Ministère de l'Enseignement Supérieur et de la Recherche

Scientifique

Ecole Normale Supérieure de Constantine

ASSIA DJEBAR

Département d'informatique et des Sciences Exactes

Filière d'Informatique

Génie Logiciel

4eme Annee PES (BAC +5), PEP (BAC+4)

Préparé par : Dr. MAALEM SOUROUR

TABLE DES MATIERES

TABLE DES MATIERES

TABLE DES MATIERES 2

Table de Figures 9

INTRODUCTION GENERALE 10

I. INTRODUCTION 11

II. OBJECTIF DU COURS 12

III. ORGANISATION DU POLYCOPIE 12

CHAPITRE I: LA CONCEPTION SYSTEME 14

Objectif du Chapitre 1: 14

I. INTRODUCTION 15

II. LES ENJEUX DE L'APPROCHE SYSTEMIQUE 15

III. NOTION DE SYSTEME 15

1. Qu'est-ce qu'un système ? 15

2. Caractéristiques d'un système 15

a. Environnement 15

b. Finalité 16

c. Valeur ajoutée 16

d. Déchets 16

e. Fonctions 16

3. Systèmes et Modèles 16

IV. TYPOLOGIE DES SYSTEMES 17

1. Commerce et finances 17

a. Systèmes de traitement de transactions 17

b. Systèmes de gestion comptable et financière 18

c. Customer Relationship Management - CRM 18

d. Service Après-Vente – SAV 18

2. Administration 18

a. Systèmes d'information exécutive 18

TABLE DES MATIERES

b. Systèmes d'aide à la décision 18

c. Systèmes de rapports de gestion 19

d. Systèmes experts 20

e. Systèmes de gestion des ressources humaines 21

f. Systèmes de gestion des connaissances 21

3. Production 21

a. Systèmes d'aide professionnelle 21

b. Systèmes de planification et de production 21

c. Systèmes de gestion de la qualité 21

d. Supplier Relationship Management – SRM 21

4. Communication 21

a. Systèmes d'information bureautique 21

V. REGIME DE FONCTIONNEMENT DES SYSTEMES 22

1. La collecte de l'information 22

2. La mémorisation de l'information 23

3. Traiter l'information 25

4. Diffuser l'information 26

VI. CONCEPTION STATIQUE D'UN SYSTEME 26

1. Spécification technique du besoin (STB) 26

2. Format type d'une STB 26

a. Décrire le produit à développer 26

b. Enumérer les besoins fondamentaux qui nécessitent la création du produit 26

c. Approfondir les besoins initialement exprimés en termes d'exigences et de contraintes techniques 27

3. Analyse structurée et modélisation technique : méthode SADT, ... 28

4. Méthode SADT 28

a. Diagramme de flot de données 28

b. Diagramme structuré 31

5. Modèle Entité Association 34

TABLE DES MATIERES

VII. CONCEPTION DYNAMIQUE D'UN SYSTEME 35
1. LE PROTOTYPAGE 36
a. Objectifs 36
b. Principes 36
c. Prototype / Maquette 36
d. Le cycle de vie d'un prototype pour l'amélioration de la compréhension 37
e. Type de prototype 37
2. LES RESEAUX DE PETRI 39
a. Grafcet 39
b. Les réseaux de Pétri 44
c. Réseau de pétri marqué 45
VIII. CONCLUSION 51
CHAPITRE II: INTRODUCTION AU GENIE LOGICIEL 53
Objectif du Chapitre 2: 53
I. INTRODUCTION 54
1. Logiciel 54
2. Caractéristiques d'un logiciel 54
II. CRISE DU LOGICIEL 54
1. Historique des faits marquants (remarquable) 54
2. Symptômes les plus caractéristiques de cette crise 56
3. Quelques sources de la crise 56
4. Crise logiciel 56
III. LE GENIE LOGICIEL 57
1. La qualité exigée d'un Logiciel 58
2. Pyramide de qualité d'un logiciel 59
a. Qualité 59
b. Plan Qualité Logiciel (PQL) 59
c. Système Qualité 60
3. Propriétés de qualité à vérifier dans un Logiciel 60

d. La qualité 60
e. La validité: 60
f. La fiabilité: 60
g. La robustesse: 60
h. L'extensibilité: 60
i. La réutilisabilité: 60
j. La compatibilité: 60
k. L'efficacité: 60
l. La portabilité: 60
m. La traçabilité: 60
n. La vérifiabilité: 61
o. L'intégrité: 61
p. La facilité : 61
4. Les principes du Génie Logiciel 61
a. Modularité 61
b. Abstraction 61
c. Réutilisabilité 61
d. Maintenance facile 61
e. Faible couplage et forte cohésion 61
f. Spécification claire 62
g. Testabilité 62
h. Traçabilité 62
i. Évolutivité 62
j. Documentation 62
IV. CONCLUSION 62
CHAPITRE III: LE CYCLE DE VIE DU LOGICIEL 63
Objectif du Chapitre 3: 63
I. INTRODUCTION 64
1. Notion de cycle de vie 64

TABLE DES MATIERES

2. Justification du cycle de vie 64

3. Les étapes du cycle de vie 65

a. Définition des besoins 65

b. Analyse des besoins 65

c. Conception du projet 66

d. Réalisation du projet 66

e. Contrôle et validation 66

f. Mise en service 67

4. La conception du logiciel 67

6. La qualité d'une conception 67

a. Cohésion 68

b. Couplage 69

7. L'implémentation 70

8. La maintenance 70

II. CONCLUSION 70

CHAPITRE IV: LES MODELES DE DEVELOPPEMENT D'UN LOGICIEL 72

Objectif du Chapitre 4: 72

I. INTRODUCTION 73

II. LE CYCLE DE VIE EN " CASCADE " (Waterfall) : 73

a. Qu'est-ce que c'est le modèle en cascade ? 73

b. Etapes du modèle Waterfall 74

c. Avantages 75

d. Inconvénients 75

III. LE CYCLE DE VIE EN " V " 75

a. Qu'est-ce que le modèle en V 75

b. Etape du modèle en V 75

c. Avantages 77

d. Inconvénients 77

IV. LE CYCLE DE VIE EN SPIRALE 77

TABLE DES MATIERES

a. Qu'est-ce que le modèle en spirale 77

b. Etape du modèle en spirale 78

c. Avantages 78

d. Inconvénients 78

V. LE MODELE PAR PROTOTYPAGE 78

a. Qu'est-ce que le modèle par prototypage 78

b. Principe du modèle par prototypage 79

c. Type de prototype 79

d. Avantages 80

e. Inconvénients 80

VI. LE MODELE PAR INCREMENT 80

a. Qu'est-ce que le modèle par incrément 80

b. Principe du modèle par incrément 80

c. Avantages du modèle par incrément 81

d. Inconvénients du modèle par incrément 81

VII. MODELE ITERATIF 81

a. Qu'est-ce que le modèle itératif 81

b. Principe du modèle itératif 82

c. Avantages 82

d. Inconvénients 82

VIII. CONCLUSION 82

CHAPITRE V: LA CONCEPTION DU LOGICIEL 84

Objectif du Chapitre 5: 84

1. LA DESCRIPTION DE LA CONCEPTION 85

2. LES PRINCIPES 86

a. Rigueur 86

b. Séparation des problèmes 86

c. Modularité 87

d. Abstraction 87

TABLE DES MATIERES

e. Anticipation du changement 88

f. Généricité 88

g. Construction incrémentale 88

3. Processus de conception de logiciel 88

4. Qualité de la conception 89

CONCLUSION GENERALE 91

Bibliographie 112

TABLE DES EQUATIONS

Table de Figures

Figure 1: Principe de GL 12
Figure 2: Un système 15
Figure 3 : Schéma des systèmes de traitement de transactions 17
Figure 4 : Schéma d'un système d'information exécutive 18
Figure 5 : Schéma des systèmes d'aide à la décision 18
Figure 6 : Schéma des systèmes de rapports de gestion 19
Figure 7: Schéma d'un SI expert 20
Figure 8: Exemple de stockages par fichiers 24
Figure 9: Exemple de stockage par base de données 24
Figure 10: Exemple de système d'un carnet de contacts 24
Figure 11: Diagramme de flot de données (DFD) 29
Figure 12: Diagramme de contexte ou (DFD0) 30
Figure 13: DFD 1 30
Figure 14: Diagramme structuré 31
Figure 15: Vérificateur d'orthographe – structure n° 1 32
Figure 16: Vérificateur d'orthographe – structure n° 2 32
Figure 17: Le cycle de vie d'un prototype pour l'amélioration de la compréhension 37
Figure 18: Relation client chef projet 58
Figure 19: Règle du CQFD : Coût Qualité Fonctionnalités Délai 59
Figure 20: Evaluer la qualité 59
Figure 21 Niveau de Cohésion 69
Figure 22 Niveau de Couplage 70
Figure 23 Modèle en cascade générique 73
Figure 24 Modèle en cascade Stage Gate 74
Figure 25 Modèle en V 75
Figure 26 Modèle en spirale 77
Figure 27 Modèle de prototypage 79
Figure 28 Modèle par incrément 81
Figure 29 Modèle itératif 82
Figure 30 La modularité 87

INTRODUCTION GENERALE

INTRODUCTION GENERALE

I. INTRODUCTION

Le GL est en forte relation avec presque tous les autres domaines de l'informatique : langages de programmation (modularité, orientation objet, parallélisme, ...), bases de données (modélisation des données, de leur dynamique, ...), informatique théorique (automates, réseaux de Petri, types abstraits, ...), etc. Comme le génie chimique utilise la chimie comme un outil pour résoudre des problèmes de production industrielle, le GL utilise l'informatique comme un *outil* pour résoudre des problèmes de traitement de l'information. Le GL est aussi en relation avec d'autres disciplines de l'ingénieur : ingénierie des systèmes et gestion de projets, sûreté et fiabilité des systèmes, etc. Les principales branches du GL couvrent :

- La conception,
- La validation/vérification,
- La gestion de projet et l'assurance qualité,
- Les aspects socio-économiques.

Dans sa partie technique, le GL présente un spectre très large depuis des approches très *formelles* (spécifications formelles, approches transformationnelles, preuves de programmes) jusqu'à des démarches absolument *empiriques*. Cette variété reflète la variété des types de systèmes à produire :

- Gros systèmes de gestion (systèmes d'information) ; le plus souvent des systèmes transactionnels construits autour d'une base de données centrale ;
- Systèmes temps-réel, qui doivent répondre à des événements dans des limites de temps prédéfinies et strictes ;
- Systèmes distribués sur un réseau de machines (distribution des données et/ou des traitements), 'nouvelles architectures' liées à Internet ;
- Systèmes embarqués et systèmes critiques, interfacés avec un système à contrôler (ex: aéronautique, centrales nucléaires, ...).

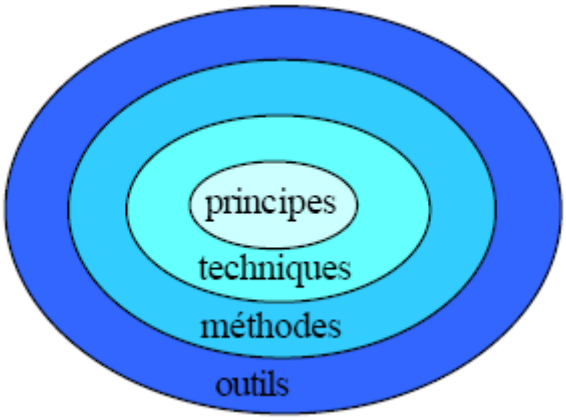


Figure 1: Principe de GL

Ce polycopié constitue un manuel de cours du Génie Logiciel et quelque étude de cas pour chaque chapitre. Il explique d'une façon simple et facile le principe et les méthodes d'analyse de conception de teste et de maintenance de logiciel.

II. OBJECTIF DU COURS

Ce module permet à l'étudiant de maitriser les concepts de base de l'analyse, conception à la réalisation d'un projet.

Ce module annuel qui est destiné aux étudiants de la 4ere année Informatique PES (Professeur d'Enseignement Secondaire) et PEM (Professeur d'Enseignement Moyen), de coefficient **3** avec un volume horaire hebdomadière de **3 heures** à raison d'une séance de cours et une séance e TD par semaine, vise en premier lieu l'introduction des concepts de base liés au Logiciel et le Génie Logiciel. Par la suite, la conception système et à la crise du LOGICIEL qui q fait naitre le Génie Logiciel sont présentés, ainsi que le cycle de vie du logiciel.

Finalement, les modèles de développement d'un logiciel sont abordés suivis par la description de la conception du logiciel qui sont nécessaires pour expliquer aux futurs Informaticien la solution type loin e toute plateforme et tout langage e programmation de la qualité et la présentation des normes et standards.

III. ORGANISATION DU POLYCOPIE

Le présent polycopié réalisé conformément au canevas destiné aux étudiants de la quatrième année PES et PEM Informatique, est réparti en cinq chapitres :

1. Le premier chapitre présente la conception système. Avec un sens large de la notion système.

2. Le deuxième chapitre comporte une introduction au génie logiciel. Il aborde la crie u logiciel qui a fait naitre le génie logiciel comme discipline.
3. Le troisième chapitre est consacré le cycle de vie du logiciel.
4. Le quatrième chapitre illustre les modèles de développement d'un logiciel
5. Le cinquième chapitre est réservé pour la conception du logiciel.

Quelques références bibliographiques sont données à la fin de ce manuscrit, grâce auxquelles nous avons pu élaborer le présent support.

CHAPITRE I : LA CONCEPTION SYSTEME

Objectif du Chapitre 1:

Ce chapitre a pour objectif de **fournir une compréhension globale de la conception d'un système**, en s'appuyant sur les fondements de l'**approche systémique**. Il vise à doter l'apprenant des outils nécessaires pour **analyser, modéliser et concevoir un système complexe** de manière rigoureuse et structurée.

À l'issue de ce chapitre, l'apprenant sera capable de :

- Identifier les **enjeux de l'approche systémique** et son importance dans l'analyse des systèmes complexes.
- Reconnaître les **différents types de systèmes** (ouverts, fermés, déterministes, stochastiques, etc.).
- Comprendre les **régimes de fonctionnement** (permanent, transitoire, stable, instable…).
- Distinguer et appliquer les concepts de **conception statique** (structure, composants, relations) et de **conception dynamique** (comportement, interactions, évolutions dans le temps).

Ce chapitre constitue une **base essentielle** pour aborder la modélisation, la simulation et le développement de **systèmes complexes** en ingénierie, en informatique ou dans tout domaine technique ou organisationnel.

CHAPITRE I: LA CONCEPTION SYSTEME

I. INTRODUCTION

La conception d'un système n'est pas quelque chose de simple. Elle est affaire de bonne pratique.

II. LES ENJEUX DE L'APPROCHE SYSTEMIQUE

L'objectif de l'approche systémique est de **comprendre la globalité d'une situation projet afin de pouvoir identifier des pistes de solutions à la hauteur des enjeux**. Cette approche va donc à l'encontre d'une démarche de rationalisation et de réduction de la complexité, fréquemment observée en gestion de projet.

III. NOTION DE SYSTEME

1. Qu'est-ce qu'un système ?

Un système est constitué d'un ensemble d'éléments en interaction dont chacun concourt à l'objectif commun ou finalité du système.

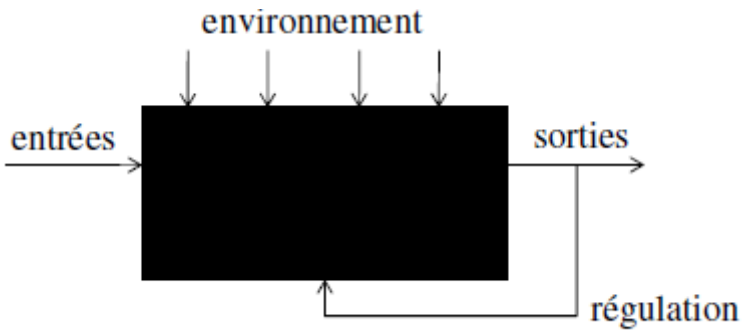


Figure 2: Un système

2. Caractéristiques d'un système

a. Environnement

L'environnement (ou milieux associés) du système est ce qui lui est extérieur lorsqu'il a été isolé par sa frontière. Le système agit sur la partie de son environnement qui est modifié, qui est la matière d'œuvre, en fonction de l'état des autres éléments extérieurs. Les relations entre le système et son environnement sont donc définies en termes de flux, c'est à dire de quantité de matière d'œuvre dans le temps.

On définit la fonction d'un système par la relation qui transforme, au niveau de la matière d'œuvre, la situation initiale en situation finale.

b. Finalité

La finalité d'un système justifie l'existence de ce dernier et ne peut être définie qu'en fonction de ce sur quoi le système agit, et qu'il transforme ou conserve, pour répondre à un besoin.
La finalité d'un système est d'apporter une valeur ajoutée à la partie de l'environnement sur lequel il agit, qui est la matière d'œuvre.

c. Valeur ajoutée

La valeur ajoutée est définie comme la modification apportée au flux de matière d'oeuvre entre l'entrée et la sortie du système.

d. Déchets

Rares sont les systèmes qui ne produisent pas de déchets (on emploiera le terme "pollution" lorsqu'ils ne sont pas traitables); et ne pas les prendre en compte risque de provoquer l'incapacité du système à assurer sa fonction.

e. Fonctions

Un système peut être étudié de deux façons : dans son aspect fonctionnel ou dans son aspect structurel.

- ***Aspect fonctionnel*** : Il s'agit de répondre à la question "à quoi ça sert ?". On parle alors de fonction d'usage. (Le citoyen qui utilise un téléviseur le voit comme un objet permettant de véhiculer des informations)
- ***Aspect structurel*** : Il s'agit de répondre à la question "comment ça marche ?". On parle alors de fonction globale. (Le technicien voit le téléviseur comme un ensemble d'éléments transformant des ondes).

Il est important de savoir si l'on observe ou conçoit un système et ses sous-systèmes dans leurs aspects fonctionnels ou structurels. En effet, un système peut être étudié (ou conçu) dans son aspect structurel alors que certains des sous-systèmes qu'il contient, communs à d'autres système, ne seront approchés que d'une manière fonctionnelle (et inversement) (le technicien peut parfaitement réparer un téléviseur sans connaître le fonctionnement du tube électronique et en respectant les branchements).

3. Systèmes et Modèles

CHAPITRE I : LA CONCEPTION SYSTEME

Un modèle est une représentation, souvent en termes mathématiques, des relations qui existent entre les variables de sortie et les variables d'entrée d'un système. Cette représentation nécessite souvent l'introduction de variables supplémentaires appelées variables d'état. A un instant donné, la valeur des variables d'état caractérise un système: leur connaissance avec celle des valeurs dans le futur des variables d'entrée permet à l'aide du modèle de déterminer les valeurs des variables de sortie dans le futur.

IV. TYPOLOGIE DES SYSTEMES

On distingue généralement deux grandes catégories de systèmes, selon les types d'application informatique :

- les systèmes de conception : fonctionnent selon des techniques temps réel ;
- les systèmes d'information de gestion, qui emploient des techniques de gestion.

1. Commerce et finances

Il existe de nombreuses classifications des systèmes pour gérer les ressources financières. Nous citons ici les systèmes de la classification la plus souvent utilisée.

a. Systèmes de traitement de transactions

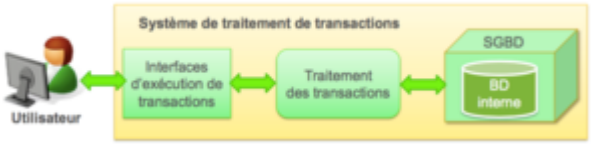


Figure 3 : Schéma des systèmes de traitement de transactions

L'objectif de ces systèmes est d'aider les entreprises dans la réalisation des opérations commerciales et logistiques.

Une transaction est une activité élémentaire exécutée durant une opération commerciale.

Exemples : une réservation des billets d'avion, une vente des produits, un achat des ressources, un inventaire des stocks, une livraison, etc.

Modes de fonctionnement :

Par lot : traite en une fois toutes les transactions accumulées durant une période prédéfinie (ex. une fois par jour)

En ligne : traite chaque transaction tout de suite après son exécution

b. Systèmes de gestion comptable et financière

c. Customer Relationship Management – CRM

d. Service Après-Vente – SAV

2. Administration

Ces systèmes sont d'une précieuse aide à l'administration et à la décision.

a. Systèmes d'information exécutive

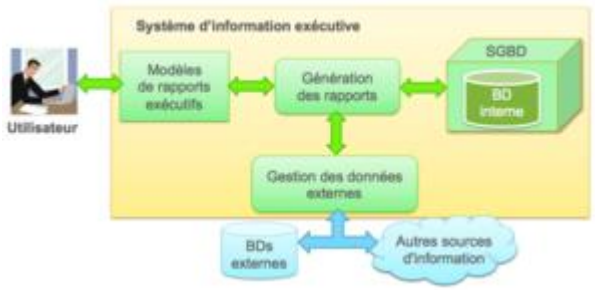


Figure 4 : Schéma d'un système d'information exécutive

L'objectif de ces systèmes est d'aider les directeur et les administrateurs de haut niveau dans leur travail exécutif en leur proposant une large variété d'informations internes et externes présentées sous forme de résumés épurés (avec un haut niveau d'abstraction).

Utilisateurs : le PDG, les directeurs, les administrateurs de haut niveau, le conseil d'administration, etc.

Information : le résumé sur la performance de la compagnie dans différents domaines.

Forme d'expression : graphique, tabulaire

b. Systèmes  d'aide à la décision

Figure 5 : Schéma des systèmes d'aide à la décision

L'objectif de ces systèmes est d'aider les administrateurs des entreprises dans les processus de prise de décisions.

Le système propose des modèles de base permettant de résoudre certains problèmes type.

Exemples : modèle de prévision des ventes, modèle de définition des prix, modèle de planification de la production, etc.

Chaque modèle comporte un ensemble de facteurs de décision et permet d'exécuter des différents scénarios en choisissant à chaque fois des facteurs différents.

Le modèle définit les indépendances entre les facteurs de décision et les conséquences possibles, et l'administrateur est demandé de faire certains jugements par la suite.

Exemple de scénario de décision

Objectif : définir le prix d'un nouveau produit.

Moyen : Système d'aide à la décision pour le marketing

Scénario d'utilisation :

L'administrateur saisit les facteurs de décision :

- le coût des ressources,
- le coût du travail,
- le coût de la promotion,
- le profit prévu durant les 5 années à venir,
- le prix d'un produit similaire,
- etc.

Le système calcule le prix du produit.

L'administrateur modifie un ou plusieurs facteurs de décision.

Le système calcule le prix du produit.

...

Au final, 'le système résume tous les résultats, et l'administrateur compare les résultats et choisit le prix.

c. Systèmes de rapports de gestion

Figure 6 : Schéma des systèmes de rapports de gestion

L'objectif de ces systèmes est d'offrir des informations aux administrateurs des entreprises sous forme de rapports de performance en fonction de leur domaine de responsabilité. Un rapport décrit une situation passée ou actuelle, mais il ne prévoit pas le futur.

Exemples : toutes les ventes réalisées l'année passée, le chiffre d'affaires classé par produit vendu, le chiffre d'affaire par client, le pourcentage de livraisons en retard, etc.

Les rapports sont générés sous forme électronique, et les données sont récupérées à partir de la base de données (enregistrées par le système de traitement de transactions) ou à partir des extraits de cette base.

d. Systèmes experts

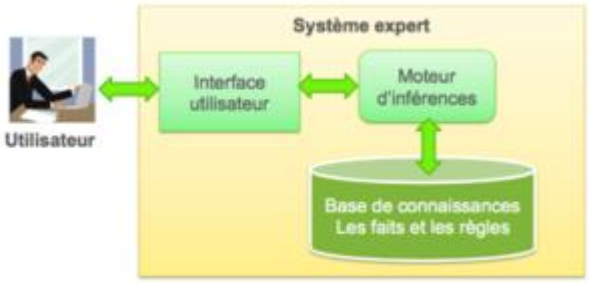


Figure 7: Schéma d'un SI expert

Les systèmes experts utilisent une intelligence artificielle pour résoudre des problèmes dans un domaine particulier qui, normalement, nécessitent des experts humains dans ce domaine précis, et qui sont généralement plus durs et coûteux à trouver.

Un système expert est un système d'aide à la décision.

Exemples : systèmes de diagnostics médicaux, systèmes de diagnostics des minéraux, systèmes d'évaluation des matières de construction, etc.

Un système expert s'appuie sur :

- une base de connaissance d'un domaine d'application très étroite : cette base contient l'ensemble des informations, en particulier des règles et des faits qui constituent le domaine de compétence du système. Elle est élaborée en collaborant avec des spécialistes du domaine.
- un moteur d'inférences permettant de réaliser des déductions logiques.

Les systèmes experts peuvent être incorporés dans tous les types de SI mais aussi utilisés comme des outils consultants. Ils sont surtout combinés avec les systèmes d'aide à la décision.

Exemple : martin praeses 2021

e. Systèmes de gestion des ressources humaines
f. Systèmes de gestion des connaissances

3. Production

a. Systèmes d'aide professionnelle

Ces systèmes sont des postes de travail étendus dont les ressources sont essentiellement conçues pour aider une catégorie de travail professionnel.

Exemple : système pour les concepteurs de voitures. Les postes de travail ont donc une résolution graphique très élevée, des outils de conception appropriés, ainsi que des outils de simulation d'accidents et de mesure de la résistance aux chocs des véhicules et la sécurité des passagers.

Autres exemples :

- systèmes d'information géographique pour les secours, pompiers, urgence, police, etc.,
- systèmes de recherche bibliographique pour les chercheurs,
- systèmes de visualisation tridimensionnelle pour les scientifiques,
- systèmes de mise en page pour les éditeurs,
- systèmes de design pour les architectes et constructeurs,
- systèmes de gestion des étudiants, des diplômes, des examens, etc.

b. Systèmes de planification et de production
c. Systèmes de gestion de la qualité
d. Supplier Relationship Management – SRM

4. Communication

a. Systèmes d'information bureautique

L'objectif de ces systèmes est de faciliter la communication entre les membres d'une organisation ainsi qu'entre l'organisation elle-même et son environnement.

Les SI bureautique aident à :

- gérer différents moyens de communication :
- les documents électroniques,
- les messageries : électronique, fax, poste, etc.
- les vidéoconférences,
- les réunions électroniques, etc.
- réaliser un objectif collaboratif :

- partager l'information dans une équipe de travail,
- éditer des documents ensemble,
- suivre le progrès d'un projet commun, etc.

V. REGIME DE FONCTIONNEMENT DES SYSTEMES

Maintenant que nous avons vu dans le détail ce qu'était un système et ses typologies, voyons plus précisément ses fonctions qui sont de collecter, stocker, traiter et diffuser l'information de son environnement.

1. La collecte de l'information

Pour toute organisation, l'information est précieuse et indispensable pour sa **pérennité.** On le verra plus en détail, mais elle permet à l'organisation de **prendre des décisions**, de **surveiller** et **piloter** son activité et même de **créer de la valeur.**

La collecte d'information du SI c'est donc recueillir l'information, puis la saisir, c'est-à-dire la faire « entrer » dans le SI. On peut dire que la collecte d'information, c'est le fait **d'enregistrer l'information afin de procéder à son traitement.** L'information ainsi recueillie va généralement être décomposée de façon structurée afin d'en faciliter le stockage et le(s) traitement(s) ultérieur(s).

Il n'en reste pas moins que l'information est coûteuse car sa saisie nécessite, en règle générale, une intervention humaine donc coûteuse pour l'entreprise.

Par exemple, une facture est un document basé sur un bon de commande. Dans le cas d'une saisie manuelle, les informations (produits, quantité, prix…) sont saisies deux fois. À la commande et à la facture. Avec un SI, les informations sont saisies une fois et en quantité limitées : les produits et la quantité.

Cette information peut avoir deux provenances distinctes :

une provenance **interne** : c'est le flux d'information qui est généré par les entités internes à l'organisation (approvisionnements, production, gestion des salariés, comptabilité, vente, etc.), par son fonctionnement (processus, méthodes), mais aussi le flux d'informations informelles (climat social, bien-être des salariés, savoir-faire, etc.) moins simples à mettre en évidence mais déterminants, etc.

une provenance **externe** : il s'agit du flux d'informations généré par des parties prenantes externes à l'entreprise (clients, fournisseurs, État...) et qui sont essentiels pour anticiper les mutations et l'adaptation du SI pour servir l'organisation. Par la mise en place de veilles (technologiques, sociétales, légales, commerciales, etc.), l'entreprise prend conscience qu'il est fondamental pour elle d'être particulièrement attentive aux informations de source externe.

Si je reprends l'exemple de Google Maps que je vous ai précédemment donné, la provenance de l'information est externe. En effet, les données cartographiques et les recherches des utilisateurs du service sont des informations ne provenant pas du SI. Le premier provient des satellites externes au SI de Google Maps, et le 2e, lui vient des utilisateurs, c'est à dire vous, lorsque vous tapez votre lieu de recherche sur le site.

2. La mémorisation de l'information

Une fois collectée et saisie, l'information doit être **stockée** de manière **durable, stable** et **sécurisée** afin de pouvoir être ultérieurement utilisée ou tout simplement pour répondre à des obligations légales.

C'est ce que fait votre opérateur téléphonique lorsque vous recevez ou passez des appels. Chaque appel est horodaté et consigné dans une base de données. Cela permet d'établir vos factures détaillées, mais aussi de transmettre vos relevés d'appels lorsque la justice l'exige.

Pour organiser le stockage de l'information, des moyens techniques et organisationnels sont mis en œuvre comme les méthodes **d'archivage,** des techniques de **sauvegarde,** de **protection** contre le piratage ou encore des méthodes pour prévenir la **destruction** de données. Comme l'information est précieuse, il est impératif que les SI sécurisent bien ces informations.

L'organisation du stockage

Les informations sont donc collectées et rangées soit dans des fichiers soit dans ce qu'on appelle une **base de données (ou BDD).**

Le **fichier** est une collection, un ensemble de données réunies sous un même nom. Techniquement c'est une information numérique constituée d'une séquence d'octets, c'est à dire d'une séquence de nombre.

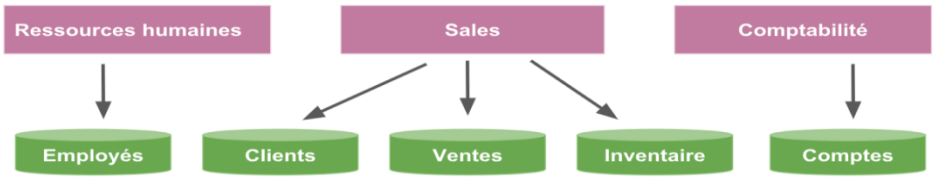


Figure 8: Exemple de stockages par fichiers

La base de données (BDD) est une structure de rangement d'informations qui prend un peu la forme de grands tableaux, comme un peu un tableau Excel. Chaque colonne sert à trier un type de données et chaque ligne représente un enregistrement.

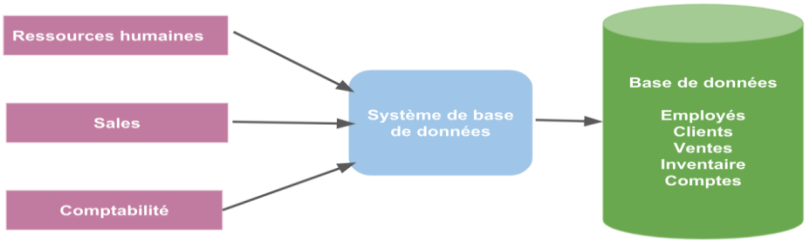


Figure 9: Exemple de stockage par base de données

Prenons l'exemple d'un carnet de contacts téléphoniques. On peut retrouver en colonne le numéro du contact, son nom, son prénom, sa fonction et son numéro de téléphone. Chaque ligne représente un enregistrement.

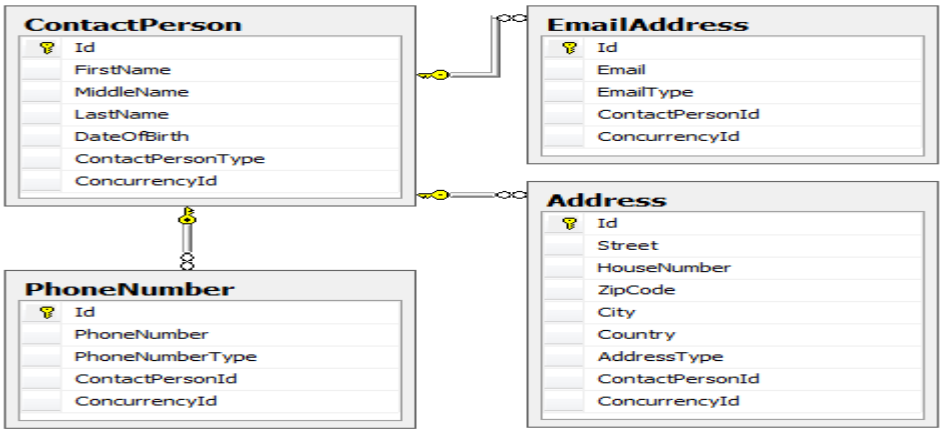


Figure 10: Exemple de système d'un carnet de contacts

Lieu de stockage

Enfin, ces fichiers et bases de données doivent être stockés. On peut les trouver : directement sur les **disques durs** des serveurs du SI de l'organisation. Dans ce contexte l'information n'est accessible que depuis ce serveur ; dans des **aires de stockages** au sein du SI de l'organisation. Il s'agit de très gros disques durs accessibles par le réseau de l'entreprise. Dans ce contexte, l'information est accessible depuis tous les composants du SI mais uniquement au sein de l'organisation ; dans le **Cloud.** Dans ce contexte, l'information est accessible de partout dans le monde. On étudiera le Cloud ensemble dans le cours.

3. Traiter l'information

Une fois que l'information a été collectée et stockée, elle est disponible pour **traitement.**

Le **traitement** de l'information consiste à produire de nouvelles informations à partir d'informations existantes grâce à des programmes informatiques ou des opérations manuelles.

Par exemple, il vous arrive d'appeler de temps à autre une hotline, lorsque vous avez un problème avec votre accès Internet. Dans la plupart des cas, on vous demande d'interagir avec le système avec des réponses basiques à des questions comme « OUI » ou « NON ». C'est un exemple de traitement. On utilise la reconnaissance vocale pour récolter vos réponses et produire ainsi de nouvelles informations.

Le traitement de l'information peut prendre 4 formes différentes. On peut :

Consulter l'information : il s'agit du traitement le plus simple puisqu'il consiste à accéder à l'information telle qu'elle a été enregistrée ;

Organiser l'information : ce traitement consiste à structurer l'information selon des critères spécifiques. Cela peut-être par exemple regrouper l'information par client, par zones géographiques, par activités et bien d'autres encore ;

Mettre à jour l'information : ce traitement va consister à reprendre une information précédemment enregistrée et à l'actualiser ;

Produire de nouvelles informations : à partir d'information(s) existante(s), ce traitement va permettre la création de nouvelles informations.

4. Diffuser l'information

Quel que soit son origine ou ce qu'elle représente, une information n'a de valeur que si elle est communiquée aux bons destinataires, au bon moment et sous une forme directement exploitable.

Prenons votre facture téléphonique. Tous les mois, vous recevez au format électronique ou par courrier, un relevé de vos communications téléphoniques ainsi que le coût de chacun de vos appels et le montant total que vous devez à votre opérateur. Cette facture est un support de diffusion de l'information.

VI. CONCEPTION STATIQUE D'UN SYSTEME

1. Spécification technique du besoin (STB)

La Spécification Technique de Besoin (STB) est un document à caractère contractuel établi par le demandeur d'un produit à l'intention du concepteur et par lequel il exprime son besoin (ou celui qu'il est chargé de traduire) en termes d'exigences techniques.

La STB doit être suffisante pour qu'un concepteur puisse élaborer une définition du produit qui y réponde sans ambiguïté.

2. Format type d'une STB

a. Décrire le produit à développer

- Présenter le concept général du produit
- Apporter les informations nécessaires pour accréditer le projet et motiver les équipes qui y participent
- Définir les phases successives de l'utilisation du produit jusqu'à son retrait de service

b. Enumérer les besoins fondamentaux qui nécessitent la création du produit

Exigences principales :

- Dans quel but le produit est il créé ?
- Sur quoi le produit agit il ?
- Qu'est ce qui pourrait le faire évoluer ? le modifier ?

Fonctions

- fonctions essentielles
- fonctions complémentaires

- décomposition fonctionnelle : toute fonction doit être décomposée pour faire apparaître chaque fonction élémentaire qui la constitue
- critères de valeurs spécifiques à chacune des fonctions : critères d'utilisation, de fiabilité, de durée de vie...
- critères généraux applicables à toutes les fonctions :
  - Aspect technique : puissance, performance, niveau d'automatisation, durée, adaptabilité...
  - Aspect opérationnel : fiabilité, sécurité...
  - Aspect commercial : délais, coût d'installation, coût de maintenance et d'exploitation, investissements...
- constituants principaux : établir le PBS

c. Approfondir les besoins initialement exprimés en termes d'exigences et de contraintes techniques

Exigences fonctionnelles

- exigences de performances : elles sont indépendantes de la solution technique adoptée
- exigences techniques : elles dépendent quant à elles de la solution technique choisie. Elles précisent les paramètres fonctionnels que les produits doivent respecter pour satisfaire aux autres exigences et contraintes que la spécification impose
- les exigences fonctionnelles doivent être définies, puis vérifiées par des contrôles en fin de fabrication, des tests, des démonstrations de compatibilité...

Exigences opérationnelles

- sûreté de fonctionnement
- durée de vie du produit

Exigences d'interfaces : mentionner les données qui influencent le montage, l'assemblage l'intégration

Contraintes d'environnement

Contraintes de conception et de réalisation :

- spécifier les contraintes dues aux options retenues par les fournisseurs, imposées à l'issue d'un processus de décision approprié, ou correspondant à des règles générales de fonctionnement
- identifier ce qui a trait à la préservation des caractéristiques du produit

Contraintes logistiques de mise en œuvre

- transport

- manutention
- stockage

3. Analyse structurée et modélisation technique : méthode SADT,...

L'**analyse structurée** est une méthode d'analyse des besoins qui permet de passer d'un cahier des charges à un ensemble de spécifications puis à un programme informatique. Elle s'accompagne de méthodes de notations comme les diagrammes de flux de contrôle (CFD), les diagrammes de flux de données (DFD), les diagrammes d'entité-relation (ERD) et les diagrammes de transition d'états (STD).

4. Méthode SADT

a. Diagramme de flot de données

L'approche fonctionnelle descendante est un moyen de dériver une architecture modulaire du logiciel à partir d'un diagramme de flots de données.

A partir d'un diagramme de flots de données on dérive la structure du programme selon une méthode décrite par Yourdon et Constantine. Le système est vu comme un ensemble de fonctions. La méthode consiste à :

-Concevoir les fonctions de haut niveau

-Affiner jusqu'à obtenir une conception suffisamment détaillée

L'exemple ci-dessous décrit comment à partir d'un diagramme de flots de données dériver un modèle d'architecture.

Exemple : Vérificateur d'orthographe :(d'après I. Somerville)

Ce programme recherche chaque mot d'un document dans un dictionnaire. Si le mot apparaît dans le dictionnaire, il n'y a pas d'autre traitement à réaliser. Sinon, le mot est affiché. Si l'utilisateur décide que le mot est correctement orthographié, il est entré dans le dictionnaire, sinon il est entré dans la liste des mots mal orthographiés.

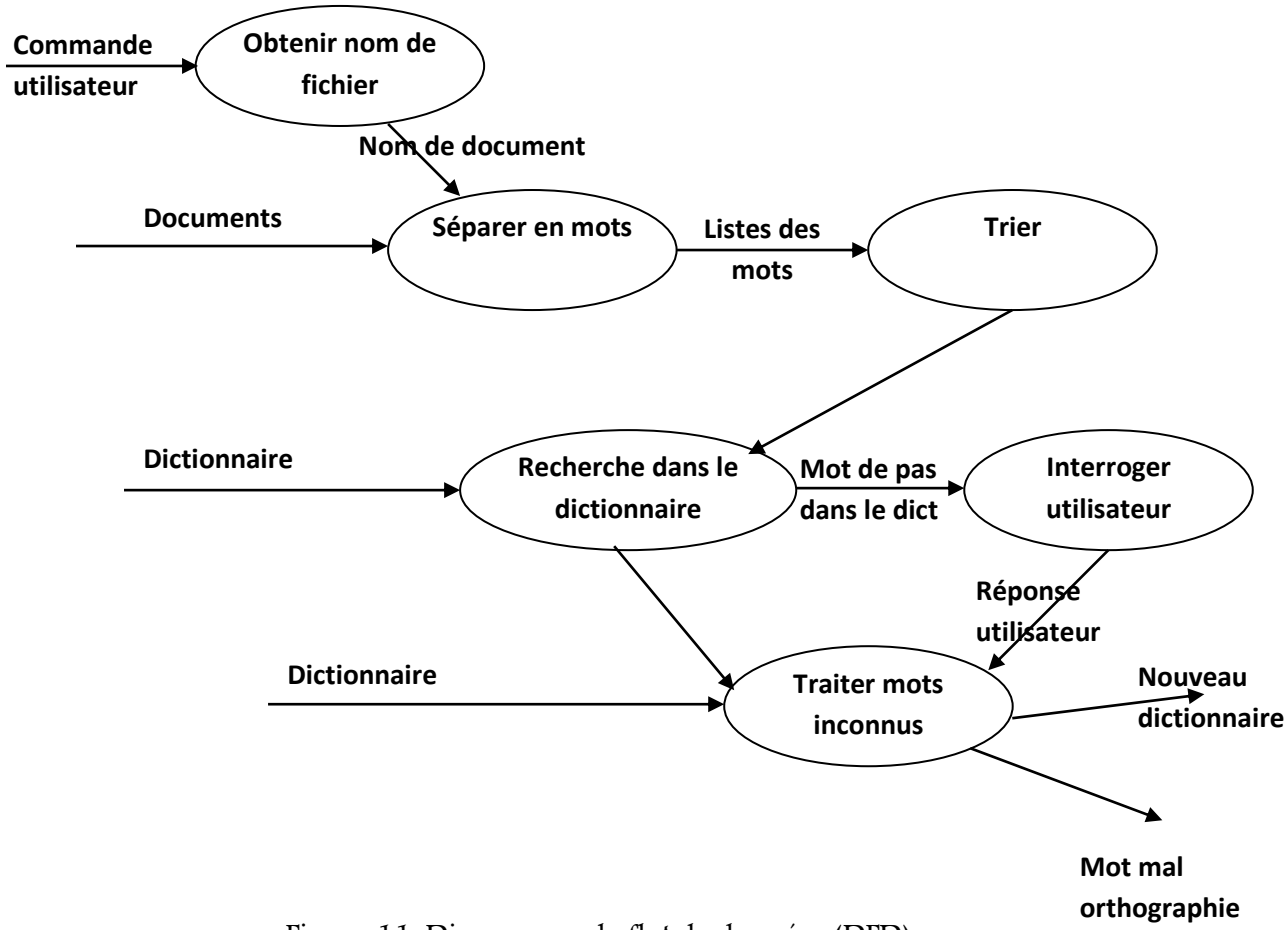


Figure 11: Diagramme de flot de données (DFD)

Définition :

Il s'agit d'une technique semi-formelle et opérationnelle. Les DFD décrivent des collections de données manipulées par des fonctions. Les données peuvent être persistantes (dans des stockages) ou circulantes (flots de données).

La représentation graphique classique distingue :

- Les *fonctions* par des cercles
- Les *stockages* par des boîtes ouvertes
- Les *flots* par des flèches
- Les *entités externes* par des rectangles

Au niveau le plus abstrait, on peut se contenter des entités (les 'acteurs') situés à l'interface (aux 'bords du système') et des flots qu'ils s'échangent, sans décomposition en fonctions. On parle alors de *diagramme de contexte*.

En faisant apparaître les fonctions et en les raffinant de plus en plus, on obtient des DFD à différents niveaux d'abstraction.

La figure suivante donne un exemple de DFD, concernant la sélection des réponses à un appel d'offre. Il s'agit du diagramme de contexte.

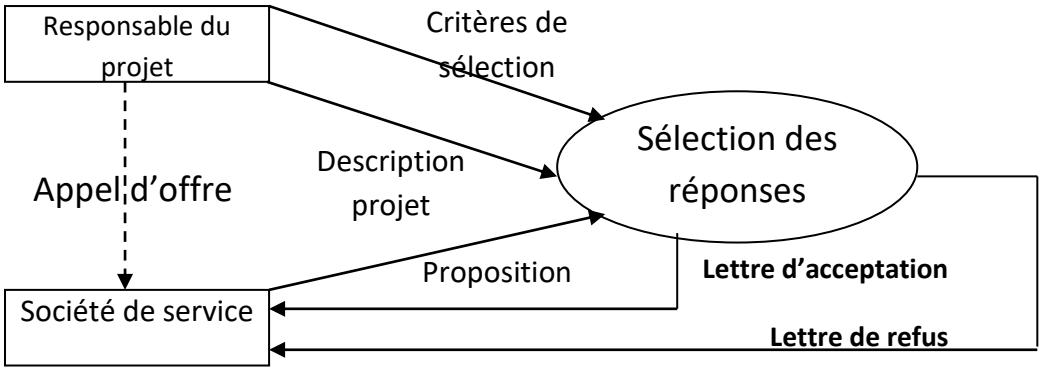


Figure 12: Diagramme de contexte ou (DFD0)

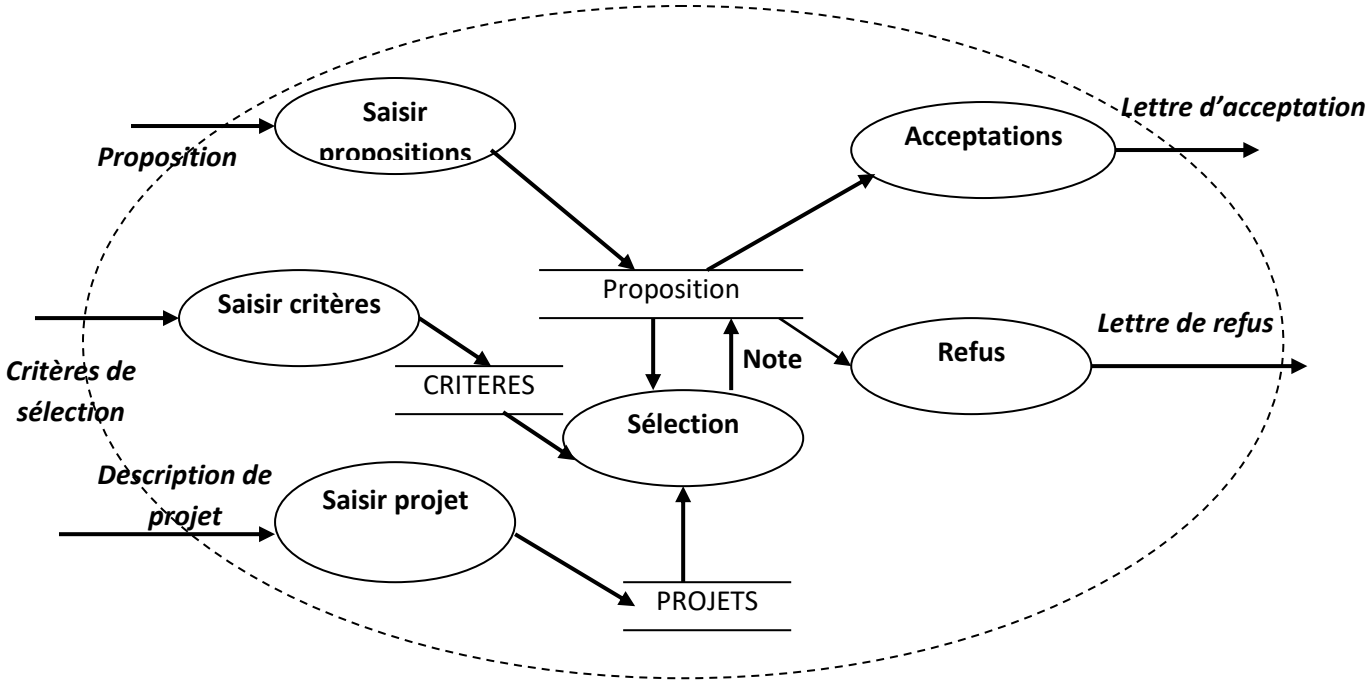


Figure 13: DFD 1

Raffinement du DFD précédent (les flèches entrantes et sortantes doivent être les mêmes)

Un niveau de raffinement doit contenir autant de diagrammes de raffinement que son niveau père contient de fonctions à raffiner (fonctions considérées comme complexes)

Les DFD sont une notation semi formelle, car :

- la sémantique des fonctions n'est pas spécifiée précisément (que signifie exactement 'Evaluation' ?),
- ni les aspects liés au contrôle (ou séquencement des opérations).

Pour ces raisons les DFD sont :

- soit complétés par d'autres spécifications,
- soit étendus :

– flots de contrôle,
– tampons,
– etc.

- Ils connaissent un très grand succès pour spécifier les fonctions d'un système à cause de leur simplicité et de leur facilité de compréhension par des non informaticiens.

b. Diagramme structuré

Un diagramme structuré constitue le graphe d'appel d'un logiciel. La méthode décrit comment les éléments d'un diagramme de flot de données peuvent être dérivés en une hiérarchie d'unités de programme (CONSTANTINE & YOURDON).

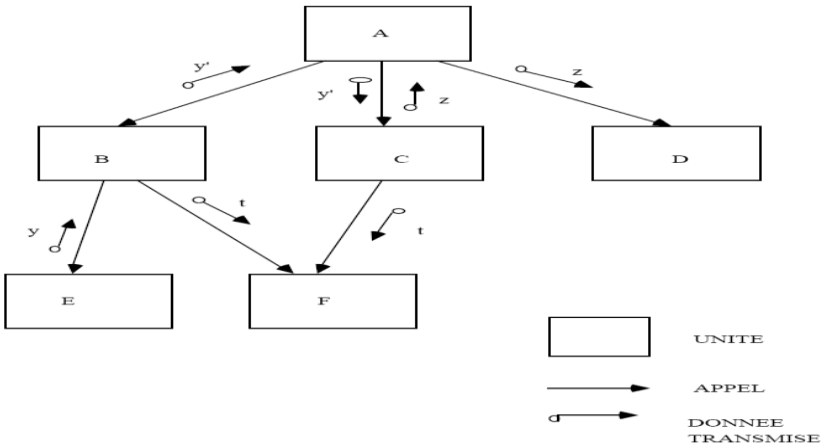


Figure 14: Diagramme structuré

À partir d'un diagramme de flot, on peut construire plusieurs diagrammes structurés.

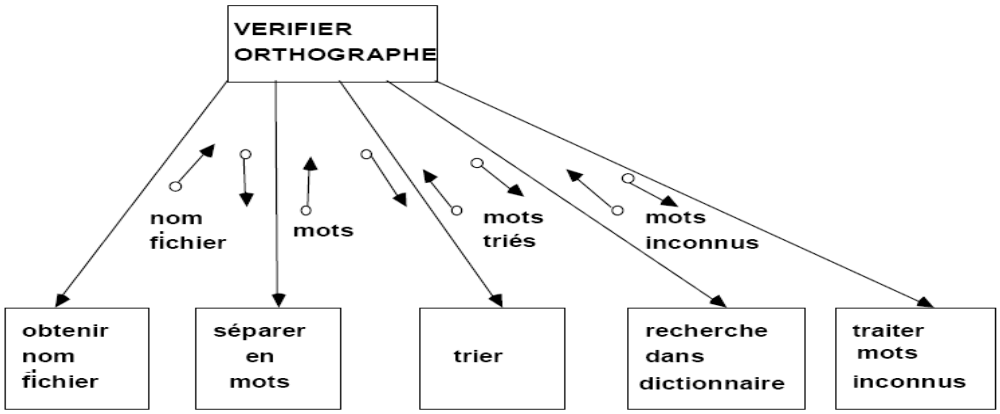


Figure 15: Vérificateur d'orthographe – structure n° 1

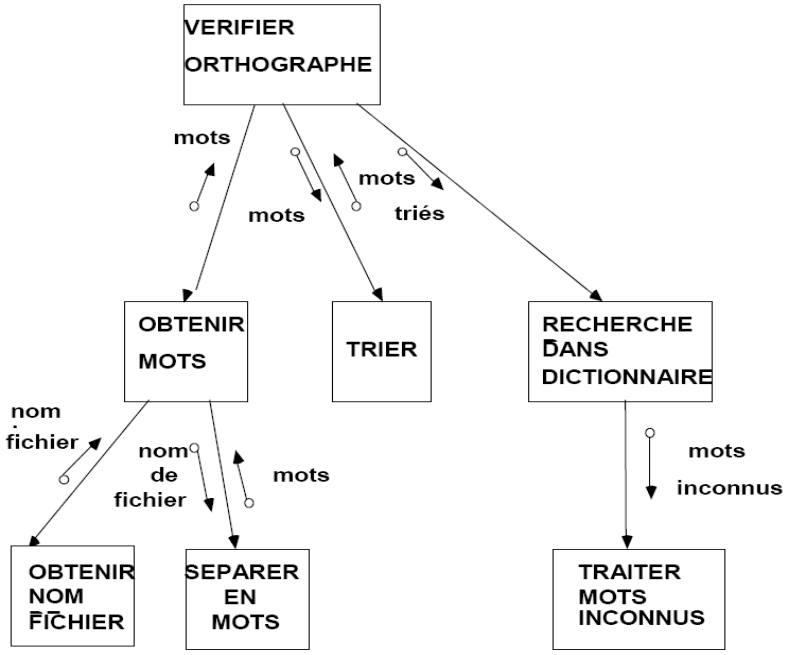


Figure 16: Vérificateur d'orthographe – structure n° 2

a) Définitions

On classe les différentes unités en :

– Unités d'entrée, dites "afférentes" : acceptent des données d'un niveau plus bas, les transmettent vers un niveau plus haut.

– Unités de sortie, dites "efférentes" : acceptent des données d'un niveau plus haut, les transmettent vers un niveau plus bas.

– Unités de transformation accepte des données d'un niveau plus haut, les transforment, les retransmettent vers un niveau plus haut.

– Unités de coordination : servent à contrôler d'autres unités.

b) Méthode de dérivation

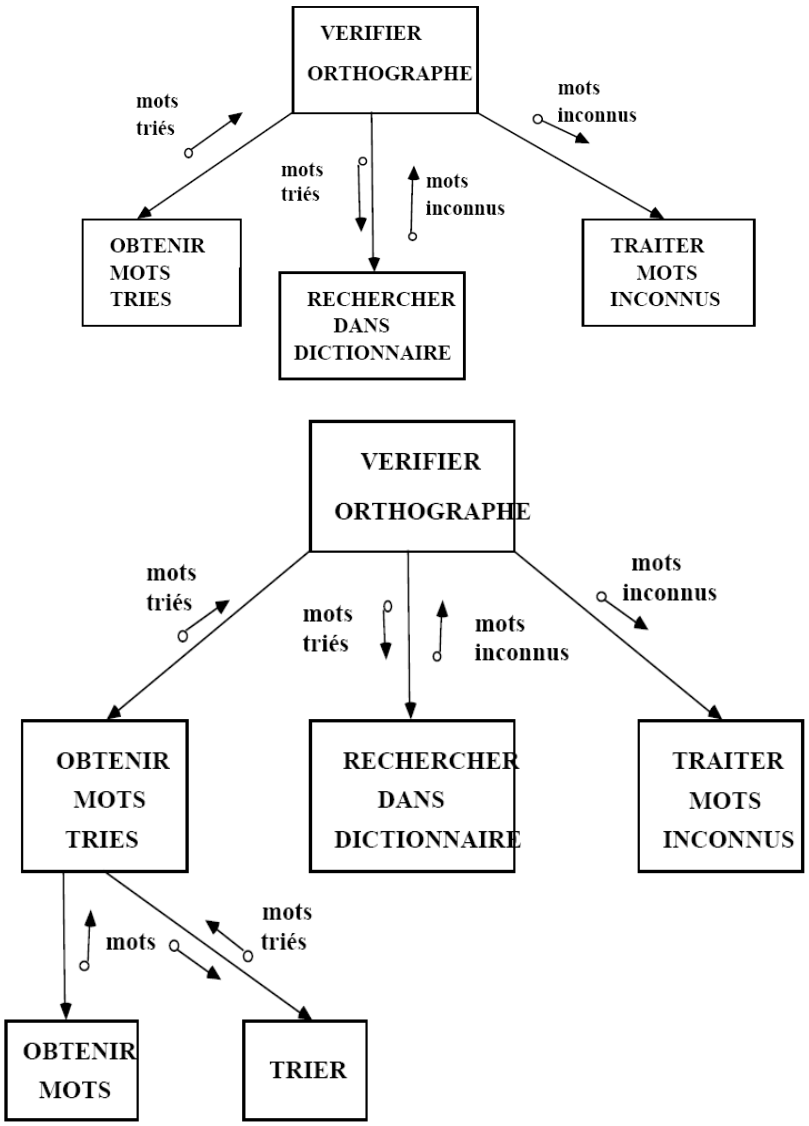
1. Identifier l'unité d'entrée de plus haut niveau : Tracer les entrées jusqu'à une bulle telle que sa sortie ne puisse être immédiatement dérivée de ses entrées. L'unité précédente est l'unité d'entrée de plus haut niveau.

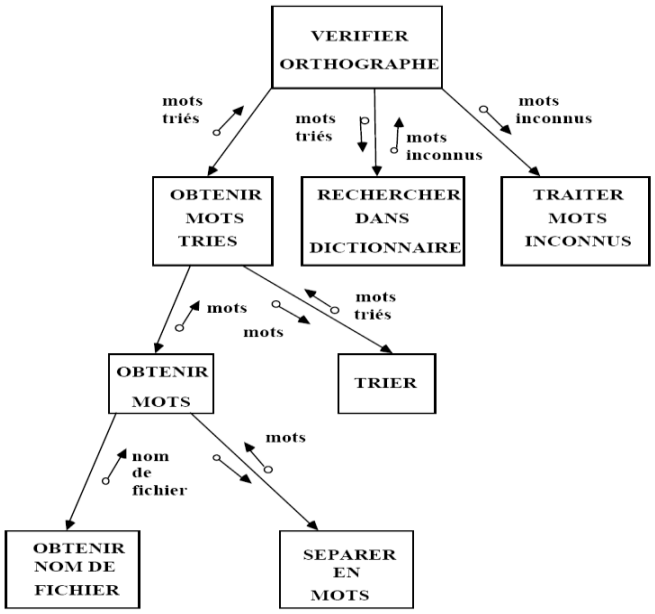
2 Identifier l'unité de sortie de plus haut niveau : Même méthode, à partir des sorties.

3 Les unités non visitées sont les unités de transformation de plus haut niveau.

4 Répéter ce processus récursivement à partir des unités de premier niveau.

Exemple : VÉRIFICATEUR D'ORTHOGRAPHE





Une fois l'architecture globale décrite, les données des différents modules doivent être précisées afin que les différents programmeurs qui coderont les différents modules soient bien d'accord sur le type et l'ordre des paramètres à échanger.

Exemple :

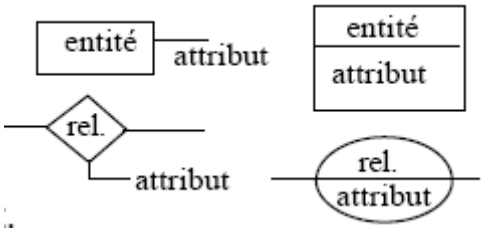
```
Procedure VERIFIER_ORTHOGRAPHE is
        Procedure OBTENIR_MOTS_TRIES
                (MOTS_TRIES :out LISTE_DE_MOTS)
        is separate ;
                Procedure RECHERCHER_DANS_DICTIONNAIRE
                (MOTS_TRIES in LISTE_DE_MOTS ;
                MOTS_INCONNUS : out LISTE_DE_MOTS)
        is separate ;
                Procedure TRAITER_MOTS_INCONNUS
                (MOTS_INCONNUS :in LISTE_DE_MOTS)
        is separate ;

MOTS_TRIES, MOTS_INCONNUS : LISTE_DE_MOTS
begin
OBTENIR (MOTS_TRIES) ;
RECHERCHER_DANS_DICTIONNAIRE (MOTS_TRIES, MOTS_INCONNUS) ;
TRAITER_MOTS_INCONNUS (MOTS_INCONNUS) ;
end VERIFIER_ORTHOGRAPHE ;
```

5. Modèle Entité Association

Il s'agit d'une technique semi formelle et déclarative (Peter Chen, 1976). Ces schémas permettent de spécifier *la structure des données et de leurs relations*, ce qui n'est fait ni dans les DFD, ni dans les modèles orientés contrôle. C'est indispensable pour les systèmes organisés autour de larges ensembles de données interconnectées.

Les concepts du modèle de base sont :

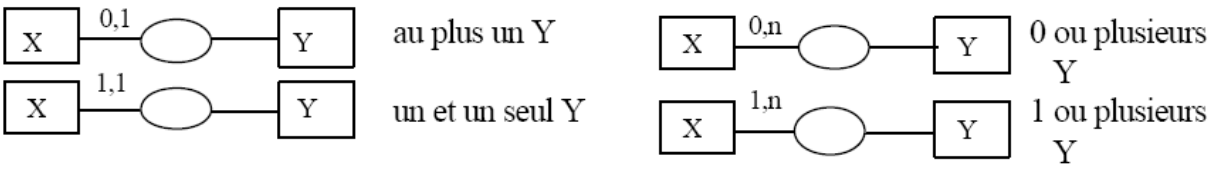
- Les *entités*, qui sont des collections d'items partageant des propriétés communes (occurrences d'entités),
- Les *associations* (ou *relations*), qui traduisent l'existence de liens entre entités (occurrences d'associations entre occurrences d'entités),
- Les *attributs* (ou *propriétés*), attachés aux entités et aux associations et qui les caractérisent.



Une entité existe indépendamment de ce qui l'entoure. Une association n'existe que si les entités extrémités existent. Chaque entité à un attribut *identifiant* qui distingue univoquement chaque occurrence d'entité.

Certains modèles, dits modèles binaires, n'autorisent que des associations entre 2 entités. Les associations peuvent être partielles. Les associations sont souvent caractérisées par leur cardinalités, qui peut être notées de diverses manières.

Avec les notations de Merise, à tout X correspond :



Le modèle EA (ou ER) n'est pas normalisé et de nombreuses notations et représentations coexistent. Certains ajoutent des relations d'agrégation et/ou d'héritage entre entités (nous les étudierons dans le cadre des modèles à objets), ainsi que des contraintes supplémentaires sur les données (ou contraintes d'intégrité).

VII. CONCEPTION DYNAMIQUE D'UN SYSTEME

1. LE PROTOTYPAGE

a. Objectifs

Eviter les Pb du modèle cascade :

- l'absence de retour d'expérience, "feed back" contrer l'effet tunnel en rendant visible l'exécution du logiciel
- le manque de flexibilité, de réactivité du processus, notamment la rigidité des spécifications,
- la longueur des délais, le niveau des coûts.

b. Principes

- Les lois qui régissent la réalisation, le comportement et l'utilisation du logiciel sont mal connues

=> Il est nécessaire d'expérimenter pour acquérir de la connaissance, mieux comprendre
=> Ne pas travailler en aveugle, mais voir ce que l'on est en train de faire

- Les utilisateurs sont seuls (?) à savoir ce que doit faire le logiciel => démarche participative

Ils ne peuvent pas valider la définition du logiciel à partir de documents,

Ils ont besoin de le voir fonctionner (cf. effet tunnel)

- A la différence d'un pont, il est possible d'accroître progressivement les fonctions assurées par un logiciel

=> Effectuer des cycles courts, spécification – conception – fabrication – installation qui s'appliquent à des sous-ensembles fonctionnels du logiciel

=> Découpage vertical, et non horizontal, de l'ensemble des tâches à réaliser.

• Utilisation de technologies apparues au début des années 80 :

- Langages de haut niveau
- Interface utilisateur graphique => Environnement de développement permettant de développer du code rapidement et à faible coût

c. Prototype / Maquette

Dans les industries manufacturières,

Maquette = Exemplaire qui diffère du produit de série par certaines caractéristiques :

- la taille, l'échelle
- la nature des composants,
- omission de certains composants ou fonctions.

Prototype = exemplaire qui ne diffère du produit de série que par le procédé de fabrication

Dans les deux cas, l'objectif est de tester, expérimenter pour améliorer la compréhension.

Un **prototype logiciel** peut différer du produit final par

- fonctions offertes (spécification)
- structure (conception)
- langage de programmation (codage)
- omission de couches : interface utilisateur, traitements, données, couplage (maquette = uniquement l'interface)
- qualités non-fonctionnelles : fiabilité, performance, confidentialité, volumes
- environnement d'exécution

d. Le cycle de vie d'un prototype pour l'amélioration de la compréhension

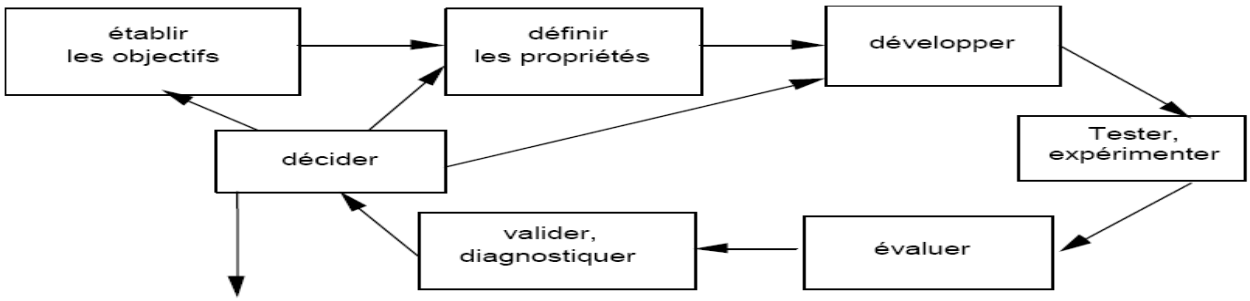


Figure 17: Le cycle de vie d'un prototype pour l'amélioration de la compréhension

Le processus est cyclique, donc + difficile à contrôler

- Quelles propriétés sélectionner ?
- Comment tester et évaluer un prototype ?
- Quel est le critère d'arrêt de la boucle ?
- Comment assurer la convergence du processus et la stabilité du logiciel ?

(Dépend des objectifs du prototype, et détermine l'organisation du processus)

e. Type de prototype

a) Prototype jetable

La programmation exploratoire, pour acquérir de la compréhension

Objectif

Améliorer la compréhension de l'un des aspects du logiciel (spécification, conception, codage, architecture technique, ...) par des expérimentations et des tests en laboratoire.

Principes

=> ne porte que sur un ou des aspects mal compris

=> sera jeté, et suivi d'un cycle de vie classique.

=> doit être bon marché et vite fait.

Exemples

Services à offrir, interface utilisateur, charge du réseau, temps de réponse, couplage avec un autre logiciel mal connu, mécanisme du langage de programmation

Limites

- L'expérimentation est biaisée :

– les conditions de test diffèrent des conditions normales d'utilisation

– le logiciel n'est pas celui qui sera livrée : les aspects non pris en cpte peuvent interférer avec ceux qui le sont

- Certains aspects sont difficiles à tester interactions avec l'environnement réel conditions d'utilisation imprévisibles
- Confondre le prototype avec un document explicitant les spécifications ou la conception du logiciel final
  - ⇨ Relations contractuelles mal définies.
- Tentation de réutiliser le prototype, dont le code est mal structuré
  - ⇨ Maintenance coûteuse.

b) Prototype évolutif

La programmation progressive, pour la validation par les utilisateurs

Objectif

Préciser les besoins réels encore mal compris, par l'utilisation d'un prototype en vraie grandeur ou sur un site pilote.

Principes

=> ne porte que sur les aspects bien compris

=> sera étendu progressivement

=> le logiciel doit être bien fait.

Limites

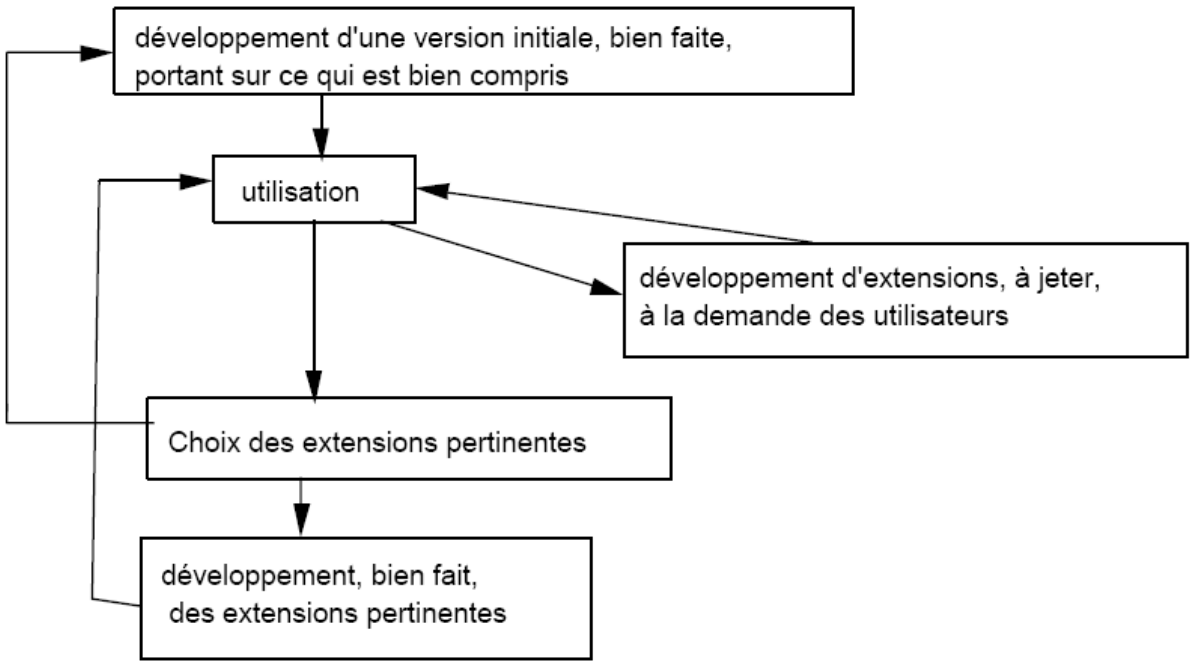
- L'expérimentation est biaisée si elle porte sur un site pilote.
- L'instabilité du logiciel peut poser problème aux utilisateurs.
- N'apporte que peu de connaissance sur les aspects mal compris.
- Difficulté d'intégrer les nouveaux aspects dans l'architecture existante

  => Dégradation de la structure du logiciel

  => Maintenance coûteuse.
- Le projet est mal défini

  => risque de dérive de la qualité, des coûts et délais

  => relations client / fournisseurs mal définies



2. LES RESEAUX DE PETRI

a. Grafcet

Le Grafcet et le réseau de Pétri sont un excellent moyen pour modéliser la dynamique des systèmes et vérifier certaines propriétés.

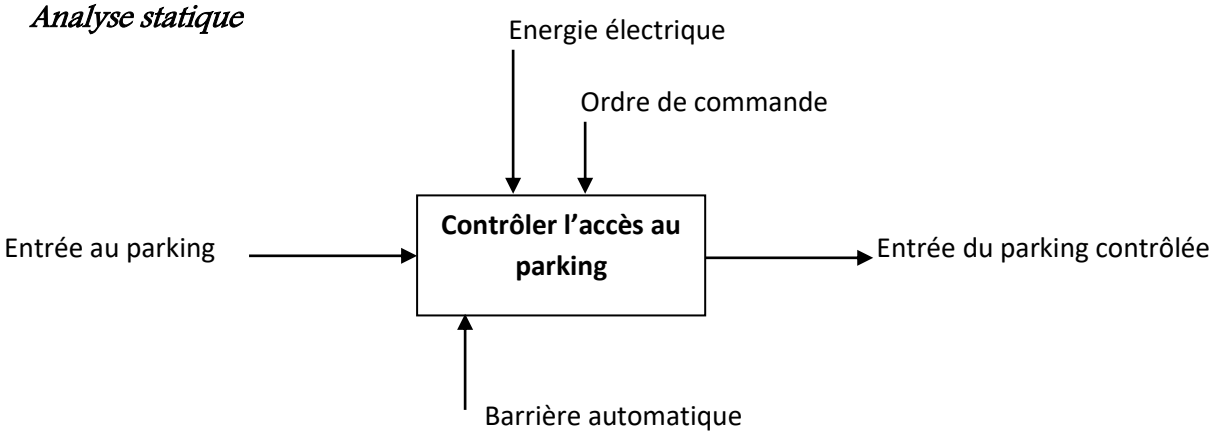
Exemple :

Réalisé une barrière automatique qui contrôle l'accès d'un parking. Seul les conducteurs possède une carte magnétique sont autorisés à commander l'ouverture, après la commande d'ouverture l'accès est possible durant 15 secondes.

Conception d'un système :

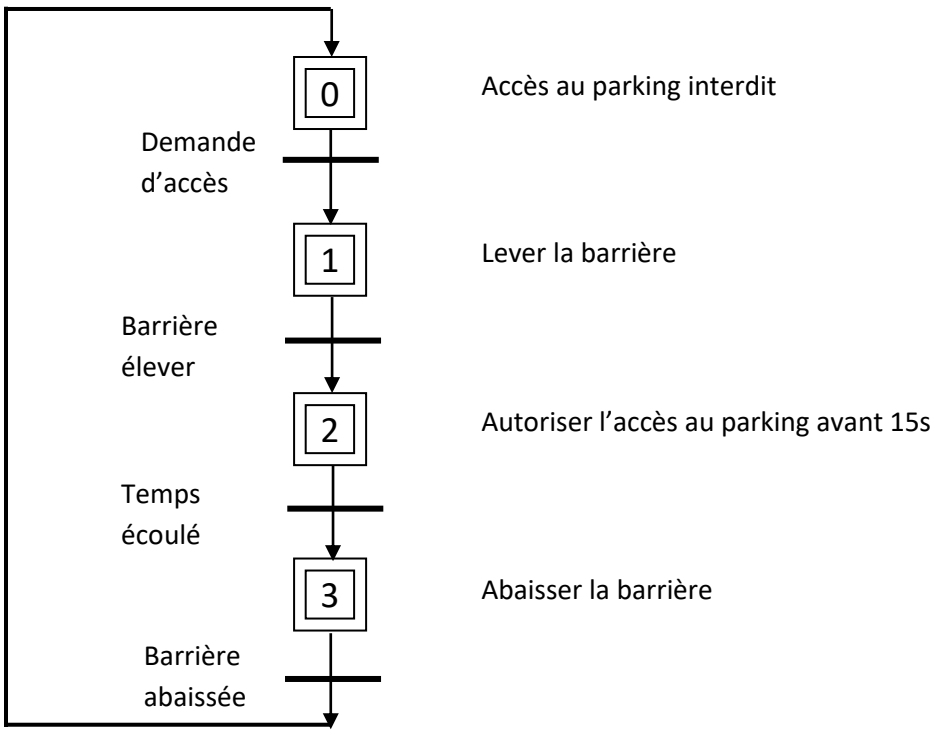
L'analyse statique : la fonction statique, la méthode SADT niveau A-0. On va données le *contexte* :

Analyse statique

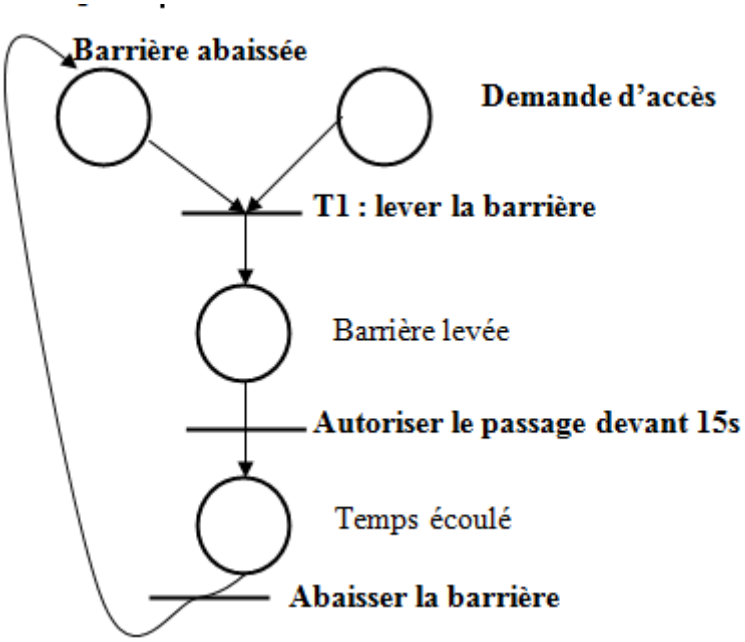


Analyse dynamique :

Grafcet :



Réseau de pétri :



Présentation

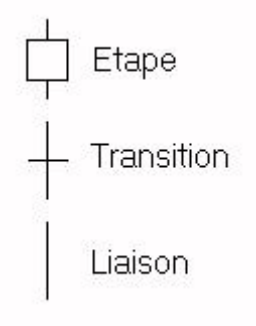
Le Grafcet pour Graphe fonctionnel de Commande Etapes Transition est un outil extrêmement puissant et bien adapté pour représenter la commande d'un système. C'est une adaptation des réseaux de Petris à la commande des systèmes. De plus il à l'avantage d'être normalisé et extrêmement utilisé dans l'industrie.

Comme son nom l'indique il s'agit d'un graphe, donc c'est agréable et facilement accessible au non initié, de plus c'est intuitif.

Les 3 éléments graphiques de bases sont :

- les étapes
- les transitions
- les liaisons

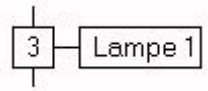
Les liaisons permettent de joindre les étapes et les transitions (à angle droit d'après la norme). La figure suivante montre ces éléments.



Les étapes

Les étapes représentent une position dans laquelle est le système. Une étape peut être active ou inactive. L'ensemble des étapes actives indique l'état du système. On associe généralement les sorties aux étapes. De plus les étapes portent des numéros.

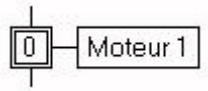
La figure suivante indique que lorsque l'étape 3 est active alors il faut allumer la lampe 1.



Ainsi lorsqu'une étape est active alors la ou les sorties associées sont activées.

En fait il existe deux sortes d'étapes, les étapes "normales" et les étapes "initiales" qui sont actives à l'initialisation, les "normales" étant bien sur inactives.

La figure suivante montre une telle étape.



Les transitions

Les transitions permettent de passer d'une étape à une autre. Les transitions sont reliées aux entrées le plus souvent et possèdent une équation logique. Si cette expression logique est vérifiée (vrai) alors la transition est dite réceptive ou active.

La figure suivante montre une étape avec son équation logique.

97 

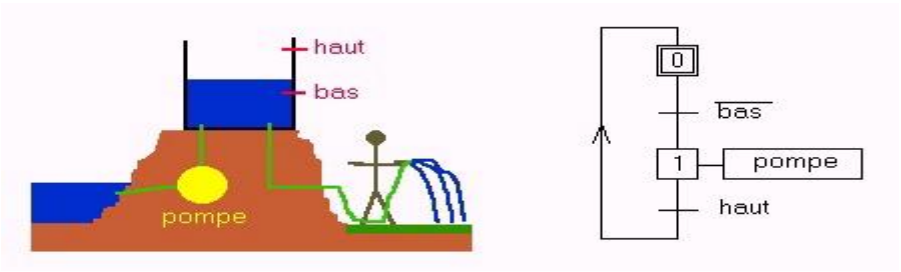
Si une transition et l'étape qui la précède sont actives alors l'étape précédente devient inactive et l'étape suivante active.

Le franchissement s'effectué ssi :

- l'étape précédente active,
- la réceptivité de la transition est vraie.

Un exemple simple

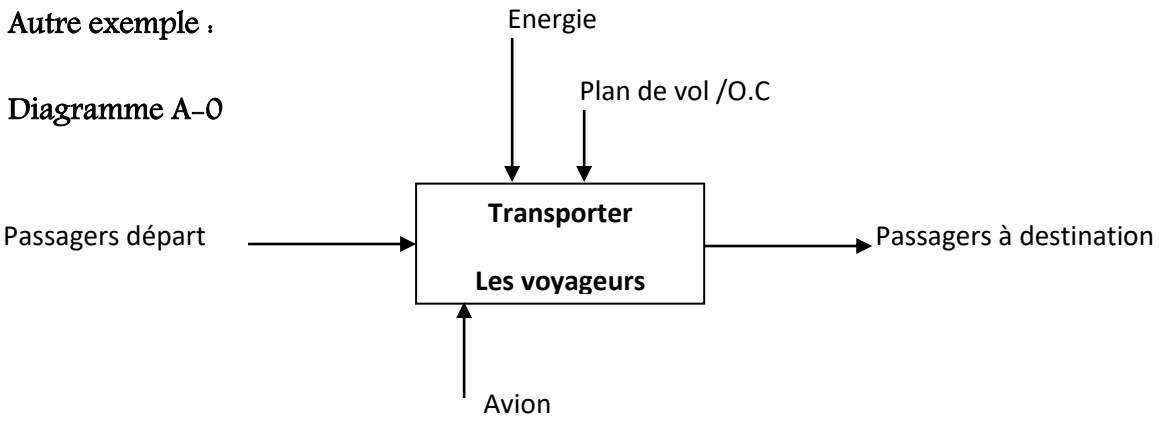
Pour illustrer ces notions, prenons un exemple simple. Il s'agit de remplir une réserve d'eau avec une pompe. Lorsque le capteur bas n'est plus recouvert alors la pompe se met en route, cette dernière s'arrête lorsque le capteur haut est recouvert. Un capteur recouvert vaut 1, sinon 0.

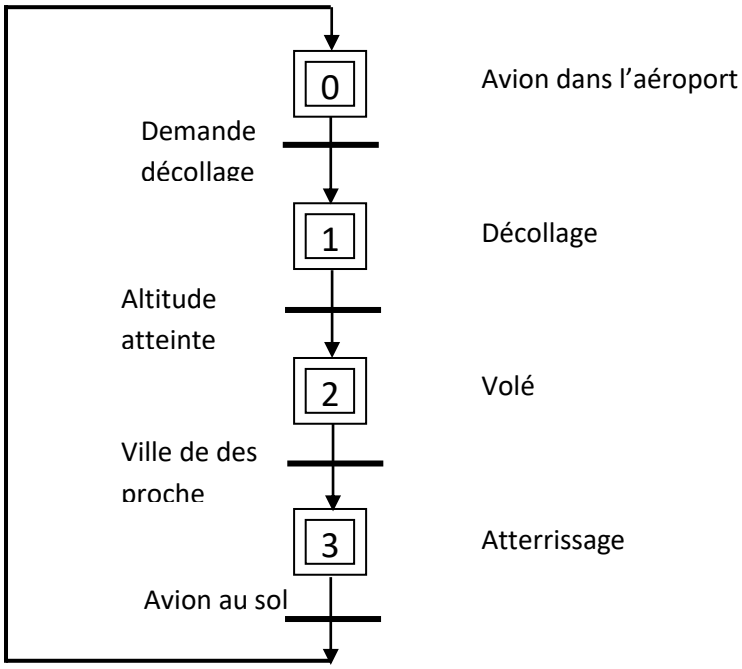


A l'initialisation, l'étape 0 est active et la 1 inactive. La pompe est arrêtée car aucune étape ne la commandant (ici la 1) est active. Lorsque le niveau d'eau passe sous le capteur bas, alors celui ci n'est plus recouvert, donc la transition "/bas" est vérifiée. Il s'en suit une désactivation de l'étape 0 et une activation de l'étape 1 d'ou la mise en route de la pompe. Grâce à la pompe, le niveau d'eau monte et lorsqu'il recouvre le capteur haut, alors la transition "haut" devient active d'ou la désactivation de l'étape 1 et de la pompe et l'activation de l'étape 0.

Autre exemple :

Diagramme A-0





b. Les réseaux de Pétri

Introduction

Cette technique formelle et opérationnelle est particulièrement bien adaptée pour décrire le *comportement des systèmes asynchrones* avec des évolutions parallèles.

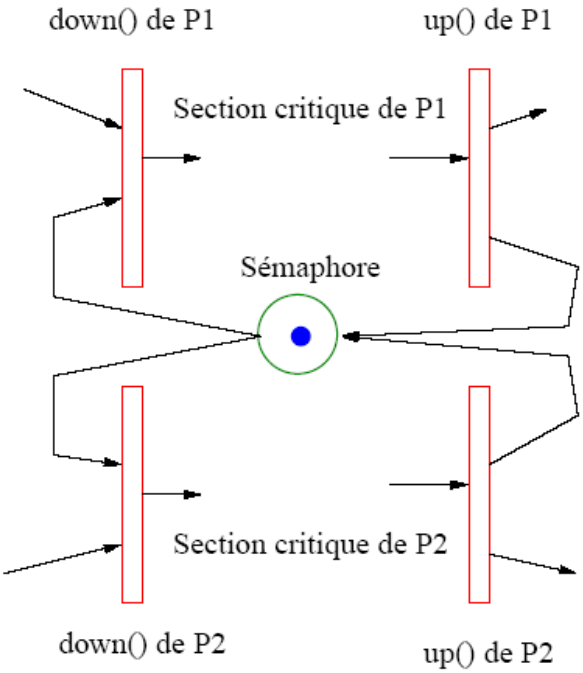
Définition :

Un réseau de Petri (RdP) est un graphe biparti comportant des places et des transitions, il est constitué :

1. D'un ensemble fini de places (représentée par un cercle),
2. D'un ensemble fini de transitions (représentée par un trait),
3. D'un ensemble fini de flèches, connectant soit des places à des transitions, soit des transitions à des places.

☞ Le nombre de places et de transitions est fini, non nul

Modélisation d'un sémaphore



c. Réseau de pétri marqué

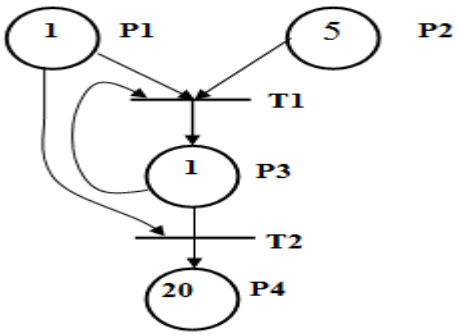
Un réseau de pétri marqué **N=<R, M>** est formé d'un réseau de pétri R et à une application M. Chaque place peut contenir un ou plusieurs *jetons*. L'état du RdP est défini par le *marquage* de ses places.

L'évolution du marquage obéit à la règle suivante :

- chaque transition a des places d'entrée et des places de sortie ;
- si toutes les places d'entrée contiennent au moins un jeton, la transition est franchissable ; elle peut alors être franchie ('tirée'), ce qui retire un jeton de chaque place d'entrée et ajoute un jeton dans chaque place de sortie.
- Si plusieurs transitions sont franchissables, le choix de celle qui est tirée est indéterministe.

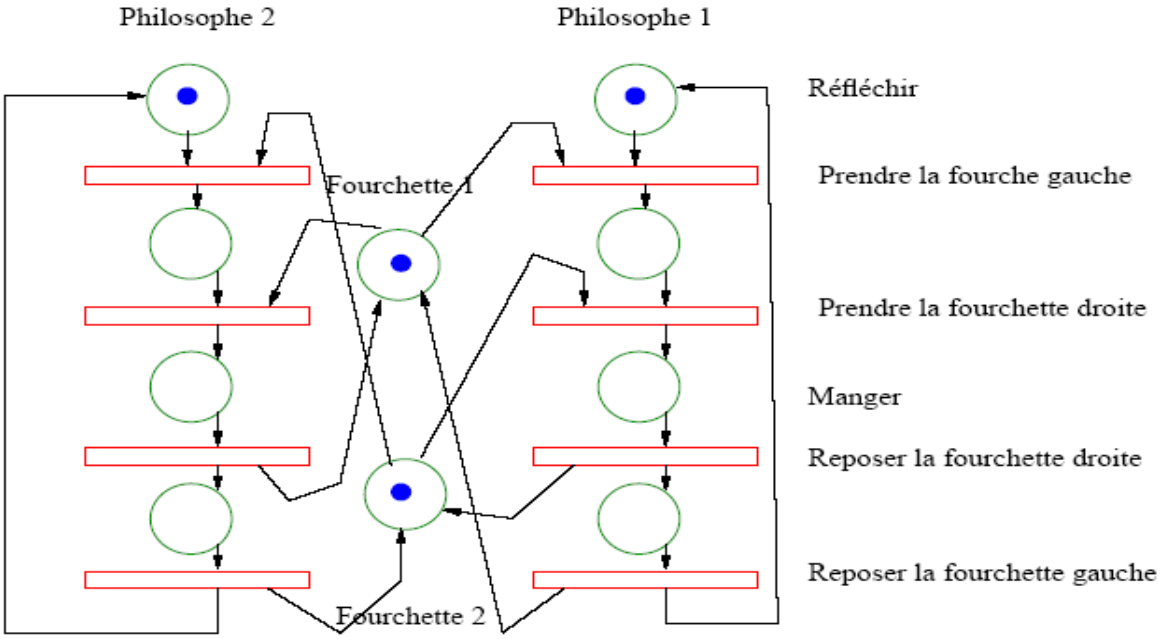
Les transitions modélisent en général des actions et la présence des jetons la satisfaction des conditions nécessaires à leur réalisation.

Exemple :



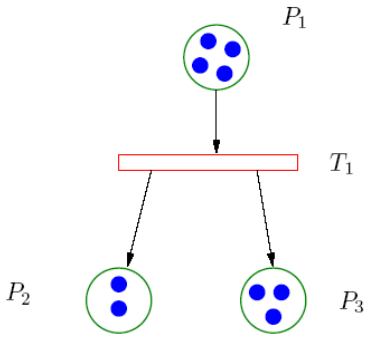
P	P1	P2	P3	P4	
M0	1		5	1	20
T1M1	0		3	1	20

Exemple : les philosophes



1. Chaque place peut contenir un nombre entier (positif ou nul) de jetons ou marques
2. Le nombre de marques de la place i sera noté M(Pi) ou Mi.
3. Le franchissement d'une transition ne peut s'effectuer que si chacune des places en amont de cette transition est validée ou franchissable, c'est à dire contient au moins une marque
4. Une transition sans place amont est toujours validée : c'est une transition source
5. Le franchissement (ou tir) d'une transition T consiste à retirer une marque dans chacune des places amont et à ajouter une marque dans chacune des places aval de la transition T.
6. Parmi plusieurs transitions franchissables, une seule est franchie à la fois.
7. Le franchissement d'une transition est considéré comme une opération atomique (indivisible).

marquage = (4,2,3)



Fonctionnement de réseau de pétri marqué

- Règle de déplacement de transition :

Une transition Test franchissable (déclanchable) pour un marquage M si pour toute place P en entrée de T on à : M(P) >= V(P,T)

- Règle de consommation et de production de jetons :

La franchissement de la transition T induite une consommation des jetons dans les places en entrées Pe et une production des jetons dans les places en sorties Ps.

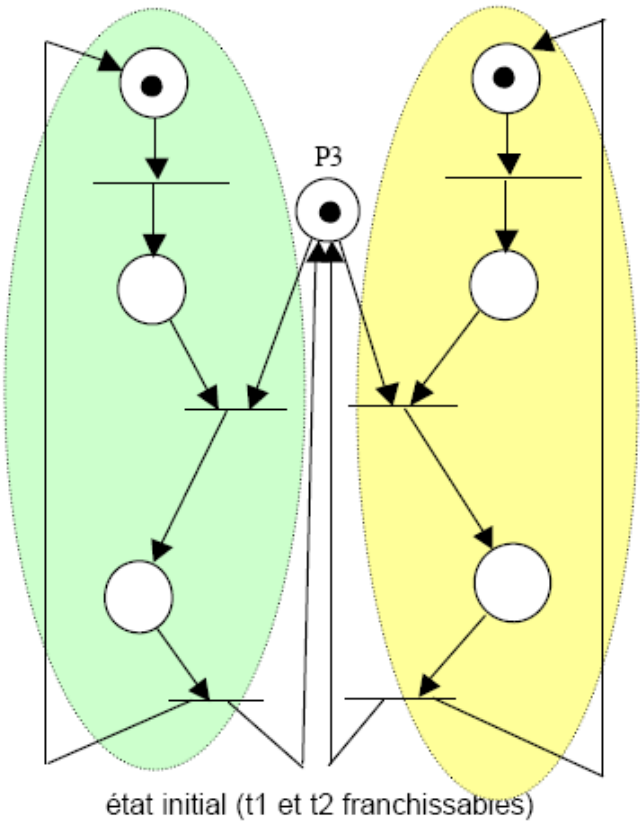
- Les places et transition ont la sémantique suivante :
  - Places avec jeton indique la présence de ressource,
  - Transition représente l'action eu évènement,
  - Le marquage indique les contraintes de synchronisation.

Exemple : un RdP et son évolution.

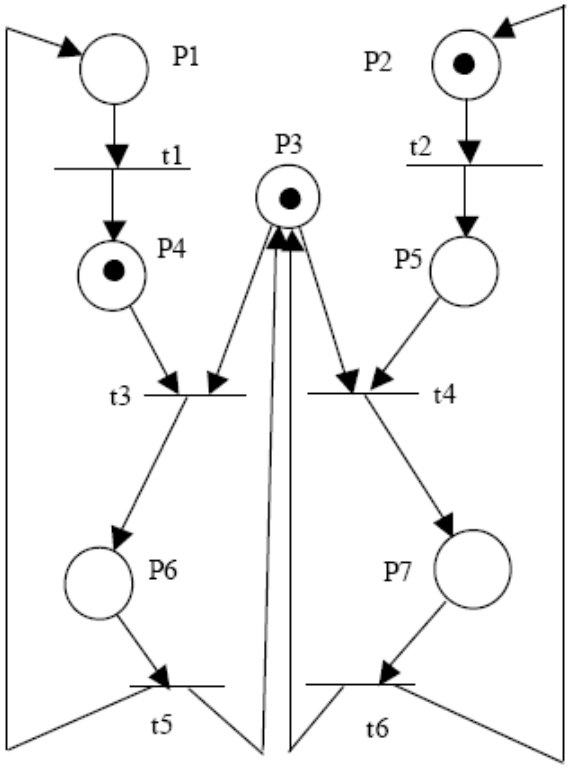
On peut interpréter ce réseau comme deux activités asynchrones qui se partagent une ressource (modélisée par la place P3).

Les 2 jetons dans P1 et P2 modélisent l'état des 2 activités (verte et jaune).

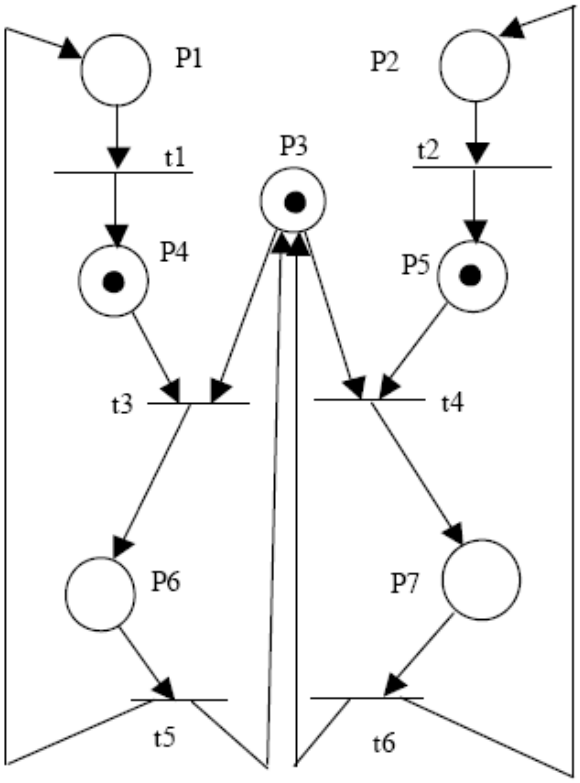
Le jeton dans P3 modélise la ressource prise par une des 2 activités (transitions t3 et t4) et rendue après utilisation (transitions t5 et t6).



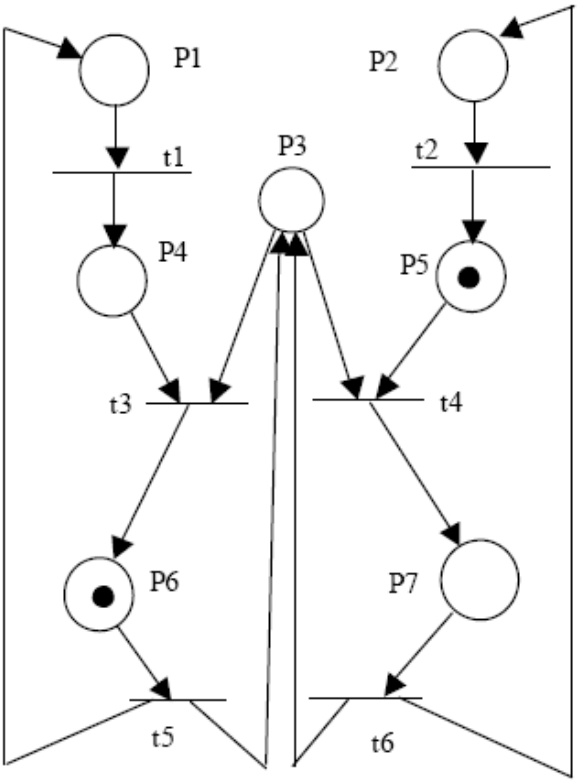
état initial (t1 et t2 franchissables)



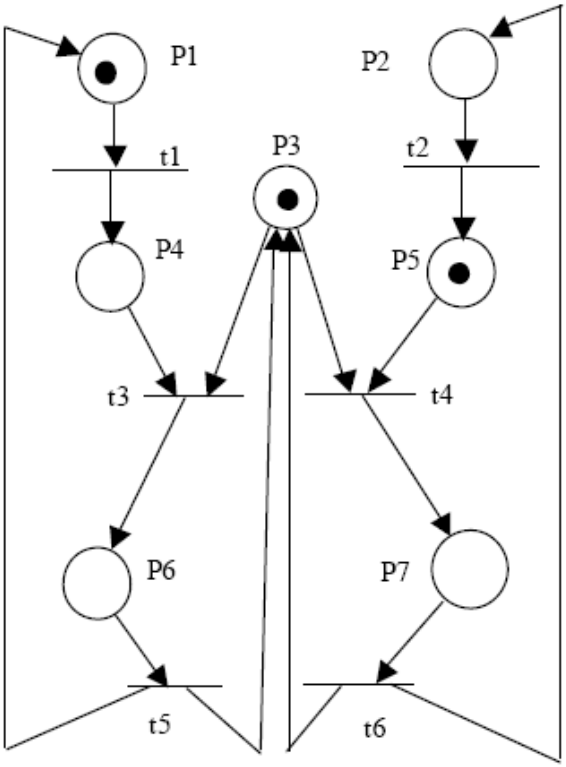
t1 franchie (t2 et t3 franchissables)



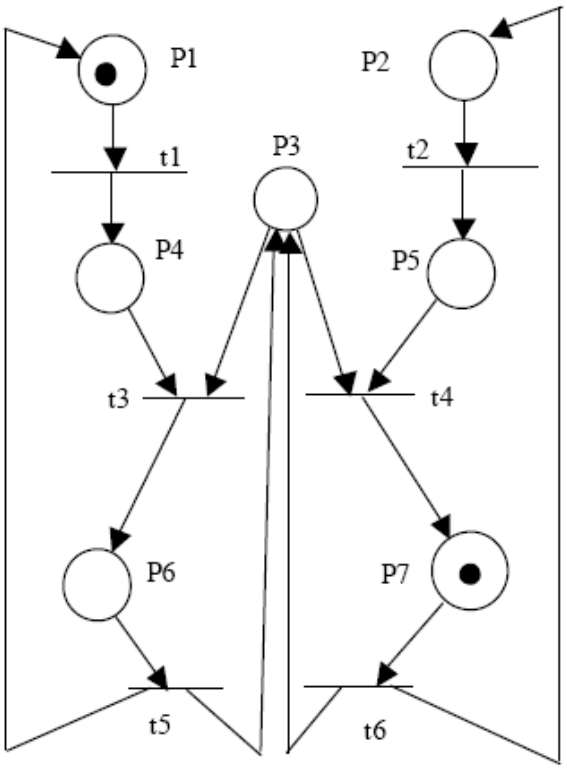
t2 franchie (t3 ou t4 franchissables)



t3 franchie (t5 franchissable)
la ressource est prise par l'activité de gauche



t5 franchie (t1 et t4 franchissables)
la ressource est rendue par l'activité de gauche

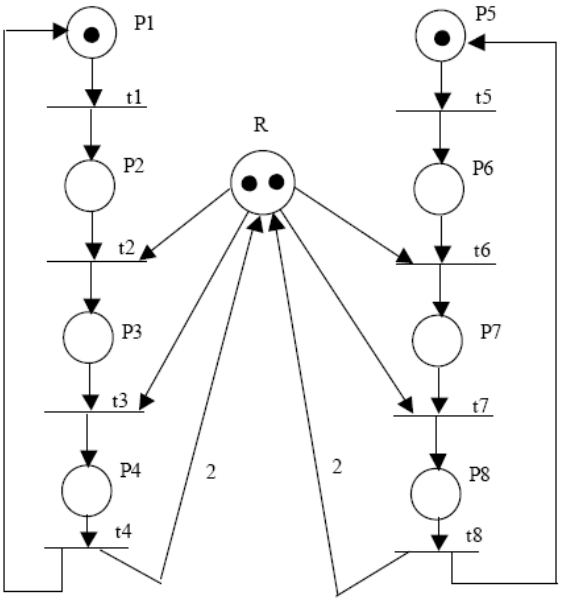


t4 franchie ((t1 et t6 franchissables)
la ressource est prise par l'activité de droite

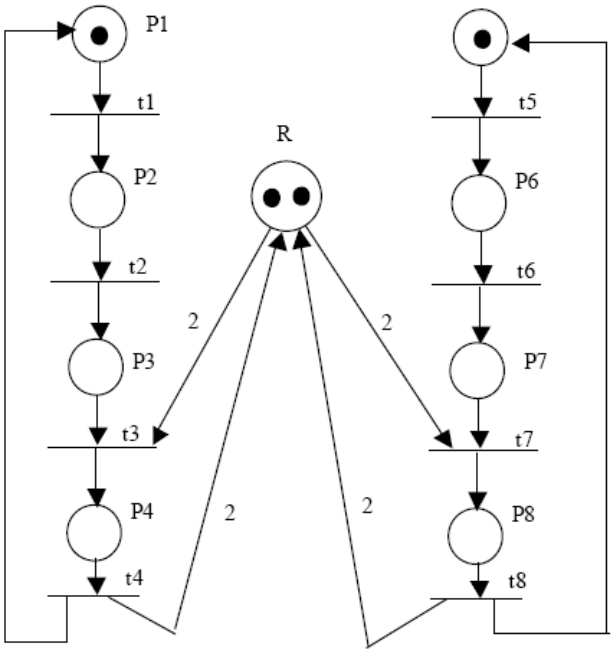
Si on suppose que les 2 activités se partagent 2 ressources, on peut avoir par exemple le RdP de la figure suivante. Celui-ci **souffre d'un danger d'interblocage**.

En effet si un jeton de R est consommé par t2 et l'autre immédiatement après par t6, t3 et t7 ne peut plus être franchies faute de jeton restant dans R.

Rien ne peut plus évoluer. (Le chiffre 2 sur certains arcs signifie que 2 jetons sont produits par la transition sur cet arc.)



Au contraire, le RdP de la figure suivante est ***vivant***, c'est à dire sans possibilité d'interblocage, car les 2 jetons de R sont consommés et rendus simultanément.



Validation des réseaux RdP

Il existe des techniques mathématiques permettant de prouver des propriétés des RdP (par exemple, existence d'interblocage ou caractère vivacité du réseau). Une technique de base est la construction de *l'arbre de tous les marquages accessibles*.

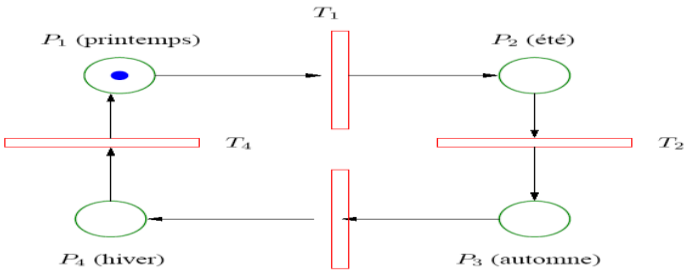
Les RdP permettent de modéliser très facilement le producteur consommateur introduit dans le paragraphe sur les machines à états finis. On n'y observe pas la même

explosion combinatoire du nombre de places car chaque sous système conserve son état (jeton).

Les RdP souffrent eux aussi de plusieurs limitations que diverses extensions tentent de contourner : typage des jetons (RdP colorés), ajout de prédicats, ajout de conditions ou caractéristiques temporelles au franchissement des transitions, etc.

Il s'agit du modèle de comportement le plus fréquemment utilisé

Exemple :



VIII. CONCLUSION

Parler des techniques de spécification est comme parler des langages de programmation :

Il n'y a ni langage ni technique idéale, ni langage ni technique permettant de tout faire.

L'informaticien doit avoir une culture assez étendue des diverses techniques comme des divers langages. Souvent les techniques de spécifications se complètent, en décrivant des vues complémentaires d'un système.

Par exemple, un système peut être spécifié à travers un diagramme de flot de données (sources d'informations, types d'informations stockées et échangées, décomposition en fonctions), un schéma EA (structuration des informations) et des machines à états finis (comportement de certains composants).

Les *méthodes tentent de proposer des assemblages efficaces de telles techniques avec des guides pour les construire et les valider.*

Il faut rester cependant relativiste : il est rare dans la pratique de rencontrer des projets dans laquelle la phase de spécification utilise les diverses méthodes de spécification.

La plupart du temps une seule méthode est utilisée parce que c'est celle que le spécificateur connaît le mieux ou à le plus l'habitude d'utiliser.

Par ailleurs, il n'est pas facile (malgré les outils existants) de concilier ces différentes spécifications parfois assez hétérogènes, et l'expertise demandée aux clients pour les comprendre l'est tout autant.

Pour cette raison les fabricants d'atelier de génie logiciel tentent de produire des AGL basés sur des méthodes intégrant tous les avantages des méthodes précédemment citées (UML a par exemple de nombreuses extensions) en reprenant leurs principes de base.

Objectif du Chapitre 2:

A la fin du chapitre l'étudiant sera capable de :

Définir le génie logiciel, Justifier son importance, Présenter le cycle de vie du logiciel, Faire la distinction entre développement "artisan" et approche "industrielle», d'Introduire les concepts clés et les bonnes pratiques et Donner une vue d'ensemble des rôles et des outils

CHAPITRE II: INTRODUCTION AU GENIE LOGICIEL

I. INTRODUCTION

1. Logiciel

Un logiciel:= Ensemble

- de programmes
- de procédés
- de règles
- de documentation

Un logiciel est un produit atypique comparativement aux produits industriels, on peut parler d'avions type, de voitures types,... mais il est quasiment impossible de parler de logiciel type.

2. Caractéristiques d'un logiciel

- Diversité des applications
- Taille et complexité
- Abstraction et invisibilité
- Coût élevé du logiciel
- Evolutivité des besoins
- Distribution et parallélisme

II. CRISE DU LOGICIEL

1. Historique des faits marquants (remarquable)

• **1945:** La programmation s'effectue en code binaire et ensuite en assembleur mais seul celui qui développe est capable de comprendre et de maintenir son projet. Les projets sont alors de petite taille

• **1955:** Apparition et utilisation de langages évolués qui permettent de développer des projets + importants

• **1965:** Crise du logiciel: On se rend compte que l'intuition ne suffit plus pour développer correctement du logiciel

La crise du logiciel se manifeste à travers

– le dérapage des délais et des coûts de développement de la plupart des projets informatiques,

– la réalisation de logiciels de mauvaise qualité,

• Qui ne satisfont pas les utilisateurs,

• Qui ne sont parfois jamais utilisés,

• Ou qui nécessitent de nombreuses et coûteuses révisions.

Exemples :

- Refus illégaux de prestations sociales à des ayants droits.

Cause : le cahier des charges n'avait pas prévu tous les cas

- Mission Vénus : passage à 5 000.000 km de la planète au lieu des 5000 km prévus

Cause : remplacement d'une virgule par un point

- 2 jours sans électricité pour la station Mir en 1997.

Cause : plantage d'un ordinateur qui contrôlait l'orientation des panneaux solaires

- Perte de satellites dans les années 70.

Cause : +I au lieu de +1 dans une boucle du programme source Fortran (le langage admet sans le signaler des identificateurs non déclarés, leur valeur est initialisée aléatoirement)

- Y2K. Dysfonctionnements possibles prévus, combat contre le bogue de l'an 2000 : 500 milliards de francs, quelques dysfonctionnements constatés

Cause : l'année codée sur 2 caractères pour gagner de la place

- Socrate : système de réservation de places de la SNCF. Mauvaise ergonomie, manque de formation → report (ajournement) important et durable de la clientèle vers d'autres moyens de transports

Cause : rachat par la SNCF d'un système de réservations de places d'une compagnie aérienne sans réadaptation totale au cahier des charges du transport ferroviaire.

• **1968**: Première conférence sur le GL

• **1970**: Définition de la notion de programmation structurée: suppression de l'utilisation du GOTO, structuration du code en niveaux hiérarchiques

• **1972**: Dvt des méthodes de preuves de programmes (difficiles à appliquer à de grands logiciels)

• **1975**: On se rend compte que développer un projet ne consiste pas seulement à le coder mais à le comprendre, le spécifier et le concevoir en des étapes successives, d'où apparition de la notion de cycle de vie et essais de développement de méthodes adaptées à ces phases

• 1980: Après avoir développé des méthodes et des outils de manière isolée, on les rassemble pour former des environnements homogènes. On prend conscience de l'importance des premières phases dans le coût de dvt d'un projet. Il semble logique qu'une erreur de conception soit moins coûteuse à corriger lors de la conception que lors d'une phase suivante, comme l'implantation ou la maintenance

• 1990: C'est la décennie de la programmation OO, avec comme objectifs,

– d'une part, la réutilisation des logiciels et,

– d'autre part, le passage aussi naturel que possible pour l'utilisateur d'une application à une autre (de Word à Excel par exemple).

2. Symptômes les plus caractéristiques de cette crise

– Les logiciels réalisés ne correspondent souvent pas aux besoins des utilisateurs

– Les logiciels contiennent trop d'erreurs (qualité du logiciel insuffisante)

– Les coûts de développement sont rarement prévisibles et sont généralement prohibitifs

– La maintenance des logiciels est une tâche complexe et coûteuse

– Les délais de réalisation sont généralement dépassés

– Les logiciels sont rarement portables

3. Quelques sources de la crise

Une idée grossière du logiciel à réaliser est suffisante pour commencer d'écrire un programme (il est assez tôt de se préoccuper des détails plus tard).

– Faux : une idée imprécise du logiciel à réaliser est la cause principale d'échecs

Une fois que le programme est écrit et fonctionne, le travail est terminé.

– Faux : la maintenance du logiciel représente un travail important : le coût de la maintenance représente plus du 50 % du coût total d'un logiciel

4. Crise logiciel

La **crise du logiciel** est un terme utilisé dans les premiers jours de 'ingénierie du logiciel pour décrire l'impact de la croissance rapide de la puissance ordinateurs et de la complexité des problèmes qui doivent être pris en compte. Les mots-clés de la crise étaient des logiciels **complexité**, **attentes** et **changement**.

Le concept de crise du logiciel a vu le jour à la fin de 1960. Une ancienne utilisation du terme était dans la conférence ACM Turing Award, « Le programmeur Humble » (EWD340), de Edsger Dijkstra publié en 1972 Communications de l'ACM.

Les causes de la crise sont liées à la complexité des logiciels des processus logiciels et l'immaturité relative de l'ingénierie logicielle. La crise se manifeste de plusieurs façons:

- Projets sur le budget
- Les projets au-delà des délais
- Logiciel de mauvaise qualité
- Le logiciel qui souvent ne répondait pas aux exigences
- Les projets et le code ingérable difficile à maintenir

La crise du logiciel a conduit alors à la naissance du génie logiciel et les premiers modèles, tels que le modèle de cascade.
Pour surmonter la crise, en fait, ils devaient être introduits:

- gestion;
- Organisation, par l'analyse et la conception;
- Théories et techniques, y compris programmation structurée et Programmation orientée objet;
- Outils, y compris Environnement de développement intégré;
- Méthodologies, y compris la modèle de cascade et méthodologie agile.

III. LE GENIE LOGICIEL

Domaine des 'sciences de l'ingénieur' dont la finalité est la *conception*, la *fabrication* et la *maintenance de systèmes logiciels complexes, sûrs et de qualité* ('Software Engineering' en anglais). Aujourd'hui les économies de tous les pays développés sont dépendantes des systèmes logiciels.

Le GL se définit souvent par opposition à la 'programmation', c'est à dire la production d'un programme par un individu unique, considérée comme 'facile'. Dans le cas du GL il s'agit de la fabrication *collective* d'un *système complexe*, concrétisée par un ensemble de documents de conception, de programmes et de jeux de tests avec souvent de *multiples versions* (« multi-person construction of multi-version software »), et considérée comme 'difficile'.

Objectifs – la règle du CQFD

Le GL se préoccupe des *procédés de fabrication des logiciels* de façon à s'assurer que les 4 critères suivants soient satisfaits.

· Le système qui est fabriqué répond aux *besoins* des utilisateurs (correction fonctionnelle).

· La *qualité* correspond au contrat de service initial. La qualité du logiciel est une notion multiforme.

· Les *coûts* restent dans les limites prévues au départ.

· Les *délais* restent dans les limites prévues au départ.

Règle du CQFD : Coût Qualité Fonctionnalités Délai.

1. La qualité exigée d'un Logiciel

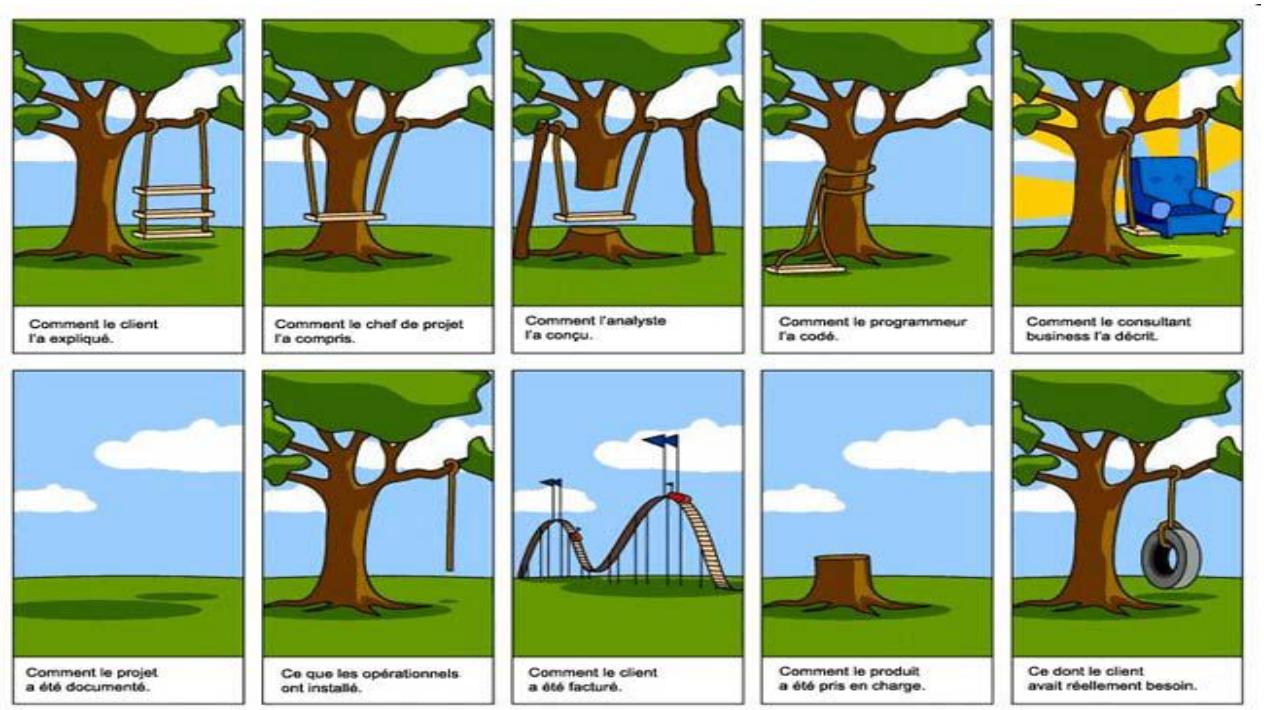


Figure 18: Relation client chef projet

Règle du CQFD : Coût Qualité Fonctionnalités Délai

Le GL se préoccupe des procédés de fabrication des logiciels de façon à s'assurer que les 4critères suivants soient satisfaits:

- Le système qui est fabriqué répond aux besoins des utilisateurs (correction fonctionnelle).
- Les coûts restent dans les limites prévues au départ.
- Les délais restent dans les limites prévues au départ.
- La qualité correspond au contrat de service initial. La qualité du logiciel est une notion multiforme qui recouvre:

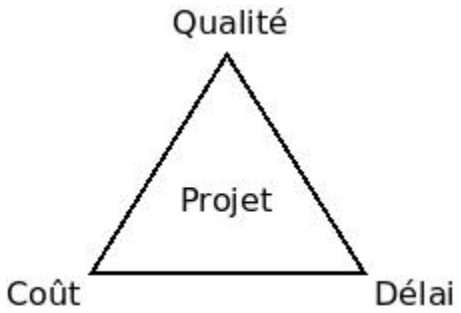


Figure 19: Règle du CQFD : Coût Qualité Fonctionnalités Délai

2. Pyramide de qualité d'un logiciel

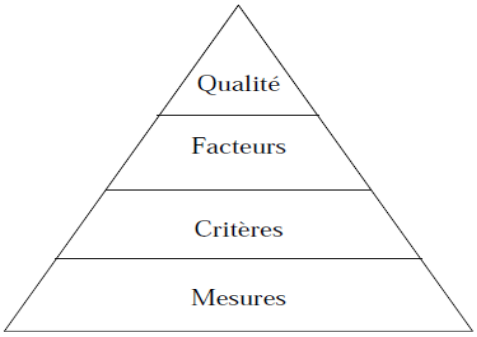


Figure 20: Evaluer la qualité

a. Qualité

Ensemble de caractéristiques que doit satisfaire un produit pour répondre aux besoins

b. Plan Qualité Logiciel (PQL)

Document précisant pour un logiciel donné les phases de développement et les facteurs et critères de qualités, ainsi que les niveaux requis pour ces derniers.

c. Système Qualité

Dispositif mis en place par une entreprise pour vérifier le respect de la procédure d'Assurance Qualité

3. Propriétés de qualité à vérifier dans un Logiciel

d. La qualité

Correspond au contrat de service initial. La qualité du logiciel est une notion multiforme qui recouvre:

e. La validité:

Aptitude d'un logiciel à réaliser exactement les tâches définies par sa spécification,

f. La fiabilité:

Aptitude d'un logiciel à assurer de manière continue le service attendu,

g. La robustesse:

Aptitude d'un logiciel à fonctionner même dans des conditions anormales,

h. L'extensibilité:

Facilité d'adaptation d'un logiciel aux changements des spécifications,

i. La réutilisabilité:

Aptitude d'un logiciel à être réutilisé en tout ou partie,

j. La compatibilité:

Aptitude des logiciels à pouvoir être combinés les uns aux autres,

k. L'efficacité:

Aptitude d'un logiciel à bien utiliser les ressources matérielles telle la mémoire, la puissance de l'U.C., etc.

l. La portabilité:

Facilité à être porté sur de nouveaux environnements matériels et/ou logiciels,

m. La traçabilité:

Capacité à identifier et/ou suivre un élément du cahier des charges lié à un composant d'un logiciel,

n. La vérifiabilité:

Facilité de préparation des procédures de recette et de certification,

o. L'intégrité:

Aptitude d'un logiciel à protéger ses différents composants conte des accès ou des modifications non autorisés,

p. La facilité :

D'utilisation, d'entretien, etc.

4. Les principes du Génie Logiciel

Le génie logiciel est une discipline de l'ingénierie qui vise à concevoir, développer, tester, déployer et maintenir des logiciels de manière structurée, fiable et efficace. Il repose sur plusieurs principes clés pour garantir la qualité, la maintenabilité et la pérennité des systèmes logiciels.

Principes fondamentaux du Génie Logiciel :

a. Modularité

Découper le logiciel en composants indépendants (modules), facilitant la compréhension, la maintenance et la réutilisation.

b. Abstraction

Masquer les détails techniques pour ne montrer que les informations essentielles d'un composant ou d'un système.

c. Réutilisabilité

Favoriser le développement de composants génériques réutilisables dans d'autres projets ou contextes.

d. Maintenance facile

Concevoir le logiciel pour qu'il soit simple à corriger, modifier et faire évoluer dans le temps.

e. Faible couplage et forte cohésion

- Faible couplage : minimiser les dépendances entre modules.
- Forte cohésion : chaque module remplit une fonction claire et précise.

f. Spécification claire

Bien définir les besoins et les exigences du système dès le départ, pour éviter les malentendus.

g. Testabilité

Concevoir le logiciel de manière à faciliter les tests, pour garantir la qualité et détecter les erreurs tôt.

h. Traçabilité

Assurer un suivi des exigences, des modifications et des décisions tout au long du cycle de vie du logiciel.

i. Évolutivité

Concevoir le logiciel pour qu'il puisse s'adapter facilement aux évolutions des besoins ou de la technologie.

j. Documentation

Fournir une documentation claire, complète et à jour, essentielle pour la maintenance, le transfert de compétences, et l'utilisation du logiciel.

Le But du génie logiciel est de concevoir des logiciels fiables, efficaces, robustes et adaptés aux besoins réels des utilisateurs, tout en respectant les coûts, les délais et la qualité.

IV. CONCLUSION

Le génie logiciel est une discipline fondamentale de l'informatique moderne, qui vise à maîtriser la complexité croissante des logiciels tout en garantissant leur qualité, leur fiabilité et leur maintenabilité. À travers ce chapitre, nous avons compris que développer un logiciel ne se limite pas à écrire du code, mais nécessite une approche structurée, collaborative et méthodique.

Nous avons découvert les principales étapes du cycle de vie d'un logiciel, les enjeux de la qualité logicielle, ainsi que les rôles et outils impliqués dans un projet logiciel. Cette introduction pose ainsi les bases nécessaires pour aborder plus en profondeur les différentes méthodes, techniques et bonnes pratiques du génie logiciel dans les chapitres suivants.

Objectif du Chapitre 3:

A la fin du chapitre l'étudiant sera capable de :

Comprendre la notion de cycle de vie du logiciel, Décrire les principales étapes du cycle de vie, Connaître les modèles de cycle de vie, Comprendre les avantages et inconvénients de chaque modèle, Appréhender les enjeux liés à chaque phase, Se sensibiliser à la documentation et à la gestion des versions dans le cycle de vie

CHAPITRE III: LE CYCLE DE VIE DU LOGICIEL

I. INTRODUCTION

1. Notion de cycle de vie

C'est la description d'un processus couvrant les phases de:

- **Création** d'un produit,
- **Distribution** sur un marché,
- **Disparition.**

Le but de ce découpage est de

- Maîtriser les risques,
- Maîtriser au mieux les délais et les coûts,
- Obtenir une qualité conforme aux exigences.

On distingue deux types de cycle de vie

- Le cycle de vie des produits s'applique à tous les types de produits, et peut être considéré comme un outil de gestion.
- Le cycle de développement des logiciels s'insère dans le précédent, on l'appelle souvent abusivement cycle de vie des logiciels

2. Justification du cycle de vie

Cycle de vie et assurance qualité sont fortement liés; il faudra donc ***en permanence*** assurer:

- la **validation**: sommes-nous en train de faire le bon produit?
- la **vérification**: est ce que nous faisons le produit correctement

La validation et la vérification sont en général garanties par la mise en place d'inspections et de revues. L'inspection est une **lecture critique** d'un document (spécification, conception, code, plan d'intégration...); elle est destinée à améliorer la qualité d'un document.

Les inspections sont à la base des décisions prises en **revues.** Une revue est une réunion permettant de valider une des phases du cycle de vie.

On distingue

- les **revues produites**: état d'un projet sous ses différents aspects: Techniques, Financiers, Commerciaux, Calendrier, ...

- les **revues techniques** (celles qui nous intéressent le plus dans le cadre de ce cours): elles permettent de fournir au marketing et à l'unité de développement une évaluation des aspects techniques du projet et des coûts de réalisation

- les **réunions de décision**: elles valident le passage à la phase suivante et font bien souvent suite à l'une des deux précédentes.

Chaque objectif intermédiaire doit être atteint:

- **Garantie de qualité**

Exemple d'après Boehm

Logiciel de réservation aérienne (Univac-United Airlines) d'un coût de 56 millions de dollars non utilisable par manque d'analyse des besoins et d'étude de faisabilité:

- 146 000 instructions par transaction,
- au lieu de 9 000 prévues.

Ceci aurait pu être évité par des inspections et des revues intermédiaires.

- **Garantie d'efficacité**

Il est plus facile de respecter un objectif à court terme qu'à moyen ou long terme

- **Tout ordre différent conduira à un produit moins satisfaisant et beaucoup plus cher.**

Les erreurs sont de plus en plus coûteuses à réparer lorsqu'elles sont découvertes tard dans le cycle de vie: d'où le rôle primordial des inspections.

3. Les étapes du cycle de vie

a. Définition des besoins

L'expression des besoins est cruciale pour établir une base solide pour le projet. Elle implique de recueillir et de documenter en détail les attentes et exigences des parties prenantes et des utilisateurs.

Ce processus, qui peut inclure des entretiens et des questionnaires, permet de comprendre les besoins spécifiques et de les intégrer dans la planification du projet. L'expression des besoins est dynamique, pouvant être mise à jour tout au long du projet pour s'adapter aux changements et évolutions.Elle guide la conception et la mise en œuvre du projet, en fournissant des orientations claires et précises.

b. Analyse des besoins

Dans cette phase, il est crucial de prendre en compte les contraintes de gestion de projet pour orienter notre analyse. On va maintenant examiner toutes les possibilités, en tenant compte de ces contraintes :

Écarter toute solution ne correspondant pas aux exigences et aux contraintes de gestion de projet identifiées.Affiner la sélection en considérant comment chaque option s'aligne avec ces contraintes.

Détailler les options, en mettant l'accent sur la manière dont elles répondent ou s'adaptent aux contraintes de gestion de projet. Préparer toutes les informations nécessaires, en tenant compte des contraintes de gestion de projet, pour l'étape suivante.

c. Conception du projet

Avant de passer à la rédaction de la charte de projet, cette phase est dédiée à l'élaboration d'une vision globale de l'initiative, sans entrer dans les détails techniques ou opérationnels.

La phase de conception ne se limite pas à la simple planification ; elle est essentielle pour aligner les visions et attentes des différentes parties prenantes.

L'usage d'outils visuels dans cette étape établit une fondation solide et partagée pour le projet.

d. Réalisation du projet

Cette phase, marquée par le développement de prototypes ou de versions bêta, intègre également les contraintes de gestion de projet dans une collaboration intense entre les différentes équipes.

Cela permet d'incorporer les retours des tests préliminaires dans le développement continu du projet.

Elle met l'accent sur l'adaptation agile du projet aux retours et aux découvertes faites pendant les tests, assurant ainsi que le produit final ou le résultat du projet soit de la plus haute qualité et réponde au mieux aux besoins des utilisateurs, tout en respectant les contraintes de gestion de projet.

e. Contrôle et validation

La phase de contrôle et validation en gestion de projet implique un suivi continu et une évaluation analytique des méthodes et résultats par rapport aux objectifs.

Cette étape permet d'identifier et de corriger les écarts.

Enfin, la clôture du projet marque la fin du cycle de vie du projet, avec un rapport final sur la réussite globale à présenter au sponsor.

f. Mise en service

La phase de mise en service comprend la préparation de la documentation finale et le transfert des connaissances, essentiels pour une transition fluide vers l'exploitation normale.

Elle vise également à préparer toutes les parties prenantes au changement, assurant ainsi leur préparation et leur adaptation au nouveau système ou processus.

Cette phase implique également l'évaluation de l'impact du projet sur l'organisation et ses clients, garantissant ainsi une transition en douceur et efficace vers la nouvelle phase d'exploitation ou de maintenance.

4. La conception du logiciel

La phase de conception suit immédiatement la phase d'analyse. Elle est prise en charge par l'équipe de développement. Elle vise à définir une architecture modulaire du logiciel ou encore une décomposition en modules qui facilitera la maintenance et permettra le développement parallèle par différents programmeurs. On regroupe en général sous le terme conception deux étapes distinctes :

- Conception globale : elle permet de définir les choix fondamentaux de l'architecture du logiciel en liaison avec l'architecture matérielle.
- Conception détaillée : elle décrit précisément chaque module, les algorithmes mis en œuvre et les traitements effectués en cas d'erreur. La documentation produite sera très utile en phase de maintenance. La phase de conception détaillée n'est pas traitée dans ce cours. Elle a souvent tendance aujourd'hui à être escamotée car la taille des modules étant réduite il n'est souvent pas nécessaire de décrire précisément les algorithmes mis en jeu.

La phase de conception globale se termine avec la rédaction du document de conception globale et du plan d'intégration qui ne sera pas traité ici.

6. La qualité d'une conception

Une « bonne » conception se définit en termes de la satisfaction des besoins et des spécifications : les critères de qualités peuvent donc varier énormément d'un système à un autre ou même pour deux implantations différentes d'un même système.
Par la suite on considérera de préférence les facteurs qui facilitent la gestion du produit tout le long de son cycle de vie. Il s'agit essentiellement de l'extensibilité, la réutilisabilité, la compatibilité, la portabilité la vérifiabilité et la facilité d'emploi.
Une bonne structuration participe largement à la production d'un logiciel qui possède ces caractéristiques. Comme dans beaucoup de domaine, cette bonne structuration se base sur la Modularité.

a. Cohésion

Définition: degré d'interaction à l'intérieur d'un module dans le but d'accomplir une et une seule fonction. Yourdon et Constantine ont établi une série de degrés de la cohésion classique pour décrire la cohésion des méthodes :

- Cohésion de coïncidence : Les éléments du module n'ont rien en commun, néanmoins ils sont présents dans le même module.
  - plusieurs fonctionnalités sans rapport
  - non réutilisable
- Cohésion logique: les éléments accomplissent des tâches similaires comme la gestion des entrées/sorties ou la gestion des fautes.
  - Plusieurs fonctionnalités reliées, sélectionnées par le module qui appelle
  - L'interface est difficile à comprendre
- Cohésion temporelle : les éléments réalisent des tâches semblables et exécutent leurs tâches dans la même tranche de temps.
  - Plusieurs fonctionnalités reliées dans le temps
  - contrôle.
  - flux de contrôle et opèrent sur le même ensemble de données.
  - mêmes données
  - communication et sont connectés par un flux de contrôle séquentiel.
  - indépendant mais qui utilisent les mêmes données
  - contribuent tous à la réalisation d'une tâche particulière. La cohésion fonctionnelle
  - représente la meilleure cohésion, elle vérifie le principe de la localité [YC79] et par
  - conséquent réduit l'effort de la maintenance.
- Cohésion procédurale : les éléments d'une méthode sont connectés par le même flux de contrôle.
  - plusieurs fonctionnalités reliées par la procédure du produit
  - non réutilisable
- Cohésion de communication: les éléments d'une méthode sont connectés par le même
  - flux de contrôle et opèrent sur le même ensemble de données.
  - plusieurs fonctionnalités reliées par la procédure du produit et qui utilisent les mêmes données non réutilisable
- Cohésion séquentielle: les éléments d'une méthode présentent une cohésion de communication et sont connectés par un flux de contrôle séquentiel.
  - plusieurs fonctionnalités ayant chacune leur point d'entrée et un bloc de code indépendant mais qui utilisent les mêmes données
  - les fonctionnalités sont structurées

- implantation d'un type abstrait de données

- Cohésion fonctionnelle : les éléments sont liés par une cohésion séquentielle et contribuent tous à la réalisation d'une tâche particulière. La cohésion fonctionnelle représente la meilleure cohésion, elle vérifie le principe de la localité et par conséquent réduit l'effort de la maintenance.
  - une seule fonctionnalité
  - réutilisable

Pressman représente les degrés de la cohésion, introduits par Yourdon et Constantine. Sous la forme d'un spectre

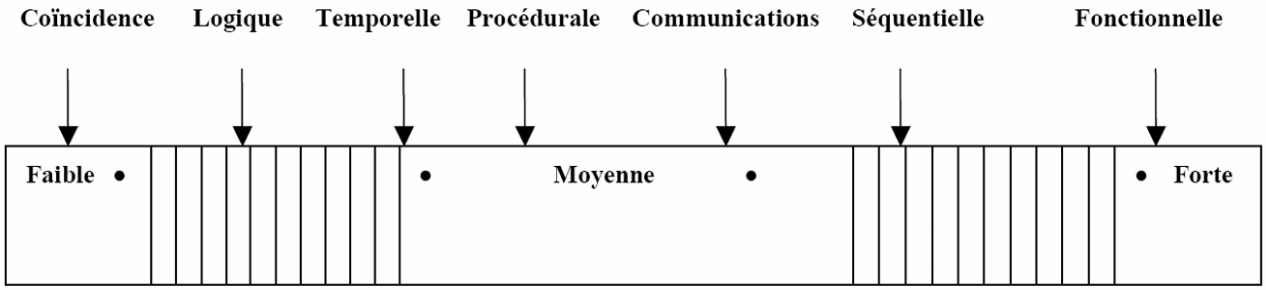


Figure 21 Niveau de Cohésion

b. Couplage

Degré d'interaction entre les modules d'un système Les sept degrés de couplage tel que rapportés par Pressman [Pre97] sont :

- Couplage Non direct : deux modules ne sont pas directement reliés.
- Couplage de Données : les données sont échangées en paramètre seulement.
- Couplage « Stamp » : un module passe à un autre une portion d'une structure de données.
- Couplage de Contrôle : un module passe un « flag » à l'autre qui s'en sert à des fins de contrôle d'exécution.
- Couplage Externe : un module est lié à quelque chose d'extérieur au logiciel.
- Couplage Commun: plusieurs modules partagent un ensemble des données.
- Couplage de Contenu : un module modifie des données ou des instructions d'un autre.

Pressman montre les degrés de couplage sous forme d'un spectre

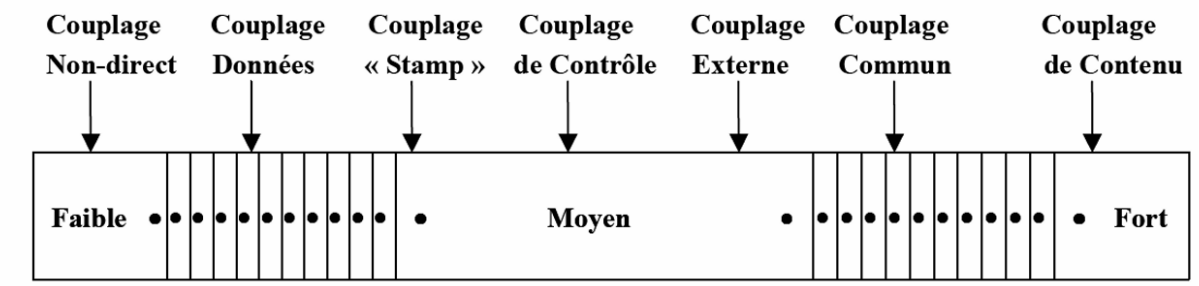


Figure 22 Niveau de Couplage

7. L'implémentation

L'implémentation est l'étape du développement logiciel où l'on traduit la conception en code exécutable, en utilisant un langage de programmation.

Son Objectif :

Transformer les spécifications techniques en un programme fonctionnel.

Plus de détaille (Détaille en chapitre 5)

8. La maintenance

La maintenance logicielle est l'ensemble des activités réalisées après la mise en service d'un logiciel pour corriger des erreurs, améliorer ses performances ou l'adapter à de nouveaux besoins.

Les Types de maintenance :

- Corrective : correction des bugs.
- Évolutive : ajout de nouvelles fonctionnalités.
- Préventive : amélioration du code pour éviter de futurs problèmes.
- Adaptative : adaptation à un nouvel environnement (matériel, OS, etc.).

Son Objectif :

Assurer que le logiciel reste fonctionnel, performant et adapté dans le temps.

Plus de détaille (Détaille en chapitre 5)

II. CONCLUSION

CHAPITRE III: LE CYCLE DE VIE DU LOGICIEL

Le cycle de vie du logiciel est un cadre structurant qui permet de planifier, organiser et gérer efficacement le développement d'un logiciel. En comprenant les différentes phases – de l'analyse des besoins à la maintenance – les développeurs et les équipes projet peuvent mieux anticiper les risques, assurer la qualité du produit final et répondre aux attentes des utilisateurs.

Les différents modèles de cycle de vie (cascade, en V, itératif, agile...) offrent des approches adaptées à divers types de projets. Ces modèles seront présentés dans le chapitre qui suit Le choix d'un modèle dépend du contexte, des exigences, et du niveau de flexibilité requis.

Ce chapitre nous a permis de saisir l'importance de chaque étape dans la réussite d'un projet logiciel, et prépare à l'étude plus détaillée des méthodes et outils associés à chaque phase dans les chapitres suivants.

CHAPITRE IV: LES MODELES DE DEVELOPPEMENT D'UN LOGICIEL

Objectif du Chapitre 4:

A la fin du chapitre l'étudiant sera capable de :

Présenter et expliquer les principaux modèles de développement logiciel, afin de permettre au lecteur de :

- Comprendre les différentes **approches méthodologiques** utilisées dans le développement d'un logiciel.
- Identifier les **étapes clés** de chaque modèle (cascade, prototypage, incrémental, itératif, etc.).
- Analyser les **avantages et inconvénients** de chaque modèle selon le contexte du projet.
- Savoir **choisir le modèle le plus adapté** en fonction des besoins, des contraintes et des objectifs d'un projet informatique.

L'objectif final est de doter l'apprenant des **connaissances nécessaires** pour orienter ou participer activement à la gestion d'un projet logiciel, en adoptant la méthode la plus pertinente.

CHAPITRE VI : LES MODELES DE DEVELOPPEMENT D'UN LOGICIEL

I. INTRODUCTION

II. LE CYCLE DE VIE EN " CASCADE " (Waterfall) :

Le modèle en cascade, considéré comme l'un des piliers du développement de projets, demeure une approche largement adoptée dans de nombreux secteurs. Son application s'étend à une diversité de projets, allant du développement logiciel à la gestion de projets d'ingénierie.

a. Qu'est-ce que c'est le modèle en cascade ?

Le modèle en cascade, connu sous le nom de Waterfall en anglais, est une approche linéaire et séquentielle des différentes phases et activités d'un projet nécessaires à la livraison du ou des livrables. Cette méthode, souvent associée à la gestion de projet en cascade, implique une progression étape par étape, où chaque phase de gestion de projet doit être complétée avant de passer à la suivante voire figure 21.

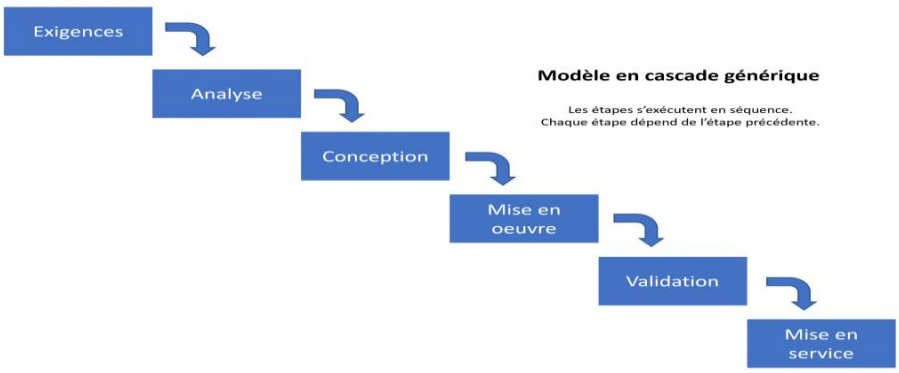


Figure 23 Modèle en cascade générique

Il a été présenté pour la première fois par Herbert D. Benington en 1956 lors d'un Symposium sur les méthodes de programmation avancées.

La méthode en cascade est plus particulièrement utilisée aujourd'hui pour les projets dont la réalisation de chaque étape dépend de l'étape précédente. Et particulièrement dans la construction.

Exemple

Il est en effet indispensable d'avoir construit les fondations d'une maison avant de pouvoir commencer la construction des murs.

Et le toit ne sera fabriqué que sur des murs solides.

Ainsi de suite.

b. Etapes du modèle Waterfall

Le modèle Waterfall comporte 6 étapes : (pour plus de détaille voir chapitre 3)

- L'expression des besoins
- Analyse des besoins
- Conception du projet
- Réalisation du projet,
- Contrôle et validation
- Mise en service

Remarque

Entre chaque phase se trouvent des "portes" ou points de contrôle pour la qualité, où sont évalués l'état du projet, les coûts, les délais, les risques, le contrôle de qualité et l'implication des équipes.

Cette méthode **Stage Gate** permet de corriger certaines erreurs en décidant de revenir à l'étape précédente si cela est nécessaire pour le bien du projet voire figure 22.

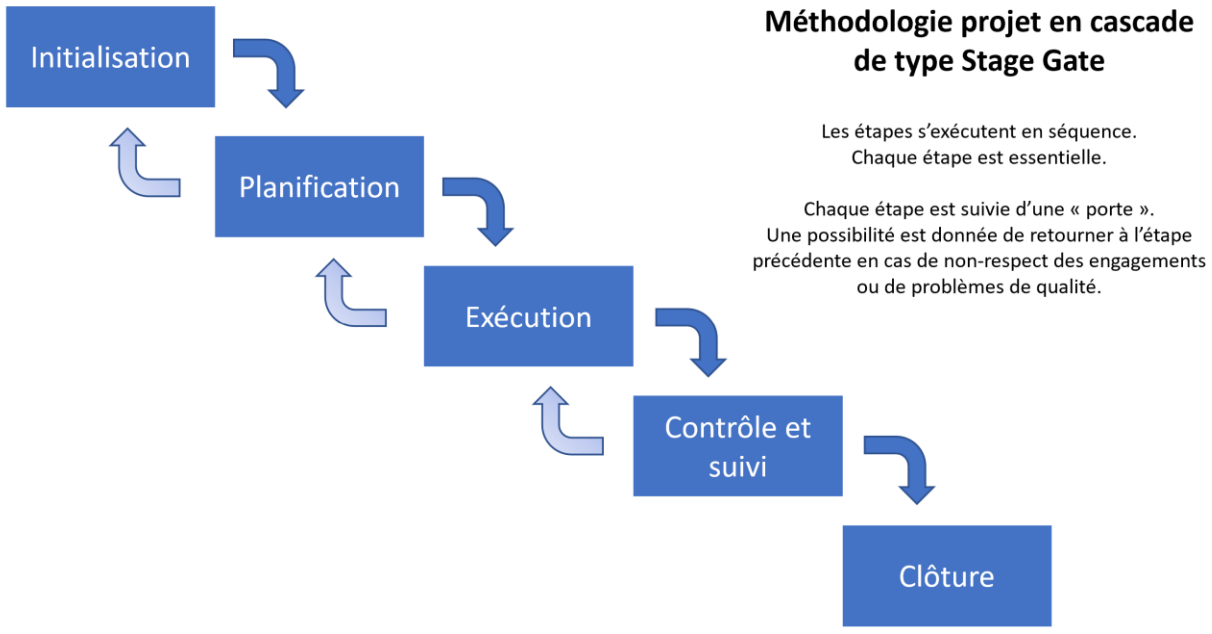


Figure 24 Modèle en cascade Stage Gate

c. Avantages

- Le planning est établi à l'avance et le chef du projet sait précisément ce qui va lui être livré et quand il pourra en prendre livraison

d. Inconvénients

- Modèle trop séquentiel c.à.d. dure trop longtemps
- Validation trop tardive (remise en question coûteuse des phases précédentes)
- Sensibilité à l'arrivée de nouvelles exigences (refaire toutes les étapes)
- Bien adapté lorsque les besoins sont clairement identifiés et stables

III. LE CYCLE DE VIE EN " V "

a. Qu'est-ce que le modèle en V

Le cycle en V (« V model » ou « Vee model » en anglais) est un modèle d'organisation des activités d'un projet qui se caractérise par un flux d'activité descendant qui détaille le produit jusqu'à sa réalisation, et un flux ascendant, qui assemble le produit en vérifiant sa qualité. Ce modèle est issu du modèle en cascade dont il reprend l'approche séquentielle et linéaire de phases.

Il l'enrichit cependant d'activités d'intégration de système à partir de composants plus élémentaires, et il met en regard chaque phase de production successive avec sa phase de validation correspondante, lui conférant ainsi la forme d'un V voire figure 23.

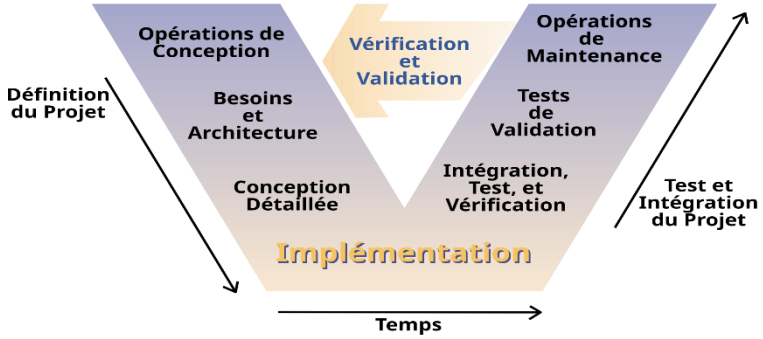


Figure 25 Modèle en V

b. Etape du modèle en V

Les étapes du modèle sont alors :

- Exigences : les exigences font l'objet d'une expression des besoins.;
- Analyse : il s'agit, à partir de l'expression de besoin, d'établir le cahier des charges fonctionnel ou les spécifications fonctionnelles;
- Conception générale, aussi appelé conception architecturale ou conception préliminaire: il s'agit de concevoir le système qui doit répondre aux exigences et de définir son architecture, et en particulier les différents composants nécessaires;
- Conception détaillée : il s'agit de concevoir chaque composant, et la manière dont ils contribuent à la réponse aux besoins ;
- Mise en œuvre: il s'agit de réaliser chaque composant nécessaire. Pour les composants et systèmes logiciels, l'activité est essentiellement le codage;
- Test unitaire: il s'agit de vérifier le bon fonctionnement et la conformité de chaque composant à sa conception détaillée ;
- Intégration et test d'intégration: il s'agit d'assembler le système à partir de tous ses composants, et de vérifier que le système dans son ensemble fonctionne conformément à sa conception générale ;
- Test système (anciennement « tests fonctionnels ») : vérification que le système est conforme aux exigences ;
- Test d'acceptation (également appelés « recette » dans le contexte de la sous-traitance) : validation du système par rapport à sa conformité aux besoins exprimés.

Une des différences entre la recette usine et la recette finale est essentiellement contractuelle. Aussi, il n'est pas rare que le maître d'ouvrage délègue la validation auprès d'un organisme de validation, cet organisme étant bien souvent constitué d'experts afin de diminuer les erreurs de validation.

Au niveau de la gestion de projet, les différentes étapes peuvent donner lieu à des phases distinctes sur l'axe horizontal du temps. Plusieurs étapes successives peuvent toutefois être regroupées au sein d'une phase plus large

c. Avantages

- La préparation des dernières phases (validation et vérification) par les premières (construction logiciel), permet d'éviter d'énoncer une propriété qu'il est impossible de vérifier après la réalisation
- Chaque livrable doit être testable

d. Inconvénients

- Le logiciel est utilisé très tard, il faut attendre longtemps pour savoir si on a construit le bon logiciel
- Ne contient pas d'activités d'analyse de risque
- Idéal quand les besoins sont bien connus, « Quand l'analyse et la conception sont claires »

IV. LE CYCLE DE VIE EN SPIRALE

a. Qu'est-ce que le modèle en spirale

Le modèle en spirale (spiral model) est un modèle de Cycle de développement logiciel qui reprend les différentes étapes du cycle en V. Par l'implémentation de versions successives, le cycle recommence en proposant un produit de plus en plus complet et dur. Le cycle en spirale met cependant plus l'accent sur la gestion des risques que le cycle en V voire figure 24.

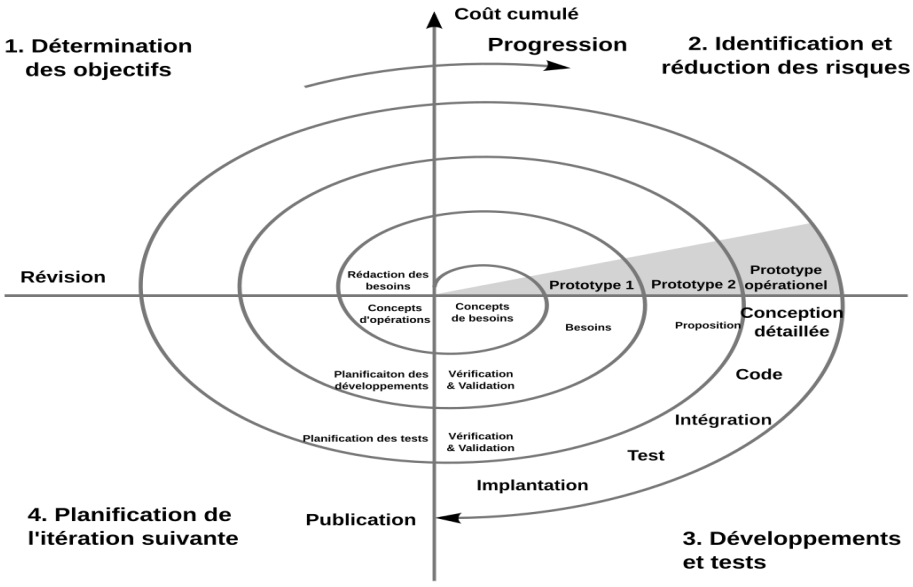


Figure 26 Modèle en spirale

Le modèle en spirale a été défini par Barry Boehm en 1988 dans son article "A Spiral Model of Software Development and Enhancement".

b. Etape du modèle en spirale

- détermination des objectifs, des alternatives et des contraintes ;
- analyse des risques, évaluation des alternatives ;
- développement et vérification de la solution retenue ;
- revue des résultats et vérification du cycle suivant.

Boehm identifie des visions erronées de son modèle provenant de simplifications excessives. Selon lui, les principales erreurs à éviter seraient les suivantes :

- Considérer la spirale comme une suite d'incrémentation en cascade
- Tous les éléments du projet suivent une seule séquence en spirale
- Tous les éléments du diagramme doivent être faits dans l'ordre indiqué sans possibilité de retour en arrière

c. Avantages

- Permet d'établir le modèle de développement le plus adéquat en regard du risque
- Tous les autres modèles de développement constituent une variante de la spirale

d. Inconvénients

- A besoin de compétences dans l'évaluation approfondie des incertitudes et des risques associés au projet et leur réduction
- Évaluer les risques impliqués dans le projet peut prendre jusqu'à le coût et il peut être plus élevé que le coût de la construction du système.

V. LE MODELE PAR PROTOTYPAGE

a. Qu'est-ce que le modèle par prototypage

Le modèle par prototypage est une méthode de développement logiciel qui consiste à créer un prototype, c'est-à-dire une version simplifiée et incomplète du logiciel final, dans le but de

mieux comprendre les besoins du client et de valider les fonctionnalités avant de développer le produit final voire figure 25.

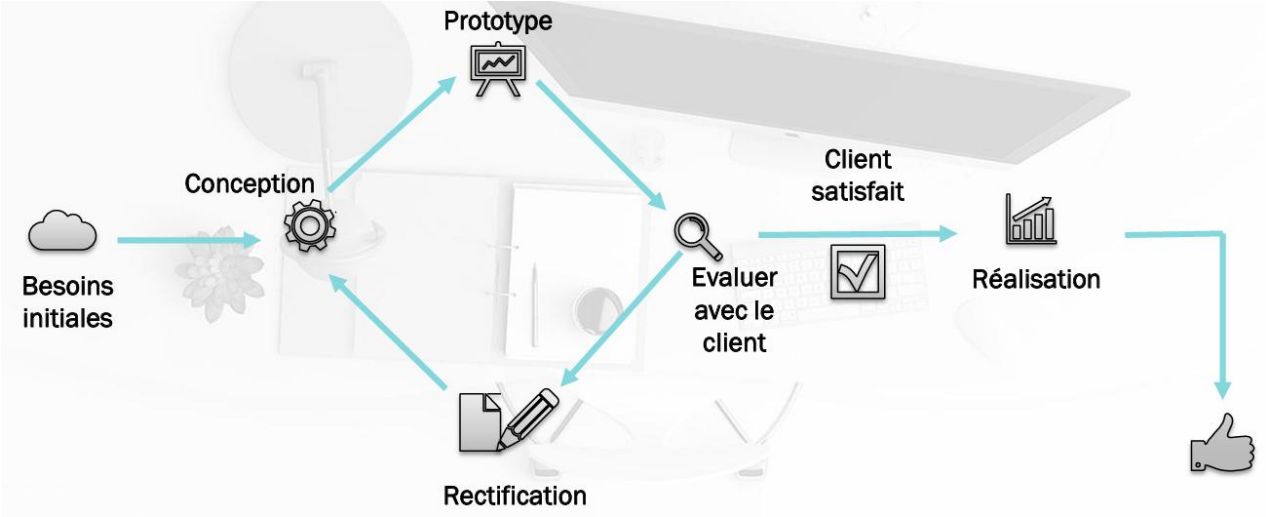


Figure 27 Modèle de prototypage

b. Principe du modèle par prototypage

- Recueil des besoins initiaux : On commence par identifier les besoins généraux du client (même s'ils sont encore flous).
- Conception rapide du prototype : Une maquette ou un système partiel est développé rapidement.
- Évaluation du prototype par le client : Le client teste le prototype et donne son retour.
- Amélioration itérative : Le prototype est modifié en fonction des retours. Ce cycle peut être répété plusieurs fois.
- Développement du produit final : Une fois les besoins bien compris et validés, on développe la version finale du logiciel.

c. Type de prototype

- **Prototypage jetable** est un squelette du logiciel qui **n'est créé que dans un but** et dans une phase particulière de développement, si ce prototype est gardé ; alors il s'appelle prototypage évolutif
- **Prototypage évolutif** comme le projet se fait sur plusieurs itérations (1. Les développeurs construisent un prototype selon les attentes du client. 2. Le prototype est évalué par le

client ♣ Le client donne son feedback. 3. Les développeurs adaptent le prototype selon les feedbacks et les nouvelles exigences client. 4. Quand le prototype satisfait le client, le code est normalisé selon les standards et les bonnes pratiques.

d. Avantages

- Implication active du client, le développeur apprend directement du client
- S'adapter rapidement aux changements des besoins

e. Inconvénients

- Difficile d'établir un planning
- Le processus peut ne jamais s'arrêter
- Idéal quand les besoins sont instables et/ou nécessitent des clarifications, conseillé pour de très petits projets impliquant très peu de personnes

VI. LE MODELE PAR INCREMENT

a. Qu'est-ce que le modèle par incrément

Le modèle par incrément est une méthode de développement logiciel où le produit est construit progressivement, par étapes successives (appelées incréments), jusqu'à obtenir la version finale.

b. Principe du modèle par incrément

- Définition des exigences globales.
- Découpage du projet en plusieurs modules ou fonctionnalités.
- Développement du premier incrément : une partie fonctionnelle du logiciel est conçue, développée et livrée.
- Livraison et test de chaque incrément : à chaque étape, un nouveau morceau du logiciel est ajouté au précédent.
- Répétition du processus jusqu'au produit final.

Chaque incrément est testé, validé, et intégré avec les parties précédentes voire figure 26.

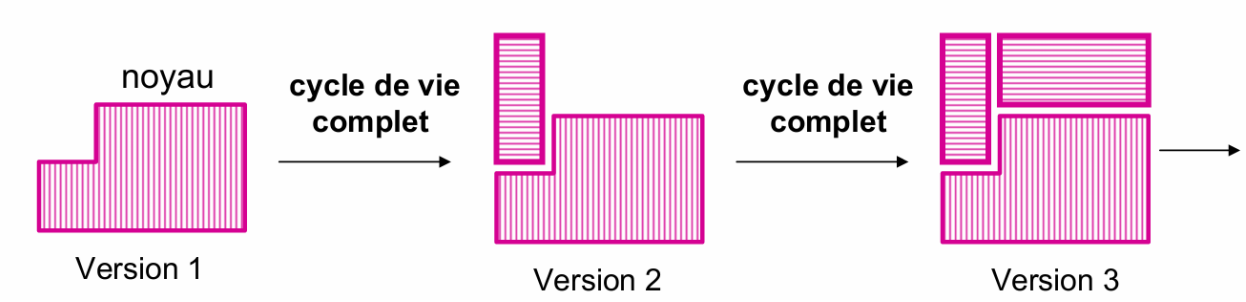


Figure 28 Modèle par incrément

c. Avantages du modèle par incrément

- Livraison rapide des premières fonctionnalités Flexibilité car Il est possible d'ajouter ou modifier des fonctionnalités entre les incréments.
- Réduction des risques où Les erreurs ou problèmes sont détectés plus tôt, ce qui facilite les corrections.
- Retour d'expérience régulier l'avis du client à chaque étape améliore la qualité finale du produit.
- Meilleure gestion des priorités ; On peut commencer par les fonctionnalités les plus importantes ou urgentes.

d. Inconvénients du modèle par incrément

- Planification complexe ce qui nécessite une bonne organisation pour gérer l'ordre des incréments et leurs dépendances.
- Problèmes d'intégration car l'"ajout progressif de nouvelles parties peut créer des conflits ou bugs s'ils ne sont pas bien intégrés.
- Besoins initiaux parfois incomplets dû à une mauvaise vision d'ensemble dès le départ peut entraîner une architecture mal adaptée.
- Coût de développement potentiellement plus élevé dû aux tests répétés, intégrations fréquentes et ajustements peuvent allonger les délais.

VII. MODELE ITERATIF

a. Qu'est-ce que le modèle itératif

Le modèle itératif est une méthode de développement logiciel où le projet est réalisé en plusieurs cycles ou itérations. À chaque itération, on développe une version améliorée et plus complète du logiciel, jusqu'à atteindre le produit final voire figure 27.

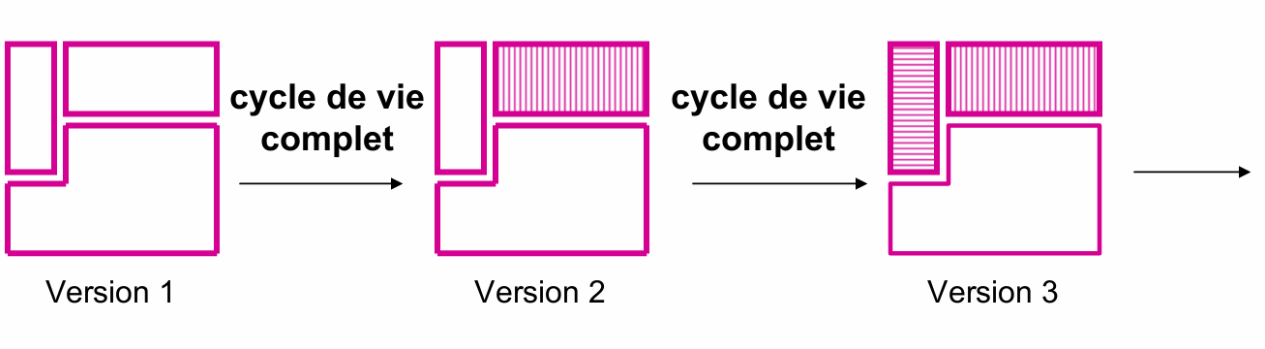


Figure 29 Modèle itératif

b. Principe du modèle itératif

- Planification initiale : on identifie les exigences principales du projet.
  - Première itération :
  - Analyse des besoins partiels
  - Conception, développement, tests
- Livraison d'une version de base
- Feedback client
- Itération suivante : on améliore ou complète le logiciel en se basant sur les retours.
- Répétition du cycle jusqu'à satisfaction du client ou atteinte des objectifs.

c. Avantages

Formation précoce des utilisateurs.

Réponse rapide possible.

Création précoce de nouveaux marchés pour nouvelles fonctionnalités.

Focus sur nouveau domaine d'expertise à chaque étape (version).

Détection précoce des problèmes imprévus (correction immédiate du système en développement).

d. Inconvénients

Risque de la remise en cause du noyau (fonctionnalités de base) au cours du développement

VIII. CONCLUSION

Les modèles de développement logiciel jouent un rôle essentiel dans la structuration et la réussite des projets informatiques. Chaque modèle — en cascade, par prototypage, par

incrément ou itératif — présente des avantages et des limites qui le rendent plus ou moins adapté selon le contexte du projet, le niveau de clarté des besoins, les ressources disponibles, et le niveau d'implication du client.

Le choix du modèle dépend donc de plusieurs facteurs :

la taille et la complexité du projet,

la stabilité des exigences,

les délais à respecter,

et la capacité de collaboration avec les utilisateurs finaux.

Dans un environnement en constante évolution, il devient également courant d'adopter des approches hybrides ou agiles, plus flexibles, permettant de mieux s'adapter aux changements et aux attentes du client. En résumé, il n'existe pas de modèle unique parfait, mais une bonne compréhension de chaque approche permet de faire un choix éclairé pour maximiser les chances de succès d'un projet logiciel.

CHAPITRE V: LA CONCEPTION DU LOGICIEL

Objectif du Chapitre 5:

Ce chapitre a pour objectif de présenter les **principes fondamentaux de la conception d'un logiciel**, en insistant sur les notions de **modularité**, de **rigueur** et de **séparation des problèmes**.

Il vise à permettre au lecteur de :

- Comprendre l'importance d'une **conception structurée** et bien pensée dans le cycle de vie d'un logiciel.
- Savoir **diviser un système complexe en modules cohérents et indépendants** pour en faciliter le développement et la maintenance.
- Appliquer une **démarche rigoureuse** dans le choix des architectures, des modèles et des outils de conception.
- Maîtriser le principe de **séparation des préoccupations**, en isolant les différentes fonctions ou responsabilités d'un système.

À travers ce chapitre, l'apprenant développera une vision claire de la manière dont une **bonne conception conditionne la qualité finale du logiciel**, sa stabilité, sa maintenabilité et sa capacité à évoluer.

-

CHAPITRE V : LA CONCEPTION DU LOGICIEL

I. INTRODUCTION

La phase de conception suit immédiatement la phase d'analyse. Elle est prise en charge par l'équipe de développement. Elle vise à définir une architecture *modulaire* du logiciel ou encore une décomposition en *modules* qui facilitera la maintenance et permettra le développement parallèle par différents programmeurs.

On regroupe en général sous le terme conception deux étapes distinctes :

- **Conception globale** : elle permet de définir les choix fondamentaux de l'architecture du logiciel en liaison avec l'architecture matérielle.

- **Conception détaillée** : elle décrit précisément chaque module, les algorithmes mis en œuvre et les traitements effectués en cas d'erreur. La documentation produite sera très utile en phase de maintenance. La phase de conception détaillée n'est pas traitée dans ce cours. Elle a souvent tendance aujourd'hui à être escamotée car la taille des modules étant réduite il n'est souvent pas nécessaire de décrire précisément les algorithmes mis en jeu.

La phase de conception globale se termine avec la rédaction du **document de conception globale** et du **plan d'intégration** qui ne sera pas traité dans ce chapitre.

II. LES ENJEUX DE LA CONCEPTION DU LOGICIEL

1. LA DESCRIPTION DE LA CONCEPTION

Une conception est utilisée de différentes manières :

- pour produire des implantations ;
- comme moyen de communication entre les concepteurs de différents sous-systèmes ;
- comme référence pour l'entretien du système.
- ...

Dans la pratique la conception du système produit une description (un modèle) du système qui sous forme de spécifications formalisées avec des techniques d'analyse et de spécification, des descriptions en langage naturel ou en langage de description de programmes (pseudo-langage).

Le choix des techniques utilisées dépend de la méthode de conception et de l'adéquation de la technique aux besoins précis (image d'ensemble, description d'algorithmes et de structures de données, etc.)

CHAPITRE V : LA CONCEPTION DU LOGICIEL

2. LES PRINCIPES

Cette partie liste sept principes fondamentaux (proposés par Carlo Ghezzi):

- Rigueur,
- Séparation des problèmes (« separation of concerns »),
- Modularité,
- Abstraction,
- Anticipation du changement,
- Généricité,
- Construction incrémentale.

a. Rigueur

La production de logiciel est une activité créative, mais qui doit se conduire avec une certaine rigueur. Certains opposent parfois créativité et rigueur. Il n'y a pas contradiction : par exemple, le résultat d'une activité de création pure peut être évalué rigoureusement, avec des critères précis.

Le niveau maximum de rigueur est la *formalité*, c'est à dire le cas où les descriptions et les validations s'appuient sur des notations et lois mathématiques. Il n'est pas possible d'être formel tout le temps : il faut bien construire la première description formelle à partir de connaissances non formalisées ! Mais dans certaines circonstances les techniques formelles sont utiles.

b. Séparation des problèmes

C'est une règle de bons sens qui consiste à considérer séparément différents aspects d'un problème afin d'en maîtriser la complexité. C'est un aspect de la stratégie générale du « diviser pour régner ».

Elle prend une multitude de formes :

- Séparation dans le *temps* (les différents aspects sont abordés successivement), avec la notion de cycle de vie du logiciel que nous étudierons en détail,
- Séparation des *qualités* que l'on cherche à optimiser à un stade donné (ex : assurer la correction avant de se préoccuper de l'efficacité),
- Séparations des *'vues'* que l'on peut avoir d'un système (ex : se concentrer sur l'aspect 'flots de données' avant de considérer l'aspect ordonnancement des opérations ou 'flot de contrôle'),
- Séparation du système en *parties* (un noyau, des extensions, …),
- Etc. Les méthodes aident à organiser le travail en guidant dans la séparation et l'ordonnancement des questions à traiter.

c. Modularité

Un *module* est un élément de petite taille (en général un ou quelques sous-programmes) qui sert, par assemblage à la construction de logiciels. Un module doit être cohérent et autonome. Un ensemble de modules (un logiciel ou un assemblage de modules) doit être bien organisé en architecture robuste.

La modularité se fait ressentir dans la conception et dans la programmation, d'où on peut parler de deux sortes de modules : modules de conception et modules de programmation.

Un système est modulaire s'il est composé de *sous-systèmes plus simples*, ou modules. La modularité est une propriété importante de tous les procédés et produits industriels (cf. l'industrie automobile ou le produit et le procédé sont très structurés et modulaires).

La modularité permet de considérer séparément le *contenu* du module et les *relations entre modules* (ce qui rejoint l'idée de séparation des questions). Elle facilite également la *réutilisation* de composants biens délimités.

Un bon découpage modulaire se caractérise par une *forte cohésion* interne des modules (ex : fonctionnelle, temporelle, logique, ...) et un *faible couplage* entre les modules (relations inter modulaires en nombre limité et clairement décrites).

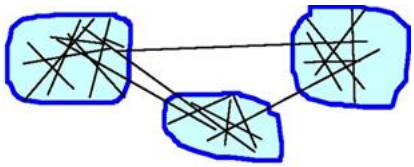


Figure 30 La modularité

Toute l'évolution des langages de programmation vise à rendre plus facile une programmation modulaire, appelée aujourd'hui 'programmation par composants'.

d. Abstraction

L'abstraction consiste à ne considérer que les *aspects jugés importants* d'un système à un moment donné, en faisant abstraction des autres aspects (c'est encore un exemple de séparation des problèmes).

Une même réalité peut souvent être décrite à différents *niveaux d'abstraction*. Par exemple, un circuit électronique peut être décrit par un modèle mathématique très abstrait (équation logique), ou par un assemblage de composants logiques qui font abstraction des détails de réalisation (circuit logique), ou par un plan physique de composants réels au sein d'un circuit

intégré. L'abstraction permet une meilleure maîtrise de la complexité.

e. Anticipation du changement

La caractéristique essentielle du logiciel, par rapport à d'autres produits, est qu'il est presque toujours soumis à des *changements continuels* (corrections d'imperfections et évolutions en fonctions *des besoins qui changent*).

Ceci requiert des efforts particuliers pour *prévoir*, faciliter et gérer ces évolutions inévitables. Il faut par exemple :

- faire en sorte que les changements soient les plus localisés possibles (bonne modularité),
- être capable de gérer les multiples versions des modules et configurations des versions des modules, constituant des versions du produit complet.

f. Généricité

Il est parfois avantageux de remplacer la résolution d'un problème spécifique par la résolution d'un problème plus général. Cette solution générique (paramétrable ou adaptable) pourra être réutilisée plus facilement.

Exemple : plutôt que d'écrire une identification spécifique à un écran particulier, écrire (ou réutiliser) un module générique d'authentification (saisie d'une identification – éventuellement dans une liste – et éventuellement d'un mot de passe).

g. Construction incrémentale

Un procédé incrémental atteint son but par étapes en s'en approchant de plus en plus ; chaque résultat est construit en étendant le précédent.

On peut par exemple réaliser d'abord un noyau des fonctions essentielles et ajouter progressivement les aspects plus secondaires. Ou encore, construire une série de prototypes 'simulant' plus ou moins complètement le système envisagé.

3. Processus de conception de logiciel

La conception d'un logiciel peut être effectuée par un processus de résolution de problème qui, à partir d'une conception initiale, identifie des abstractions de plus en plus raffinées. Le processus se répète jusqu'à ce que l'on soit en mesure de préparer une spécification de la conception de chaque abstraction. À chaque itération de ce processus il faut effectuer les étapes suivantes :

- Étude et compréhension du problème (analyse et spécifications) : examiner le problème sous différents angles.

- Identifier les caractéristiques d'au moins une solution possible. Il est souvent utile (et parfois indispensable) d'évaluer plusieurs solutions.
- Décrire toutes les abstractions utilisées dans la solution. Il est conseillé de créer et de mettre au point une description informelle de la conception, ce qui permet de corriger les erreurs de haut niveau avant de commencer la documentation de la conception.

On peut raffiner description donnée précédemment du processus de conception comme suit :

- Conception de l'architecture : identifier les sous-systèmes et leurs relations.
- Spécification abstraite : pour chaque sous-système produire une spécification abstraite des services qu'il offre et des contraintes auxquelles il est soumis.
- Conception de l'interface : pour chaque sous-système, concevoir et documenter (de manière claire) l'interface avec les autres sous-systèmes.
- Conception des composants de chaque sous-système.
- Conception détaillée des structures de données.
- Conception détaillée des algorithmes.

La conception doit représenter le système à plusieurs niveaux d'abstraction. Chaque activité de conception doit produire une spécification formelle correspondante au niveau d'abstraction, ces spécifications seront donc de plus en plus détaillées et doivent aboutir à la spécification des algorithmes et des structures de données permettant l'implantation du système. Mais, malgré l'apparence séquentielle de ce processus, il n'est pas rare d'entamer une nouvelle phase de conception avant la fin de la précédente pour avoir des retours d'information.

4. Qualité de la conception

Une « bonne » conception se définit en termes de la satisfaction des besoins et des spécifications : les critères de qualités peuvent donc varier énormément d'un système à un autre ou même pour deux implantations différentes d'un même système.

Par la suite on considérera de préférence les facteurs qui facilitent la gestion du produit tout le long de son cycle de vie. Il s'agit essentiellement de l'extensibilité, la réutilisabilité, la compatibilité, la portabilité la vérifiabilité et la facilité d'emploi.

Une bonne structuration participe largement à la production d'un logiciel qui possède ces caractéristiques. Comme dans beaucoup de domaine, cette bonne structuration se base sur la *Modularité*.

CHAPITRE V : LA CONCEPTION DU LOGICIEL

La qualité de la conception d'un logiciel détermine sa **facilité de maintenance, d'évolution, de compréhension** et de **réutilisation**. Une bonne conception repose sur plusieurs principes fondamentaux, dont :

- **La modularité**
- **La réutilisabilité**
- **La testabilité**
- Et surtout : **le faible couplage** et la **forte cohésion**

.

III. CONCLUSION

La phase de conception du logiciel est une étape clé du développement, car elle pose les fondations de la structure du futur système. Une bonne conception repose sur des principes essentiels tels que la modularité, la rigueur méthodologique, et la séparation des problèmes.

La modularité permet de diviser le système en composants indépendants, facilitant ainsi la compréhension, la maintenance et l'évolution du logiciel. Elle favorise également la réutilisation du code et l'isolation des erreurs.

La rigueur dans la conception garantit la cohérence entre les besoins spécifiés et la solution technique proposée. Elle implique une documentation claire, des choix réfléchis, et une architecture logique qui anticipe les futures évolutions.

Enfin, la séparation des problèmes consiste à analyser et traiter chaque partie du système indépendamment, selon sa propre logique fonctionnelle. Cela réduit la complexité et permet d'aborder les défis techniques de manière ciblée et efficace.

En appliquant ces principes, la conception devient un outil de maîtrise du projet logiciel, capable de transformer des besoins complexes en une solution fiable, évolutive et performante.

CONCLUSION GENERALE

CONCLUSION GENERALE

En guise de conclusion, la conduite et/ou la gestion de projets informatiques est un ensemble de techniques qui permettent d'identifier, de planifier et de piloter le développement d'un produit logiciel. Toutefois l'évolution actuelle à fait susciter l'aspect managériale afin d'avoir une plus grande valeur ajoutée qui permet la conduite du projet vers la réussite. Ces techniques et outils ne peuvent fonctionner pleinement que dans le cadre d'une gestion dimensionnée par projet.

En effet, nous avons voulu montrer, dans les différents chapitres de ce cours, qu'il existe des techniques permettant de maîtriser le management hors hiérarchie qu'implique une organisation par projet répondant aux nécessités du troisième millénaire. Le tout étant d'accepter d'y consacrer les moyens voulus en fonction de l'ambition du projet. Parce qu'il ne reproduit pas de modèle mais crée des modèles nouveaux, la conduite par projet est l'outil du passage à une nouvelle forme de progrès: développer à la fois le potentiel des hommes et celui des métiers de l'entreprise.

Plus qu'une méthode d'organisation, la conduite de projets informatiques représente une nouvelle approche culturelle pour les entreprises informatiques ; une approche fondée sur la culture de la transversalité. En ce sens, la conduite de projets va donc bien plus loin que le fait de faire travailler ensemble des gents venants de différents métiers sur un objectif commun. C'est à la fois un outil stratégique de l'entreprise pour répondre aux changements rapides des marchés et un formidable outil de motivation pour ceux qui y participent comme pour les métiers, si tous les apprentissages réalisés sont mémorisés. Le tout est d'avoir envie et de savoir-faire vivre cette transversalité entre les objectifs à court terme des projets et les objectifs à long terme de l'entreprise. Faire vivre un projet est une question de méthodes et d'hommes. C'est aussi, pour les entreprises, une question de volonté.

CONCLUSION GENERALE

L'auteur de ce polycopie, le Docteur Sourour Maalem, vous remercie infiniment du temps que vous avez et que vous consacrerez à l'exploitation de ce dit ouvrage usité aux différentes méthodes de conduite de projets informatiques.

Bonne lecture et digestion.

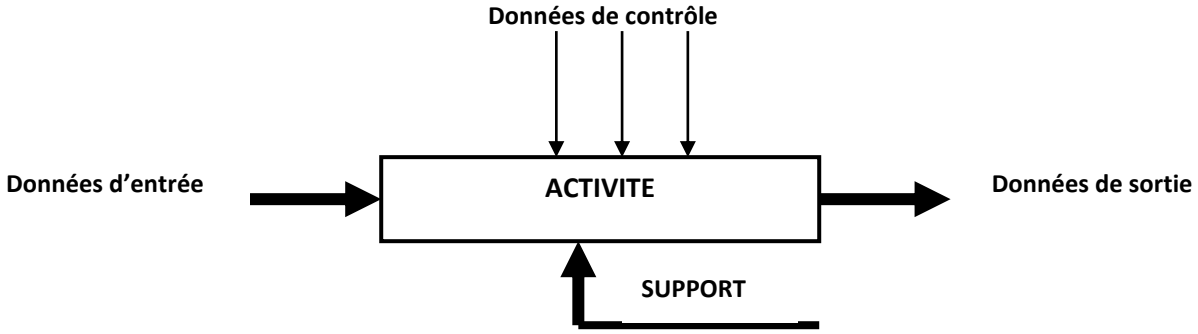
SERIES DE TD

TD1

Analyse Structurée et modélisation des systèmes

Rappels : La conception d'un système nécessite une analyse statique et une analyse dynamique du système .Une démarche utilisée par de nombreuses entreprises consiste à identifier et analyser les fonctions du système par la méthode SADT (Structured Analysis et Design Technique), puis s'intéresser aux données qui se traduit par un outil graphique composé d'actigrammes et de datagrammes .Un diagramme d'activité ou actigramme s'écrit en répondant aux question :

- **SUR QUOI ?**sur les entrées
- **POURQUOI ?**pour donner une valeur ajoutée
- **QUE FAIRE ?**une certaine activité
- **AVEC QUOI ?**en utilisant des moyens
- **COMMENT ?**en respectant certaines contraintes



Exercice 1 : A l'aide des DFD et de Diagramme de structure décrire les systems suivants :

- Compilateur des programmes
- Correcteur d'orthographe

Exercice 2 : A l'aide d'un diagramme d'activité (actigramme), décrire la fonction globale des systèmes suivants :

- Une machine-outil (fraiseuse, perceuse…)
- Un cyclomoteur (moto)
- Un Capteur solaire
- Une station de lavage automatique de véhicules
- Un avion

- Un système de production (Usine de fabrication de pièces)

Exercice 3 : A laide d'un diagramme de données (datagramme), décrire les données suivants :

- Une piéce détachée
- L'énergie électrique

TD2

SADT & GRAFCET

Exercice 1 : (Station automatique de lavage de voiture)

Description :

Une station automatisée de lavage de voitures comprend : une aire (surface) de lavage, et un portique mobile. Le portique mobile se déplace sur des rails entre deux butées Avant et Arrière. Sur ce portique se trouve : deux brosses verticales B1 et B2, une brosse horizontale B3, un ensemble de jets de produits lessiviels et un pupitre de commande qui comprend un monnayeur dans lequel le client introduit le jeton qu'il a acheté à la caisse.

Fonctionnement :

Après introduction du jeton par le client dans le monnayeur :

- Les brosses tournent et le portique avance ; B1 et B2 se rapprochent au maximum, B3 descend et il y a jets de produits lessiviels.
- Le portique continue à avancer jusqu'à la butée Avant.
- Le portique recule et les brosses changent de sens de rotation avec jets de produits lessiviels.
- Le portique continu à reculer jusqu'à la butée Arrière.
- Les brosses B1 et B2 s'écartent au maximum et B3 est relevée.
- Le cycle de lavage de la voiture est terminé.

Questions :

1. Décrire la fonction globale du système à l'aide d'un diagramme A-0
2. Décrire le fonctionnement du système à l'aide d'un Grafcet.

Exercice 2 : (Problème de l'écluse)

SERIES DE TD

Un cours d'eau est rendu navigable par la présence dune écluse. Nous considérons que les bateaux vont de **A** vers **B** en passant par un SAS. A l'arrivée du bateau, le niveau du SAS est égal à celui de **B**. le bateau situé en **A** désirant parvenir en **B** doit franchir le SAS. Le fonctionnement du SAS peut être résumé comme suit :

- A l'arrivée du bateau au point **A**, il y a ouverture de vannes du SAS en **A** pour que le niveau du SAS soit rendu 2gal au niveau de **A**. ensuite, il y a ouverture des portes en **A** et passage du bateau dans le SAS.
- Après la fermeture des portes en **A**, les vannes du SAS en **B** sont ouvertes pour que le niveau du SAS soit rendu égal au niveau de **B**. Ensuite, il y a ouverture des portes en **B** et passage du bateau en **B.**
- Fermeture des portes en **B** et le cycle est terminé.

Questions :

3. Décrire la fonction globale du système à l'aide d'un diagramme A-0
4. Décrire le fonctionnement du système à l'aide d'un Grafcet.

SERIES DE TD

TD3

Réseaux de pétri marqués

Exercice 1 : (Station automatique de lavage de voiture)

Description :

Une station automatisée de lavage de voitures comprend : une aire (surface) de lavage, et un portique mobile. Le portique mobile se déplace sur des rails entre deux butées Avant et Arrière. Sur ce portique se trouve : deux brosses verticales B1 et B2, une brosse horizontale B3, un ensemble de jets de produits lessiviels et un pupitre de commande qui comprend un monnayeur dans lequel le client introduit le jeton qu'il a acheté à la caisse.

Fonctionnement :

Après introduction du jeton par le client dans le monnayeur :

- Les brosses tournent et le portique avance ; B1 et B2 se rapprochent au maximum, B3 descend et il y a jets de produits lessiviels.
- Le portique continue à avancer jusqu'à la butée Avant.
- Le portique recule et les brosses changent de sens de rotation avec jets de produits lessiviels.
- Le portique continu à reculer jusqu'à la butée Arrière.
- Les brosses B1 et B2 s'écartent au maximum et B3 est relevée.
- Le cycle de lavage de la voiture est terminé.

Questions :

Décrire le fonctionnement de ce système à l'aide d'un réseau de Pétrie marqué.

Problème 2 : (Problème de l'écluse)

Un cours d'eau est rendu navigable par la présence dune écluse. Nous considérons que les bateaux vont de **A** vers **B** en passant par un **SAS**. A l'arrivée du bateau, le niveau du SAS **peut être égal à celui de A ou de B**. le bateau situé en **A** désirant parvenir en **B** doit franchir le **SAS**. C'est l'ouverture des vannes en **A** qui permet la régularisation de ce niveau. Lorsque le niveau du **SAS** est égal à celui de **A**, l'ouverture de porte en **A** permet le passage du bateau. Lorsque le bateau est entièrement dans le **SAS**, la porte est refermée. Le bateau tente alors de passer en **B**.

pour cela, les niveaux de **SAS** et de **B** sont régularisés de la même manière en ouvrant les vannes en **B**. lorsque le niveau du SAS aura atteint celui de **B**, l'ouverture de la porte **B** permettra le passage du bateau en **B**. décrire le fonctionnement de ce système à l'aide d'un réseau de pétri marqué.

Problème 3 : (Files d'attente)

Pour retirer de l'argent dans une banque, on passe généralement par deux guichets. Un premier guichet de type **A** permet de vérifier le chèque et de débiter le compte de l'intéressé. Un seconde guichet de type **B** permet, une fois le chèque validé, de délivrer la somme demandée au client. On suppose que l'on dispose de deux guichets de type **A** (**A1** et **A2**) et un seul guichet de type **B**. pour être servi, le client doit s'adresser, en premier lieu, au guichet **A1** ou **A2** selon la disponibilité. Ensuit, il doit passer au guichet **B** pour retirer son argent.

Décrire le fonctionnement d'un tel système par un réseau de Pétrie marqué.

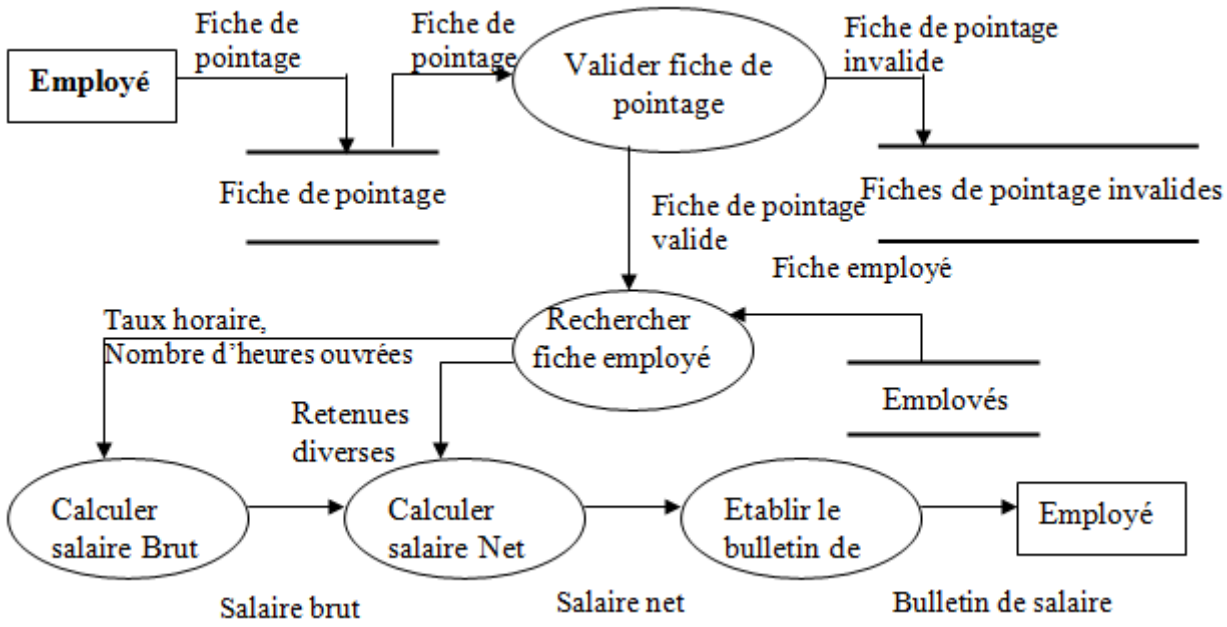
SERIES DE TD

TD4

(Méthode cartésienne : SASD)

Exercice 1 :

On considère le DFD suivant représentant le graphe de fonctionnement d'un logiciel de paie.



Question : Expliquer brièvement à travers le graph. ci-dessus le fonctionnement de ce logiciel de paie (interprétation de DFD).En déduire un Diagramme de Structure représentant la structure en termes de modules de ce logiciel.

Exercice 2 :

Le texte à traiter est un télégramme terminé par «FINTEL». Chaque groupe de mot est terminé par le mot « stop ». On veut avoir le nombre de mots facturables dans tout le texte. Un mot est facturable s'il est différent de FINTEL et STOP.

Question : établir le DFD et en déduire un diagramme de structure représentant la solution du problème en termes de modules.

SERIES DE TD

TD 5

Exercice 1: *Diagramme de flots de données*

L'ordinateur de bord d'une automobile doit réaliser les fonctions suivantes :

· convertir les signaux de rotation des roues en valeurs numériques (en signaux par seconde),

· afficher à l'aide de diodes électro-luminescentes la vitesse instantanée (conversion en tours/mn puis en km/h),

· avertir par un signal sonore si la vitesse maximum (stockée) est dépassée,

· signaler par une flèche lumineuse dessinée à l'aide des diodes le sens de l'accélération (différence entre deux mesures de signaux par seconde).

- Dessiner le DfD correspondant.

Exercice 2 : *Location de cassettes (DfD)*

On utilise les diagrammes de flots de données pour spécifier les fonctionnalités d'une application de location de cassettes vidéo.

Le cahier des charges précise que :

- le client peut
  - o louer et rapporter des cassettes; les locations sont enregistrées ainsi que les clients
- le gérant peut
  - o ajouter/supprimer des cassettes au catalogue
  - o changer les prix des cassettes au tarif
- l'application
  - o calcule le prix d'une location selon le tarif des cassettes empruntées et la durée de l'emprunt
  - o génère des états de caisse en fin de journée.

1. Dessiner le diagramme de contexte.

2. Raffiner ce diagramme en faisant apparaître une fonction de gestion des locations de cassettes, entourée d'autres fonctions, flots et stockages.
3. Raffiner à un deuxième niveau la fonction de gestion des locations de cassettes.

Exercice 3 : *Diagrammes entités associations.*

Gestion de séjours de vacances. Après inventaire, les principaux concepts à représenter sont : les clients, les prestataires, les hébergements (offerts par les prestataires), les stations (où se situent les hébergements), les propositions, les réservations, les demandes en attente, les types de prestataires (souhaités dans les demandes en attente), les types d'hébergement (souhaités dans les demandes en attente), les activités (possibles dans les stations), les services (offerts par les prestataires).

1. Proposez un diagramme entités relations avec des cardinalités réalistes. Donnez les principaux attributs.

SERIES DE TD

TD6

Exercice 1 :

A propos de la difficulté de développer des systèmes complexes. Lire le communiqué officiel expliquant la désintégration du premier exemplaire de la fusée Ariane 5, paru sur Internet. Analyser la cascade d'erreur qui a été commise.

« Communiqué de presse conjoint ESA-CNES Paris, le 19 juillet 1996 »

« Rapport de la Commission d'enquête Ariane 501 Echec du vol Ariane 5O1 -Président de la Commission: Professeur J.-L LIONS »

Avant-propos

1. L'échec
    1. Description générale
    2. Informations disponibles
    3. Récupération des débris
    4. Autres anomalies observées sans rapport avec l'accident
2. Analyse de l'échec
    1. Séquence des événements techniques
    2. Commentaires du scénario de défaillance
    3. Procédures d'essai et de qualification
    4. Autres faiblesses éventuelles des systèmes incriminés
3. Conclusions
    1. Résultats de l'enquête
    2. Cause de l'accident
4. Recommandations

SERIES DE TD

Exercice 1.2

A propos de la difficulté de spécifier précisément un besoin fonctionnel.

La spécification de la règle de notation à un examen est la suivante : « L'examen est un ensemble de 20 questions à réponses multiples. Chaque bonne réponse à une question rapporte 1 point. Chaque mauvaise réponse fait perdre 1/3 de point. Chaque question sans réponse donne 0 point. »

- Pensez vous que cette spécification est claire ?
- Pour le vérifier, calculez chacun la note des 3 étudiants suivants :

	Réponses correctes	incorrectes	sans	doubles réponses
Duchnock	10	5	5	
Dutif	4	1	6	
Dudule	10	3	4	3 (1 juste, 1 fausse)

- Recensez les résultats possibles.
- Donnez une spécification plus précise.

SERIES DE TD

TD7

Exercice 1 : *Diagramme de contexte et diagramme de flots de données (DfD).*

On considère la gestion d'un bureau de location pour plusieurs stations touristiques.

Différents prestataires offrent des locations. Ils peuvent les retirer tant qu'aucune réservation définitive n'est réalisée. Ils touchent 90% du prix de la location.

Les clients adressent des demandes de renseignement. Le bureau y répond par des propositions de location et d'assurance annulation ou une mise en attente. Le client peut alors refuser ou demander une réservation en envoyant des arrhes et en souscrivant éventuellement l'assurance annulation. Si la location choisie est encore libre, elle est réservée, sinon la demande est mise en attente. Après un délai de 8 jours, la réservation est confirmée de manière définitive. En cas d'annulation après ces 8 jours, un pourcentage est dû par le client sauf s'il a souscrit l'assurance annulation. L'annulation en cas de mise en liste d'attente est toujours possible sans frais.

La facture est envoyée avant le début du séjour. Un rappel suit en cas de non paiement. En cas de nouveau défaut de paiement le dossier est transmis au contentieux.

1. Dessiner le diagramme de contexte et décomposez le en un premier niveau de DFD.
2. Raffiner la gestion des réservations en un diagramme de flots de données plus détaillé.

Exercice 2 : *Réseau de Petri(RDP).*

On veut modéliser la gestion des cabines et des paniers dans une piscine. A l'entrée, une personne qui a trouvé une cabine libre se change en posant ses vêtements dans la cabine puis demande un panier qu'elle remplit pour libérer la cabine. Après la baignade, la personne rentre dans une cabine avec son panier, le vide, libère le panier et se rhabille pour libérer la cabine.

1. Modéliser cette organisation, avec une place représentant le stock des cabines (par exemple 3) et une place modélisant le stock des paniers (par exemple 5).
2. Quel est le maximum de baigneurs simultanés ? Montrer que cette organisation risque de conduire à un blocage.
3. Proposer une organisation qui évite la possibilité d'un blocage et la modéliser en RDP.

SERIES DE TD

Exercice 3 : *Diagrammes entités associations.*

Gestion de séjours de vacances. Après inventaire, les principaux concepts à représenter sont : les clients, les prestataires, les hébergements (offerts par les prestataires), les stations (où se situent les hébergements), les propositions, les réservations, les demandes en attente, les types de prestataires (souhaités dans les demandes en attente), les types d'hébergement (souhaités dans les demandes en attente), les activités (possibles dans les stations), les services (offerts par les prestataires).

1. Proposez un diagramme entités relations avec des cardinalités réalistes. Donnez les principaux attributs.

Exercice 4 : *Modélisation de la dynamique (grafcet et réseau de Petri).*

Le système est composé de deux tâches informatiques cycliques T1 et T2 qui se partagent un processeur unique. Les tâches peuvent être soit en attente de processeur, soit en cours d'exécution sur le processeur. L'allocation et la désallocation des tâches au processeur se fait selon une certaine politique que l'on ne cherche pas à décrire.

a) Modéliser les états et les transitions d'état de ce système avec un grafcet. Expliquer la signification des états et des transitions.

b) Modéliser le même système avec un réseau de Petri. Expliquer la signification des places et des transitions. Le processeur sera modélisé par une place.

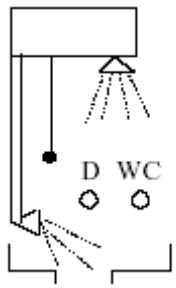
Pendant son exécution T1 peut se mettre en attente d'un événement. Le processeur peut alors être alloué à T2 jusqu'à ce que l'événement arrive. Modifier en conséquence la modélisation.

c) Nouveau grafcet.

d) Nouveau réseau de Petri. L'arrivée de l'événement sera modélisée par une transition.

Exercice 5 : *La 'WCdouche' (réseau de Petri)*

La 'WCdouche' est une douche -WC, pour installations à petit budget... Elle est décrite par le schéma ci-dessous. Elle peut fonctionner comme une douche (arrosage par le haut) ou comme un WC (chasse d'eau vers le bas).



Initialement la WCdouche est dans l'état douche. On sélectionne l'état voulu en appuyant sur un des 2 boutons (D ou WC). Un seul état est possible à chaque instant. Quand on tire sur la ficelle et que l'état WC est choisi, la vanne de la chasse d'eau est ouverte. Quand l'état douche est choisi et que l'on tire sur la ficelle, c'est la vanne de la douche qui est ouverte. On ferme la vanne qui est ouverte en tirant de nouveau sur la ficelle. L'état ne peut être modifié quand une vanne est ouverte.

1. Décrivez par un réseau de Petri le comportement de ce système. Les transitions correspondront aux actions de l'utilisateur (sur les boutons ou la ficelle).

Exercice 6 : *Location de cassettes (DfD)*

On utilise les diagrammes de flots de données pour spécifier les fonctionnalités d'une application de location de cassettes vidéo.

Le cahier des charges précise que :

- le client peut
    - o louer et rapporter des cassettes; les locations sont enregistrées ainsi que les clients
- le gérant peut
    - o ajouter/supprimer des cassettes au catalogue
    - o changer les prix des cassettes au tarif
- l'application
    - o calcule le prix d'une location selon le tarif des cassettes empruntées et la durée de l'emprunt

o génère des états de caisse en fin de journée.

1. Dessiner le diagramme de contexte.
2. Raffiner ce diagramme en faisant apparaître une fonction de gestion des locations de cassettes, entourée d'autres fonctions, flots et stockages.
3. Raffiner à un deuxième niveau la fonction de gestion des locations de cassettes.

TD8

Méthode cartésienne : SASD

Exercice 1 : on voudra écrire des programme ADA permettant de :

1. lire une phrase, l'inverser et l'affiche à l'écran.
2. déterminer le mode d'une série de nombres entiers.
3. déterminer la médiane d'une série de nombres.
4. déterminer les solutions d'une équation du second degré.
5. vérifier si un mot donné est palindrome.
6. calculer la variance et l'écart type d'une série de nombres.

Pour chacun des cas, établir le DFD et en déduire un diagramme de structure correspondant.

Exercice 2 :

Une pharmacie reçoit des prescriptions délivrées par des médecins et doit fournir des médicaments aux malades. La pharmacie doit tenir à jour la liste de toutes les prescriptions, de tous les médicaments disponibles, ainsi que le compte financier de la pharmacie. Elle doit aussi faire savoir aux médecins des associations incompatibles des médicaments et tenir un approvisionnement satisfaisant.

Construire un modèle de la pharmacie (DFD+DD) du point de vue du pharmacien avec comme objectif de déterminer les activités automatisables. En déduire un diagramme de structure.

Exercice 3 :

Ecrire un programme ADA permettant de lire deux matrices et de calculer leur produit matriciel. Ce programme doit contenir au moins trois procédures :

- Une procédure de lecture de matrice ;
- Une procédure de calcul du produit matriciel ;
- Une procédure d'affichage de matrice.

Exercice 4 :

Ecrire un programme ADA permettant de lire deux matrices carrées N × N et de déterminer les points **cols**. Un point M(i,j) de la matrice est dit **col** s'il est le minimum de la ligne i et le maximum dans la colonne j ou inversement.

TD 9

Exercice 1. *Test boîte blanche.* Soit le programme suivant :

```
lire(x)
lire(y)
z = 0
signe = 1
si x < 0 alors
        signe = -1
        x = - x
finsi
si y < 0 alors
        signe = - signe
        y = - y
finsi
tant que x >= y faire
        x = x - y
        z = z + 1
fin
z = signe * z
```

a) Dessiner le graphe de contrôle associé à ce programme en numérotant ses nœuds.

b) Par quelle suite de nœuds faut-il passer pour satisfaire le critère de couverture des instructions ? Donner un jeu d'essai minimum qui satisfasse ce critère.

c) Par quelle suite de nœuds faut-il passer pour satisfaire le critère de couverture des arcs ? Donner un jeu d'essai minimum qui satisfasse ce critère.

d) on appelle ***critère de couverture des i-chemins***, le critère qui garantit que l'on passe sur tous les chemins possibles en répétant de 0 à i fois chaque boucle. Par quelle suite de nœuds faut-il passer pour satisfaire le critère de couverture des 1- chemins? Donner un jeu d'essai minimum qui satisfasse ce critère.

Exercice 2. *Blanc et noir*

a*. Test de type "boîte blanche"*

Donnez 3 jeux d'essai satisfaisant les critères de couverture des instructions, des arcs et des chemins pour l'extrait de code suivant :

if (x > 10) then a = a + 1; endif

if (x % 2 = 0) then b = b + 1; endif (où x % 2 donne le reste de la division entière de x par 2).

b*. Test de type "boîte noire"*

On considère une procédure 'triangle' qui reçoit en paramètres 3 réels a, b et c qui sont les longueurs des côtés d'un triangle. La procédure retourne comme résultat un code 0 si le triangle défini par a, b et c est invalide, 1 si le triangle est équilatéral, 2 si le triangle est isocèle et 3 pour un triangle valide quelconque (ni isocèle, ni équilatéral).

Donnez un jeu d'essai *exhaustif* pour cette procédure testant tous les cas de figure en *distinguant* les 3 entrées a, b et c.

Exercice 3. *Tests boîte blanche*

Soit le programme de comparaison de chaînes de caractères suivant exprimé en pseudo code :

1. equal : un booléen
2. string1, string2 : deux chaînes de caractères
3. Lire(string1, string2)
4. si (string1.length = string2.length) alors // sring1.length donne la longueur de la chaîne string1
5. i <- 1 // la flèche correspond à une affectation = teste une égalité
6. tant que i <= string1.length et string1.character[i] = string2.character[i]
// string1.character[i] donne le ième caractère de la chaîne string1
7. i <- i + 1

8. si i = string1.length + 1 alors // le premier caractère a l'indice 1

9. Afficher(« chaînes égales »)

Donner un jeu d'essai couvrant tous les chemins possibles ; on suppose que la plus petite chaîne à une taille <= 2 (0 ou 1 ou 2). Pour y parvenir, dessiner le graphe de contrôle en numérotant les nœuds du graphe puis lister tous les chemins avec la suite des numéros.

REFERENCES BIBLIOGRAPHIQUE

REFERENCES BIBLIOGRAPHIQUE

Bibliographie

[1]. Gaudel M.C. « Précis de génie logiciel », éditions Dunod

[2]. Meyer B. « Conception et programmation orientées objet », éditions Eyrolles, 2000

[3]. Satzinger, Jackson, Burd, Simond et Villeneuve. « Analyse et conception de systèmes d'information », éditions Reynald Goulet, 2e édition, 2003.

[4]. Introduction to Software Engineering, Ronald J. Leach, CRC Press, Taylor & Francis Group, 2016 https://1library.net/document/rz3r50mz-introduction-to-software-engineering-pdf.html

[5]. Génie logiciel, principes, méthodes et techniques, Presses Polytechniques et Universitaires Romande, 1996, https://www.leslibraires.fr/livre/596107-genie-logiciel-principes-methodes-et-techniques-alfred-strohmeier-didier-buchs--presses polytechniques-et-universitaires-romandes