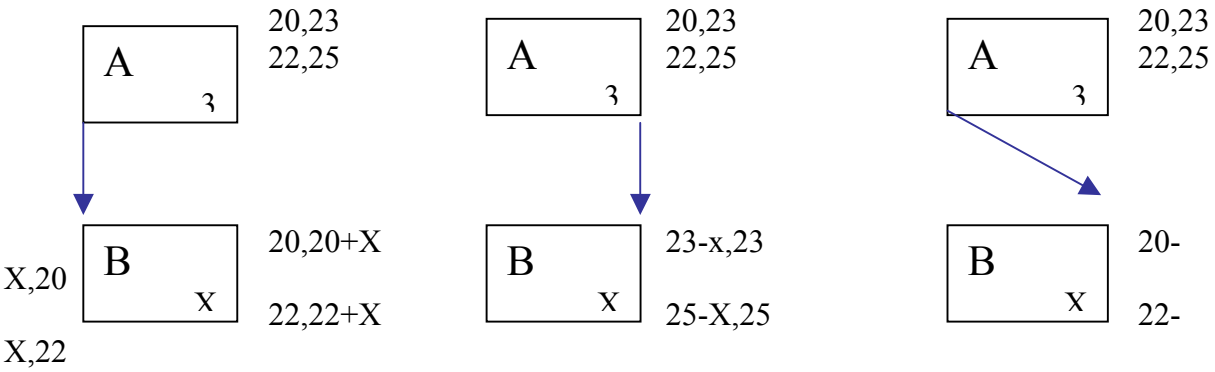
Alors F_tard (T_i) = t_f
D_tard (T_i) = F_tard (T_i) - d_i

Sinon F_tard (T_i) = min { D_tard (successeur (T_i))}
D_tard (T_i) = F_tard (T_i) - d_i

- Marges: c'est la "latitude" dont on dispose pour le temps de réalisation d'une tâche. Elle s'obtient en faisant la différence entre le temps au plus tard et le temps au plus tôt.

(D_tard – D_tôt ; F_tard - F_tôt)

N.B. Pour les autres types de liens:



- Chemin critique: c'est le chemin du graphe ayant les plus petites marges (ou marge nulle au minimum).
- Remarque: la marge ne doit jamais être négative!

Si l'on trouve une marge nulle, alors il faut:

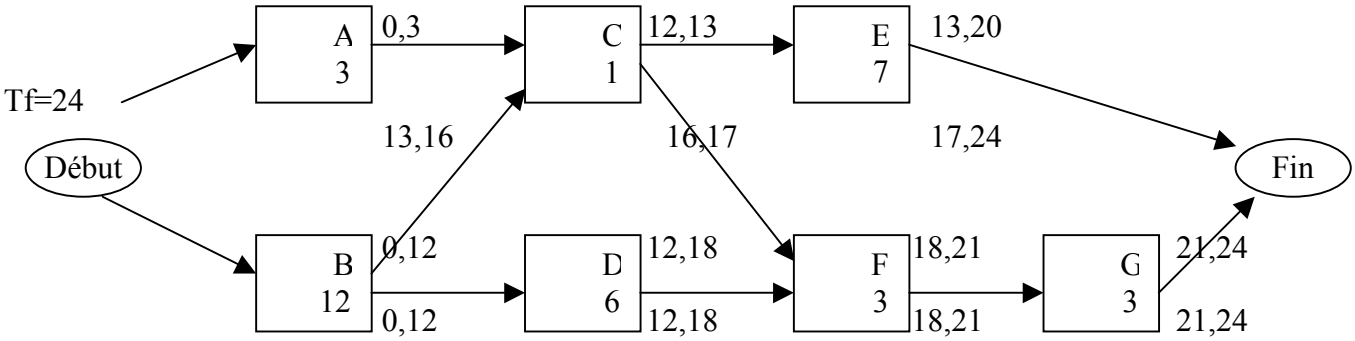
- Décomposer certaines tâches pour le parallélisme,
- Lever certaines contraintes,
- Modifier la date de fin.

EXERCICE : CHEMINS CRITIQUES.

Après découpage du projet, on obtient les contraintes suivantes:

(A,3) → C (B,12) → C,D (C,1) → E,F (D,6) → E
(E,7) (F,3) → G (G,3)

1) Construire le graphe associé.

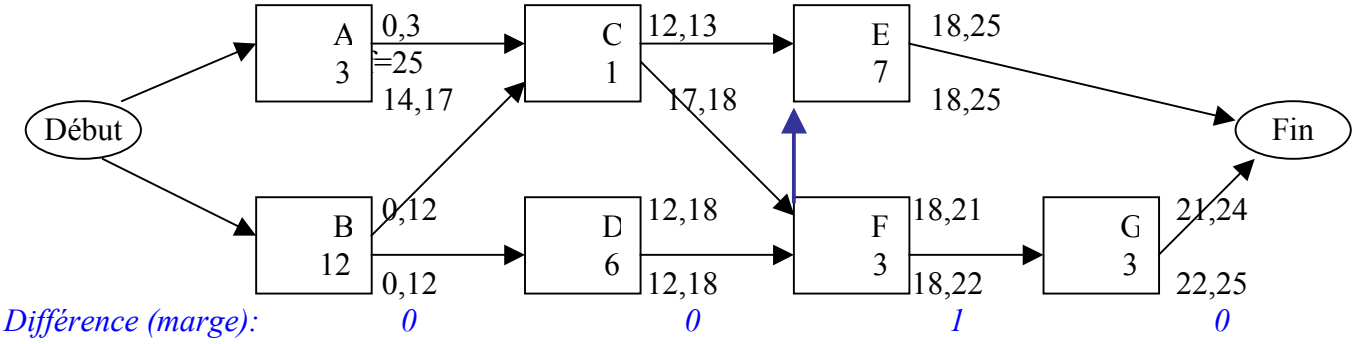


2) Déterminer le *chemin critique* (la plus petite marge).

Ici le chemin B, D, F, G donne 24 jours (avec des marges respective de 0, 0, 0 et 0).

3) Supposons qu'on ajoute une nouvelle dépendance entre F et E (de type début vers début). Que devient le chemin critique?

La nouvelle dépendance induit donc ce nouveau graphe:

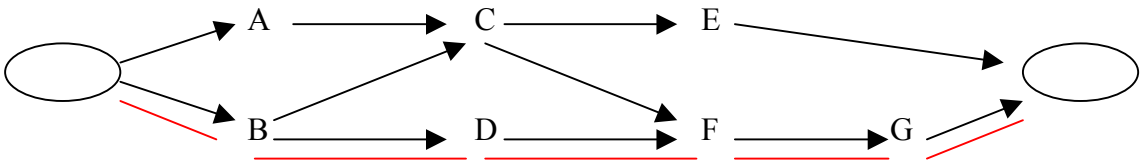


E et F doivent commencer en même temps (donc départ à 18 pour tous les deux).
Le *chemin critique* est toujours B,D,F,G mais avec un *temps de fin* de 25 jours.

3.3) Diagramme de GANTT

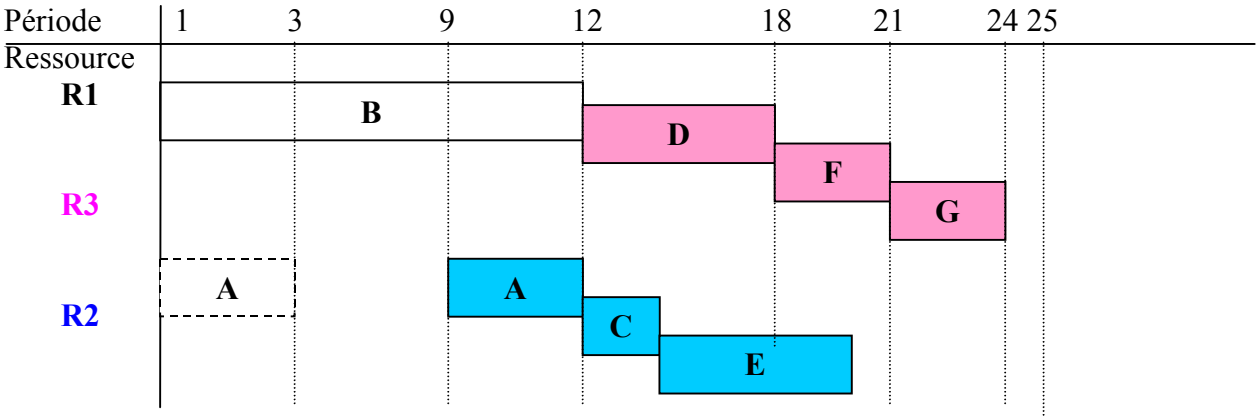
À partir de résultats obtenus du réseau PERT, plus les hypothèses sur la ressource disponible, on construit un planning (calendrier) sous forme de diagramme dont l'axe des abscisses représente le temps et l'axe des ordonnées représente les tâches.

Exemple:



Remarques :

- Selon qu'on charge le diagramme suivant le temps au plus tôt ou le temps au plus tard: on obtient un diagramme au plus tôt ou au plus tard.
- On commence toujours par charger le chemin critique.



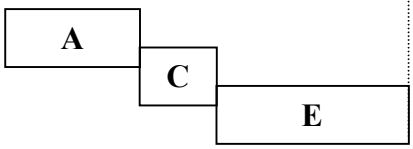
La tâche A peut être décalée pour ne pas avoir à attendre avant d'enchaîner sur les tâches C et E.

a) Supposons que l'on a 2 ressources R1 et R2.

Diagramme au plus tard: *on commence par la fin et l'on remonte.*

R1 *idem*

R2



b) S'il y a un fort déséquilibre sur les charges, on peut proposer un autre calendrier en ajoutant une troisième ressource R3.

NB : Il faut qu'une tâche soit déjà une charge d'au moins une semaine pour apparaître ici.

3.4) TD PLANIFICATION

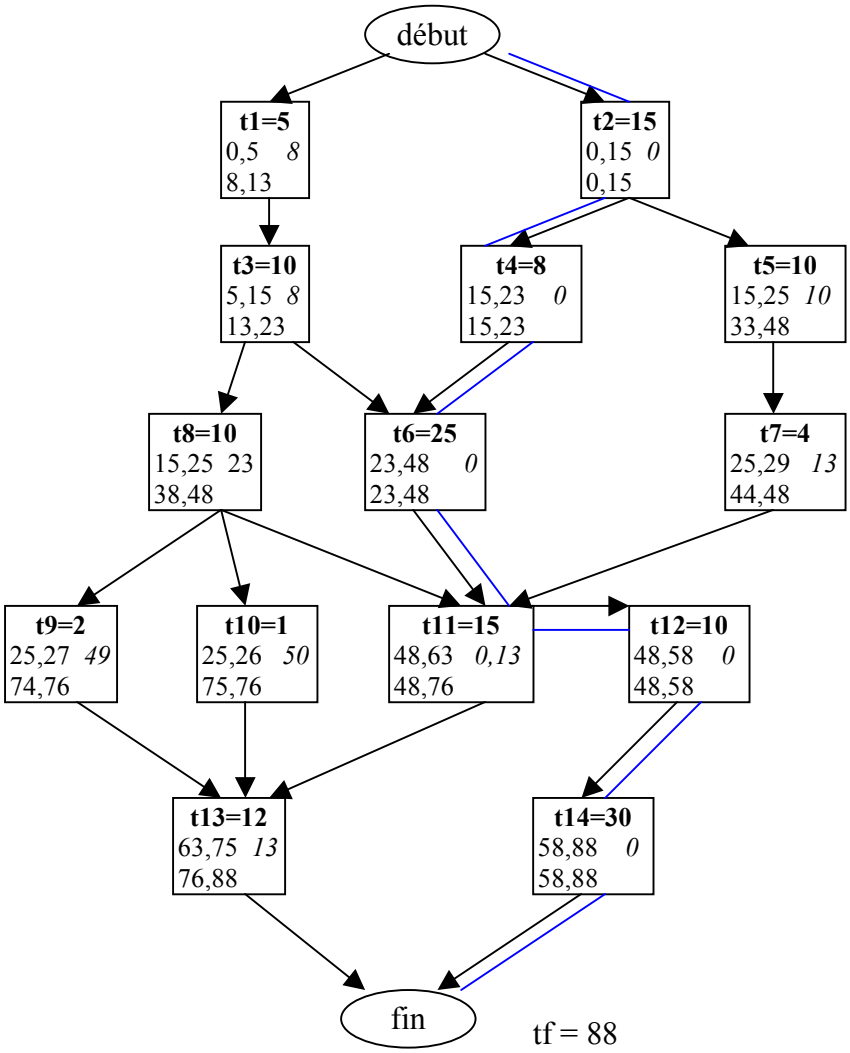
TECHNIQUES PERT ET GRANTT :

Soit le projet représenté dans le tableau suivant:

Tâche	Durée en semaine	Lien
t1	5	fin t1 - début t3
t2	15	fin t2 - début t4, t5
t3	10	fin t3 - début t6, t8
t4	8	fin t4- début t6
t5	10	fin t5- début t7
t6	25	fin t6- début t11
t7	4	fin t7- début t11
t8	10	fin t8- début t9, t10, t11
t9	2	fin t9 - début t13
t10	1	fin t10 - début t13
t11	15	début t11 – début t12 fin t11 - début t13
t12	10	fin t12 - début t14
t13	12	fin t13 - fin
t14	30	fin t14 - fin

Table 1: Enoncé de l'exercice Pert et Grantt

1. Calculer les paramètres clés et faites-les figurer sur le réseau Pert.
2. Pour chaque tâche, indiquez les dates de début au plus-tôt, de début au plus-tard, de fin au plus-tôt et de fin au plus-tard, ainsi que les marges.



3. Déterminer le chemin critique. Que devient celui-ci avec l'ajout de la contrainte suivante : « t9 ne peut pas commencer avant t=80 » ?

Soit le chemin critique. Si t9 ne peut pas commencer avant t=80, alors le chemin critique change et passe par t13 (tf=94).

4. Proposer deux plannings correspondant au Pert sans date imposée, d'abord en chargeant au plus tôt, ensuite au plus tard. Vous les représenterez sur un diagramme Grantt.

- Au plus tôt : on charge d'abord le chemin critique que l'on affecte à R1 soit t1, t3, t6, t12, t14. Ensuite pour R2, on répond au début et on place le plus tôt possible les autres tâches.
- Au plus tard : on commence par affecter R1 jusqu'à son départ

5. Sachant qu'on dispose des ressources Rl,R2 et R3, ayant les contraintes suivantes:
   - R1 est absent entre les périodes 26 et 50
   - R2 ne peut pas commencer avant la période 8
   - R3 travaille à mi-temps (50 %).

Etablissez un diagramme Grantt.

On commence par affecter R1 jusqu'à son départ. La suite du chemin critique passe à R2, à qui on confie d'abord t4. A R3, on affecte les tâches dont les marges sont le double de la durée. Comme t11 pose problème, on va la partager entre R3 et R1 qui prend la suite à son retour.

IV. PILOTAGE DE PROJET

Suivre un projet, c'est un ensemble d'actions à faire :

1. Mesurer l'état d'avancement,
2. Mesurer ce qui a été consommé,
3. Comparer les écarts entre les réalisations et les prévisionnels,
4. Expliquer les écarts,
5. Proposer les actions correctives.

Tableau de bord :

Les informations codifiés sont à deux niveaux : le suivi individuel et le suivi du projet.

4.1) Suivi individuel :

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

Nom	Tâche	Temps estimé	Temps passé	Reste à faire
xxx	Réalisation module 1 Représentation syndicale Réunion de coordination	10	3 1 ½	6
xxx	Programmation de test Maladie Congé	8	4 1 2	5

Récapitulatif mensuel :

Temps passé : T
Reste à faire : R
Avancement : A = calculé comme la différence entre le R(n-1) et R(n).

Mois	Tâche	Semaine 1			Semaine 2			Semaine 3			…	Total du mois		
xxx	A(12)	T 4	R 8	A 4	T 5	R 3	A 5	T 1	R 0	A 3		T 10	R 0	A 12
	B(10)							3	7	3				
xxx	C(12)													

Coefficient du temps passé sur le projet :

$$\frac{T(n) \times 100}{\text{nbre de jour (n)}}$$

Coefficient de productivité :

$$\frac{A(n) \times 100}{\text{nbre de jour (n)}}$$

4.2) Suivi du projet

Cela regroupe les informations globales sur le projet qui servent de base à un point d'avancement périodique. Ici, on s'intéresse aux tâches (en dehors des intervenants) même celles qui n'ont pas encore commencé.

	mois précédent			*mois courant*						
Tâche	**Mois (n-1)**			**Mois (n)**			**Evolution de charge**	**Total Temps passé**	**Evolution globale**	**Avancement**
	T	R	A	T	R	A				
A										
B										
C										
…										

Evolution de charge restant :

$$T(n) - A(n) \Leftrightarrow \boxed{T(n) - R(n-1) + R(n)}$$

Evolution globale :

$$\boxed{\sum T(n) + R(n) - \text{Charge initiale}}$$

MAINTENANCE DE LOGICIEL

MAINTENANCE DE LOGICIEL

GENIE LOGICIEL – CNAM BORDEAUX 1999-2000

MAINTENANCE DE LOGICIEL

Objectif de la maintenance :

- Gérer un processus de modification pour éviter que des corrections partielles soient faites en dehors du processus d'itérations.
- Fidéliser le client.

I. TYPES DE MAINTENANCE

1.1) Maintenance perfective (évolutive)

Elle consiste à maintenir les fonctionnalités antérieures tout en ajoutant des nouvelles fonctionnalités qui modifient profondément l'architecture.

Exemple : 1) changement de OS.
2) changement de SGBD…

1.2) Maintenance adaptative

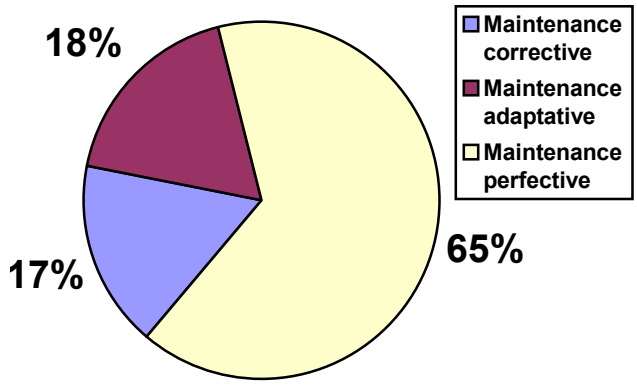
Ajout de petites fonctionnalités qui ne modifie pas l'architecture.

Exemple : 1) Mise à l'euro.
2) Passage de données par fichiers…

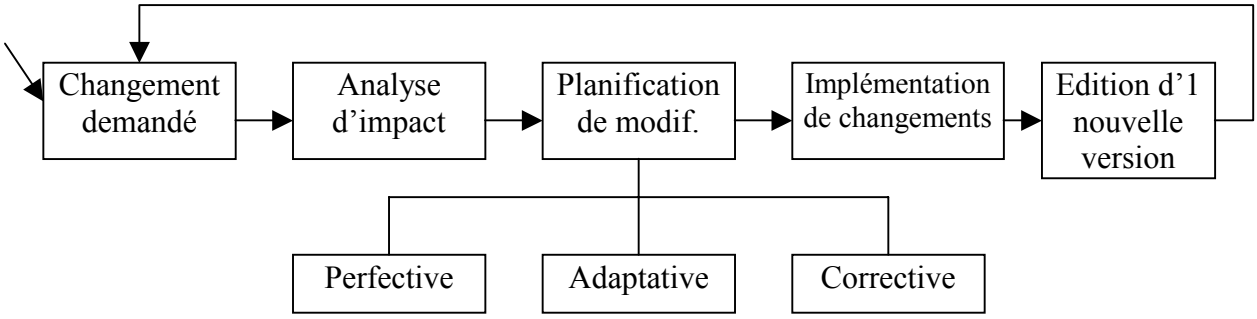
1.3) Maintenance corrective

Critère important de la qualité qui corrige les anomalies ou erreurs mises à jour par le client et non pas lors des tests de vérification et de validation.

1.4) Distribution de l'effort



II. PROCESSUS DE LA MAINTENANCE



2.1) Informations nécessaires pour la maintenance

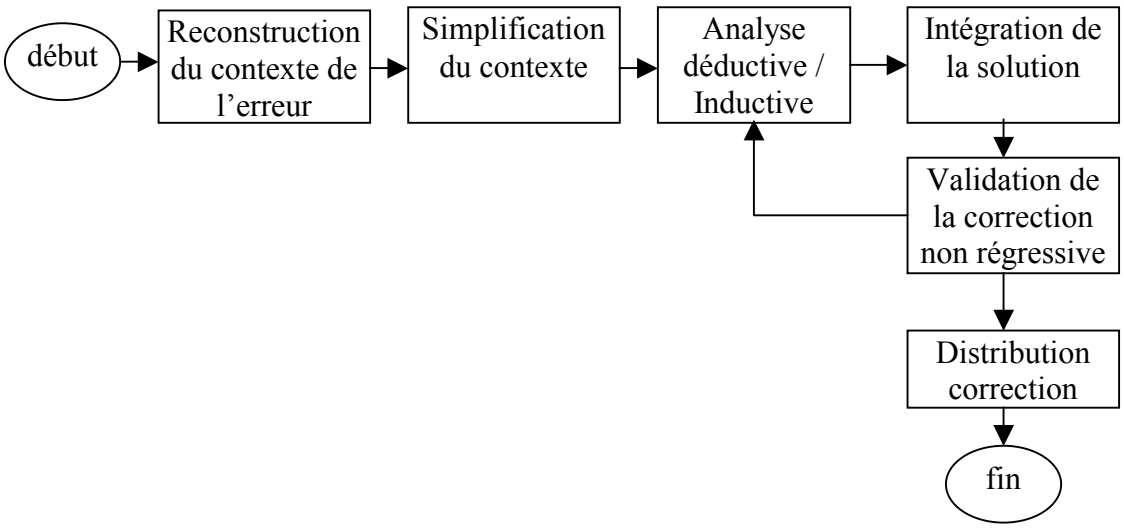
De l'équipe de développement :

- Si elle est encore en place,
- Analyse : spécifications fonctionnelles,
- Listing de sources,
- Dossier de test
- Algorithmes et les références
- Options de compilations, standard utilisés…

De l'utilisateur :

- Anomalie ou erreur (description précise), ou nouvelle fonctionnalité,
- Contexte de l'erreur ou fonction,
- Données du client,
- Environnement technique…

2.2) Cycles de développement d'une correction



2.3) EXERCICES :

1. Quelles sont les difficultés induites par l'activité de maintenance ?

Si l'équipe de développement est aussi l'équipe de maintenance, il faut revenir à un travail ancien sinon il faut comprendre la logique du système, l'âge du logiciel , la saturation de l'architecture…

2. Quels sont les facteurs qui influencent le coût de la maintenance ?

- Age du logiciel,
- Importance de la modification,
- Stabilité de l'équipe,
- Qualité de la documentation technique,
- Document de la maintenance…

III. ESTIMATION DU COUT DE LA MAINTENANCE

3.1) Formules

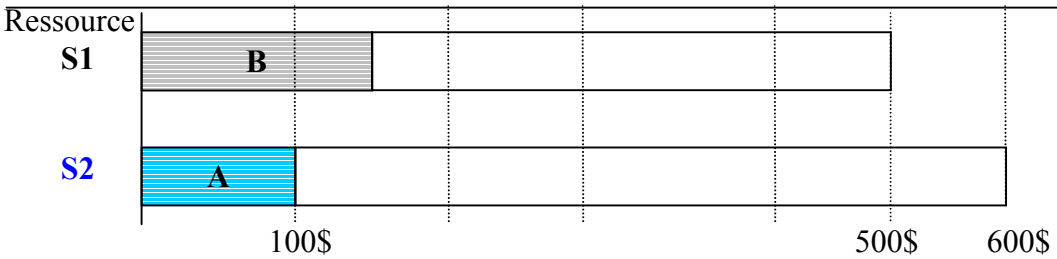
Mr Barry W. BOEHM a mesuré que les coûts annuels de maintenance sont trois à quatre fois ceux du développement d'applications nouvelles. Il proposa en 1982, une formule d'estimation du coût de la maintenance. Pour lui, le Taux Annuel de Modification (*TAM*) est basée sur le pourcentage de ligne source du logiciel à corriger dans une année.

On le déduit de l'effort de développement (qui est en fait le coût initial de développement) :

EAM = TAM x TDL

On obtient un coût de maintenance grossier (exprimé en Homme-mois). Pour obtenir le coût net, on applique des facteurs multiplicatifs. Le bénéfice sur le coût total de maintenance d'un système est proportionnel au pourcentage d'investissement sur le développement.

Exemple :



S1 = 150 000 $ 500 000 $ 50% de plus au départ
S2 = 100 000 $ 600 000 $ au final 10% de plus

3.2) Quatre facteurs multiplicatifs

Ces facteurs prennent une valeur comprise dans [0.7 ; 1.4].

- FIAB : (fiabilité) plus on vise la qualité, plus la valeur est grande
- EXPA : (expérience application)
- EXPL : (expérience logiciel) } inversement
- PROM : (langage / environnement)

Exemple :

On a un logiciel qui a coûté 236 H-M et on estime le TAM = 15%. On a fixé les facteurs multiplicateurs FIAB = 1.10, EXPA=0.91, EXPL=0.95 et PROM=0.72 (suivant le coût annuel de la maintenance). La direction décide d'utiliser une équipe moins d'expérimentée par économie d'argent (en sachant qu'un programmeur inexpérimenté vaut 7 000$ alors qu'un expert vaut quand même 9 000$). Calculer l'estimation et la différence supposée gagnée.

IV. LES EFFETS DE LA MAINTENANCE

On distingue trois catégories d'effets :

1. Effet de bord du codage.
   - Modification ou suppression d'un sous programme
   - Modification ou suppression d'un identifiant
   - Opérations logiques
   - Test de condition de sortie de boucle…

2. Effet de bord de données. Il est induit généralement par une modification d'une structure de données ou d'un champ.
   - Réduction ou augmentation de la taille d'un tableau ou structure complexe.
   - Redéfinition de format de fichier
   - Redéfinition de constante locale ou globale
   - Réinitialisation d'un pointeur…

3. Effet de bord de la documentation. Essentiellement l'adéquation entre le code source et les autres documents.

Remarque : Toute modification du code doit être reflétée dans les documents de maintenance, le manuel de l'utilisateur et le document de conception.

V. MAINTENANCE DU CODE ETRANGER

Définition :

Un code étranger est un programme (qui date de plus de 15 ans généralement) auquel aucun membre de l'équipe de maintenance n'a participé à son développement. Aucune méthodologie du génie logiciel n'a été appliqué (pas structuré, documentation pauvre et incomplète et pas de sauvegarde de modification…). Que faire dans ce cas là ? ? ?

1. Etudier le programme avant d'apporter une modification.
2. Se familiariser avec le programme en essayant de tracer un graphe de flot.
3. Evaluer l'adéquation de la documentation.
4. Insérer vos propres commentaires si vous jugez cela utile à la compréhension.
5. Ne jamais éliminer du code avant de s'assurer qu'il n'est pas utilisé ailleurs sinon avec beaucoup de précautions.
6. Indiquer absolument toute instruction que vous avez changé sur le listing.
7. Eviter de partager les variables (locales), déclarer les votre pour éviter des collisions.

VI. RE-INGENIERIE

Dans le domaine du génie logiciel, cela signifie processus qui ne consiste pas seulement à redécouvrir la conception des logiciels existants mais aussi à utiliser cette information pour reconstruire le système existant dans le but d'améliorer toutes ses qualités.

Quand décider la poursuite ou non de la maintenance?

Cela dépend du coût d'un nouveau produit (analyse) par rapport à celui de la maintenance. Si ce dernier est trop élevé, il est alors temps d'arrêter sa maintenance.

Est-il raisonnable qu'une entreprise envisage la réingénierie de tous ses logiciels?

Non, il a des logiciel qui n'auront pas à évoluer, les besoins n'évoluant pas. D'autres au contraire auront besoin d'évoluer rapidement.

VII. MAINTENANCE EVOLUTIVE

Considérons un programme « spaghettis » (prog. non structuré). Que faire pour le maintenir?

1. On peut le refaire complètement en utilisant l'atelier de génie logiciel et les principes du génie du logiciel.
2. On peut travailler dessus jusqu'à arriver, modification après modification, au changement nécessaire (en introduisant les principales de génie logiciel).
3. On peut attendre de comprendre le fonctionnement et la structure interne avant toute modification.
4. On peut refaire une conception, implémenter et tester les parties du logiciel qui exigent une modification.

Quelle est la meilleure solution?

Cela dépend de ce que l'on veut réaliser, l'importance du logiciel et de l'entreprise. A priori, la deuxième solution est la moins bonne (modification après modification).

7.1) Techniques de restructuration :

Il s'agit de produire la même fonction que le programme original mais avec une haute qualité.

Exemple : Considérons un programme P qui manipule des données A, B, C, D et accomplit les actions V, W, X, Y, Z et R définies par sa table de vérité.

A	B	C	D	V	W	X	Y	Z	R
0	0	0	0						X
0	0	0	1						X
0	0	1	0	X			X		
0	0	1	1	X			X		
0	1	0	0						X
0	1	0	1						X
0	1	1	0	X				X	
0	1	1	1	X				X	
1	0	0	0						X
1	0	0	1						X
1	0	1	0	X	X	X			
1	0	1	1	X	X	X			
1	1	0	0						X
1	1	0	1						X
1	1	1	0	X	X				
1	1	1	1	X	X				

On en déduit les relations suivantes :

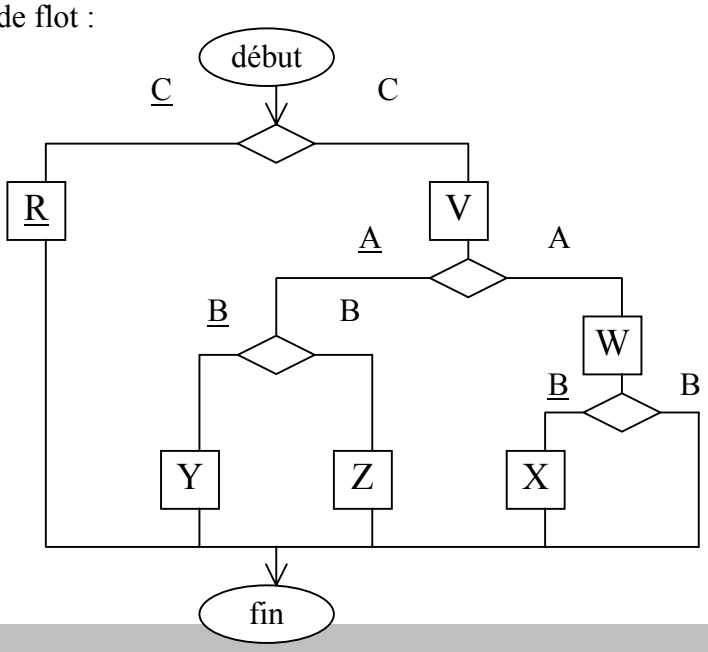
V= $\underline{A}BC\underline{D}+\underline{A}BCD+\underline{A}BC\underline{D}+\underline{A}BCD+A\underline{B}C\underline{D}+A\underline{B}CD+ABC\underline{D}+ABCD$

= $\underline{A}BC(D+\underline{D})+\underline{A}BC(D+\underline{D})+A\underline{B}C(D+\underline{D})+ABC(D+\underline{D}) = \underline{A}C(\underline{B}+B)+AC(\underline{B}+B) = (\underline{A}+A)C$

= C

W = CA ; X = CA$\underline{B}$; Y = $\underline{AB}$C ; Z = $\underline{A}$BC ; et $\underline{R} = \underline{C}$

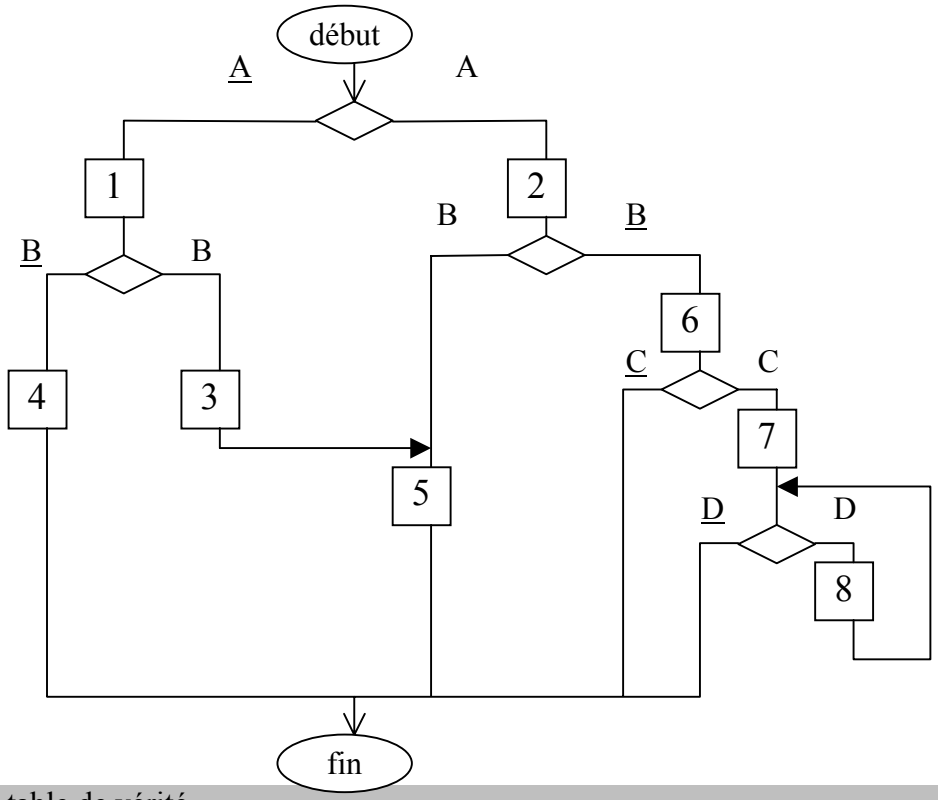
On en déduit le graphe de flot :



7.2) Exercice sur les techniques de restructuration

0.1 Simplification

Soit un programme utilisant les données A, B, C, D et réalisant les actions 1, 2, 3 4, 5, 6, 7, 8 décrit par son graphe de flot suivant:



1. Donner sa table de vérité

A	B	C	D	1	2	3	4	5	6	7	8
0	0	0	0	X			X				
0	0	0	1	X			X				
0	0	1	0	X			X				
0	0	1	1	X			X				
0	1	0	0	X		X		X			
0	1	0	1	X		X		X			
0	1	1	0	X		X		X			
0	1	1	1	X		X		X			
1	0	0	0		X				X		
1	0	0	1		X				X		
1	0	1	0		X				X	X	
1	0	1	1		X				X	X	X
1	1	0	0		X			X			
1	1	0	1		X			X			
1	1	1	0		X			X			
1	1	1	1		X			X			

2. Simplifier les actions

1 = A ; 2 = A ; 3 = $\underline{A}$B ; 4 = $\underline{AB}$; 5 = AB+$\underline{A}$B = B ;

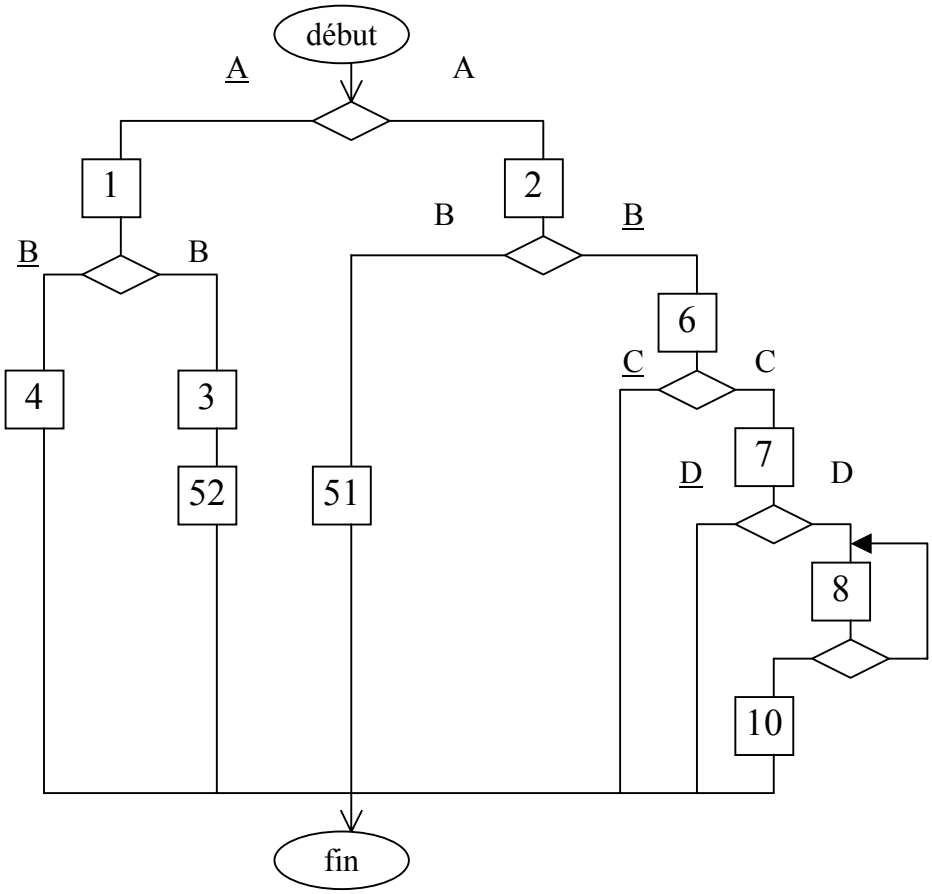
6 = A$\underline{B}$; 7 = A$\underline{B}$C ; 8 = A$\underline{B}$CD.

Pour l'action 8, D n'est en fait executé qu'une seule fois. Pour simplifier, on va considérer une action 10 = A$\underline{B}$CD.

3. Dériver un graphe simplifié

Pour simplifier le graphe, il est nécessaire de dissocier l'action en deux actions distinctes 51 = $\underline{A}$B et 52 = AB pour éviter les regroupements.

On en déduit le graphe de flot :



0.2 Traduction

1. Trouver la spécification du programme (écrit en pseudo FORTRAN) suivant :

```
Start:    Get (Time-on, Time-off, Time, Setting, Temp, Switch)
          if Switch off goto off
          if Switch on goto on
          goto Cntrld
off:      if Heating-status on goto Sw-off
          goto loop
on:       if Heating-status off goto Sw-on
          goto loop
Cntrld:   if Time = Time-on goto on
          if Time = Time-off goto off
          if Time < Time-on goto Start
          if Time> Time-off goto Start
          if Temp> Setting then goto off
          if Temp < Setting then goto on
Sw-off:   Heating-status := off
          goto Switch
Sw-on:    Heating-status := on
Switch:   Switch-heating
loop:     goto Start
```

C'est un programme qui est un contrôleur de système de chauffage. La valeur Switch peut prendre trois valeurs : si le système est sous contrôle, alors il peut-être soit allumé, soit éteint, soit dépendant de la minuterie et du thermostat. Si le chauffage est ON, l'interrupteur du chauffage passe à OFF et inversement.

2. Le traduire en un programme équivalent structuré (en Ada on en C)

```
While (true)
{  Get (Time-on, Time-off, Time, Setting, Temp, Switch)
   Case Switch of
      When OFF if Heating-status = = ON then { Heating-status := OFF;
                                               Switch-heating;          }
      When ON if Heating-status = = OFF then { Heating-status := ON;
                                               Switch-heating;          }
      When Controled if {    Time >= Time-on and Time <= Time-off    }
                     Then {  if Temp > Setting and Heating-status = = ON
                             Then {  Heating-status := OFF;
                                     Switch-heating;          }  }
                     Else {  if Temp < Setting and Heating-status = = OFF
                             Then {  Heating-status := ON;
                                     Switch-heating;          }  }
}
```

GESTION DE LA QUALITE

GESTION DE LA QUALITE

GENIE LOGICIEL – CNAM BORDEAUX 1999-2000

GESTION DE LA QUALITE

I. DEFINITION

La gestion de la qualité est l'activité qui a pour but de donner confiance au client pour certifier que le produit livré a une certaine qualité fixée par entreprise. La notion de qualité est relative et vise à promouvoir le produit ou d'entreprise. La gestion de la qualité implique la définition de procédés, le choix de standards et procédures, et surtout le contrôle de l'équipe de développement qui doit suivre les dispositifs mis en place pour les objectifs qualité.

Remarques : La gestion s'articule autour de trois activités :

- *Assurance qualité*: concerne la définition de la manière dont l'entreprise comptait atteindre la qualité.
- *Planification qualité*: sélection de procédures et standards appropriées pour un projet bien déterminé.
- *Contrôle qualité*: implique l'observation du processus de développement pour assurer que les procédures d'assurance qualité ont été suivies.

II. NORMALISATION

La normalisation répond au souci d'interchangeabilité (ou interopérabilité). Dans le domaine du logiciel, on distingue trois niveaux:

- 1er niveau : caractéristiques,
- 2ième niveau : modèle (Merise),
- 3ième niveau : la qualité (ISO 9001).

a) Classe ISO : 9001 (concerne toute la vie du logiciel), 9002 (ne concerne pas la conception), 9003 (ne concerne que la mise en service et la maintenance).
b) Classe ISO 9004 : concerne le contrôle qualité (audit).

En fait, elle définit les principes de base pour mettre en œuvre le contrôle qualité les principaux concepts : politique qualité, gestion qualité, assurance qualité, contrôle qualité.

III. MANUEL QUALITE.

Le manuel qualité est le document qui contient le système mis en œuvre pour assurer la qualité. C'est un engagement de la direction. On distingue deux types d'informations:

- Informations techniques: standards, procédures...
- Informations méthodologiques: méthodes de spécification, conception, développement.

La certification et valable trois ans en France (délivré par l'AFNOR). Cet organisme définit la qualité ainsi: « c'est l'ensemble des propriétés et caractéristiques d'un produit ou d'un service qui lui confère l'aptitude à satisfaire des besoins exprimés ou implicite ».