الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Oran's Higher Teachers College
AMMOUR Ahmed
المدرسة العليا للأساتذة بوهران
عمور احمد

Département des Sciences Exactes
Spécialité : Informatique

Polycopié de cours et exercices

4 ieme année PES Informatique
Semestre 4 et 5 License Informatique

Génie Logiciel

Proposé par :

Dr. BELAYACHI Naima, Maitre de conférences « A ».

Année universitaire 2025-2026

Dédicace

À mes étudiants ...

Table des matières

Table des figures

Liste des tableaux

Résumé

Ce cours de Génie Logiciel a pour objectif d'initier les étudiants aux principes fondamentaux de la conception et du développement des systèmes et des logiciels. Il met l'accent sur l'approche systémique, les méthodes de conception statique et dynamique, ainsi que sur les concepts clés du cycle de vie du logiciel.

Le présent support aborde les enjeux liés à la qualité logicielle, depuis l'analyse des besoins jusqu'à la maintenance, en passant par les différents modèles de développement tels que les cycles en cascade, en V, en spirale, par prototypage et par incrément. Une attention particulière est accordée à la phase de conception, qu'elle soit orientée fonctions ou structurée, afin de sensibiliser les étudiants à l'importance d'une démarche méthodique avant toute implémentation.

Ce cours permet aux étudiants de consolider leurs connaissances à travers des exercices et des études de cas, et de développer des compétences analytiques et méthodologiques nécessaires à la réalisation de projets logiciels de qualité.

Mots clés : Génie logiciel, conception des systèmes, approche systémique, cycle de vie du logiciel, analyse des besoins, modélisation, qualité logicielle, modèles de développement, conception statique et dynamique, prototypage, maintenance logicielle.

Avant Propos

Ce polycopié a été élaboré dans le but d'accompagner les étudiants de quatrième année PES Informatique dans l'apprentissage des concepts fondamentaux du Génie Logiciel. Il présente les principes essentiels liés au cycle de vie du logiciel, depuis l'analyse des besoins jusqu'à la maintenance, en mettant l'accent sur les méthodes, les outils et les bonnes pratiques de conception.

L'objectif principal de ce support est de sensibiliser les étudiants à l'importance d'une démarche structurée et méthodique dans le développement logiciel, en privilégiant la phase de conception avant toute activité de codage. À travers une approche progressive et pédagogique, ce polycopié vise à fournir une base solide permettant aux étudiants de mieux comprendre les enjeux du développement de logiciels fiables, évolutifs et de qualité.

Ce document présente un support d'apprentissage et de référence, destiné à faciliter la compréhension des notions abordées en cours et à accompagner les étudiants tout au long de leur parcours académique dans ce module.

Le présent polycopié est organisé en deux partie comme suit :

— La première partie est consacrée aux aspects théoriques du Génie Logiciel et de la conception des systèmes, permettant aux étudiants d'acquérir les connaissances fondamentales et méthodologiques nécessaires à la compréhension du processus de déve-

loppement logiciel.

— La deuxième partie est dédiée aux exercices et examens types, visant à renforcer l'assimilation des concepts abordés dans la première partie et à développer les compétences analytiques et méthodologiques des étudiants.

Fiche matière

Public cible

Ce cours s'adresse aux étudiants de :

— $4^{ième}$ année PES Informatique à l'ENS Oran Ammour-Ahmed.

— Semetre 4 (L2) en Licence Informatique

— Semetre 5 (L3) en Licence Informatique

Pré-requis

Pour suivre convenablement ce cours de Génie Logiciel, l'étudiant doit :

— Maîtriser des concepts fondamentaux de l'informatique et de la programmation.

— Avoir des notions de logique informatique et algorithmique.

— Avoir des connaissances sur le développement logiciel, et les langages de programmation.

— Posséder des compétences analytiques et organisationnelles,

Objectifs

Ce cours permet de fournir à l'étudiant une compréhension approfondie du cycle de vie du logiciel, ainsi que des enjeux et contraintes propres à chacune des phases du développement, depuis l'analyse des

besoins jusqu'à la phase de maintenance.

Son objectif est de sensibiliser l'étudiant à l'importance de chaque étape et de l'aider à adopter et suivre une approche structurée et réfléchie dans le processus de développement d'un logiciel, plutôt que de se concentrer directement sur le codage et la programmation.

Le cours met l'accent sur la phase de conception, afin de rendre l'étudiant capable à :

— Analyser et formaliser les besoins des utilisateurs,

— Modéliser et planifier les architectures logicielles,

— Anticiper les problèmes potentiels et les solutions appropriées,

— Prendre des décisions éclairées avant toute implémentation.

Le but ici, est de préparer l'étudiant à aborder le développement logiciel de manière méthodique, en comprenant que la qualité et la robustesse d'un logiciel dépendent autant de la conception que de sa réalisation effective.

Déroulement du cours

Le cours du Génie Logiciel est annuel pour la formation PES (*Professeur d'Enseignement Secondaire*) Informatique avec une séance de cours et une séance de TD par semaine et semestriel pour la formation L2 et L3 de spécialité Licence Informatique.

Coefficient

Le cours de Génie Logiciel est de coefficient **3** pour la formation PES Informatique, et de coefficient **2** pour la formation L2 et L3 Licence Informatique.

Partie I : Cours

Cette partie est dédiée à la présentation des concepts théoriques, permettant aux étudiants d'acquérir les connaissances fondamentales pour le Génie Logiciel.

Chapitre 1: La conception Système

Table des matières

1.1 Les enjeux de l'approche systémique

L'approche systémique est une manière de définir, étudier, ou expliquer tout type de phénomène, qui consiste avant tout à considérer ce phénomène comme un système : un ensemble complexe d'interactions, souvent entre sous-systèmes, le tout au sein d'un système plus grand.

Elle se distingue des approches traditionnelles qui s'attachent à découper un système en parties sans considérer le fonctionnement et l'activité de l'ensemble, selon diverses perspectives et à différents niveaux d'organisation. L'approche systémique prend en compte les diverses interactions existantes entre les parties du système.

1.2 L'approche systémique en pratique

Cela revient à répondre à la question : Comment transformer ces concepts théoriques en actions concrètes sur le terrain ?

Cas pratique : Système de gestion d'une bibliothèque universitaire, où l'université souhaite développer un système permettant de :

1. Gérer les livres,
2. Gérer les emprunts et retours,
3. Gérer les utilisateurs (étudiants, enseignants),
4. Suivre la disponibilité des ouvrages.

Définition du système (vision globale) : Système étudié : Système de gestion de bibliothèque universitaire

Objectif : Automatiser la gestion des livres et améliorer l'accès aux ressources.

Environnement :

1. Étudiants

2. Enseignants

3. Bibliothécaires

4. Base de données

5. Réseau universitaire

Décomposition du système (vision modulaire) : Le système est structuré en quatre modules principaux, voir Table (1.1), chacun assure des fonctionnalités précises.

Modules	**Fonctionnalités**
Gestion des utilisateurs	Inscription Authentification Gestion des profils
Gestion des livres	Ajout / suppression de livres Classification Disponibilité
Gestion des emprunts	Enregistrement des emprunts Suivi des retards Historique
Recherche	Recherche par titre Recherche par auteur Filtrage par catégorie

TABLE 1.1 – Modules et fonctionnalités du système de gestion de bibliothèque

Cette organisation modulaire facilite la compréhension, la maintenance et l'évolution du système.

La Figure (1.1) illustre la carte mentale pour ce système de gestion de bibliothèque universitaire.

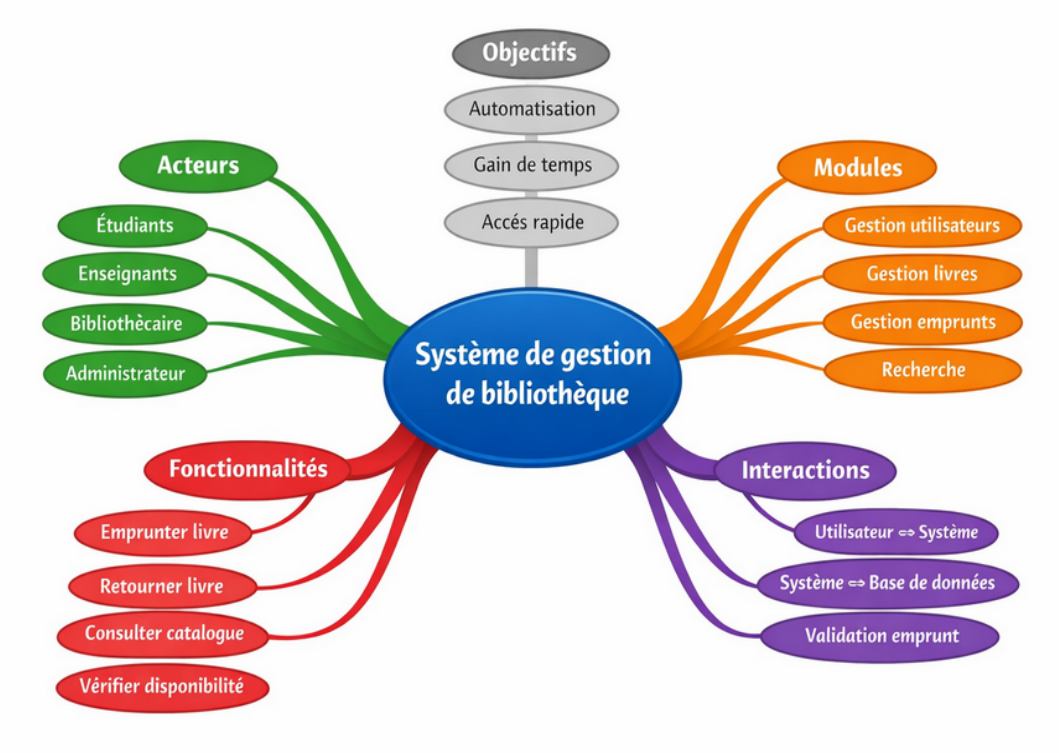


FIGURE 1.1 – Carte mentale du système de gestion de bibliothèque

Le système de gestion de bibliothèque est conçu comme un ensemble structuré et inter-connecté centré sur les utilisateurs, notamment les étudiants, enseignants, bibliothécaires et administrateurs.

Le système repose sur plusieurs modules essentiels, chacun avec des fonctionnalités bien précises, comme expliqué précédemment.

Les interactions entre les utilisateurs, le système et la base de données garantissent le bon fonctionnement et la cohérence des opérations.

L'objectif global est d'optimiser la gestion de la bibliothèque en automatisant les processus, en réduisant le temps de traitement et en facilitant un accès rapide et efficace à l'information.

1.3 Notion de système

Un système est un ensemble d'éléments interagissant entre eux selon certains principes ou règles. Un système est déterminé par :

— Sa frontière, c'est-à-dire le critère d'appartenance au système (déterminant si une entité appartient au système ou fait au contraire partie de son environnement) ;
— Sa mission (ses objectifs et sa raison d'être) ;
— Ses interactions avec son environnement ;
— Ses fonctions (qui définissent ce qu'ont le droit de faire ou non les entités faisant partie du système, leur organisation et leurs interactions) ;
— Ses ressources, qui peuvent être de natures différentes (humaine, naturelle, matérielle, immatérielle...), leur organisation et leurs interactions.

Quatre concepts sont fondamentaux pour comprendre un système, voir Figure (1.2) :

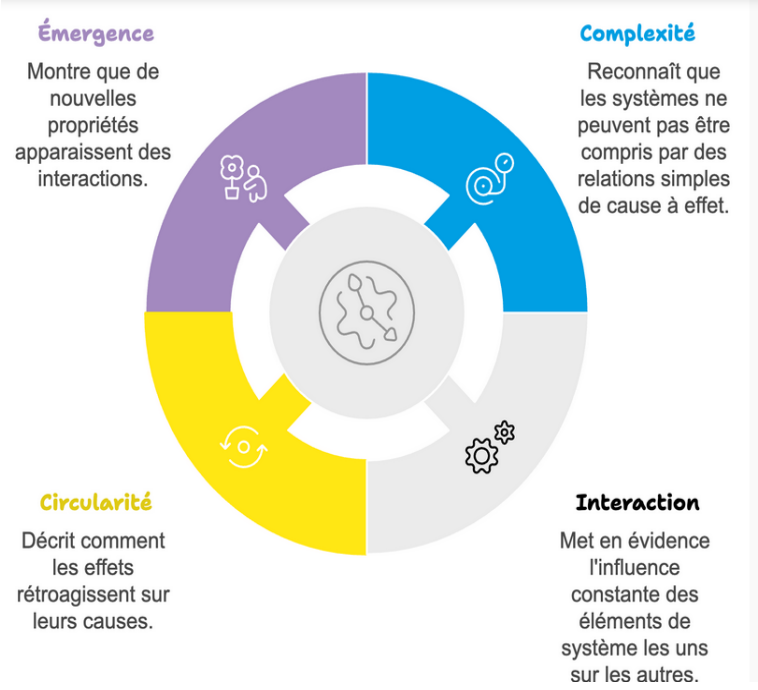


FIGURE 1.2 – Concepts fondamentaux de l'approche systémique

— L'interaction (ou l'interrelation) : Les éléments d'un système sont en interaction constante et s'influencent mutuellement.
— La circularité : Les effets rétroagissent sur leurs causes dans des

boucles de feedback.

— L'émergence : De nouvelles propriétés émergent des interactions entre les éléments.

— La complexité d'un système tient au moins à trois facteurs :
  — Le degré élevé d'organisation ;
  — L'incertitude de son environnement ;
  — La difficulté ou l'impossibilité d'identifier tous les éléments et toutes les relations.

1.3.1 Description d'un système

La description d'un système concerne la description de son l'aspect structurel et son aspect fonctionnel.

Aspect structurel d'un système

D'un point de vue structurel, un système comprend quatre composants comme suit :

1. Les éléments constitutifs : on peut en évaluer le nombre et la nature (même si ce n'est qu'approximativement). Ces éléments sont plus ou moins homogènes (exemple d'un automobile : groupe motopropulseur, châssis, habitacle, liaison au sol, carrosserie).
2. Une limite (ou frontière) qui sépare la totalité des éléments de son environnement.
3. Des réseaux de relations : les éléments sont en effet inter-reliés. Plus les interrelations sont nombreuses, plus le degré d'organisation est élevé et plus grande est la complexité. Les relations peuvent être de toutes sortes. Les deux principaux types de relations sont les transports et les communications.

4. Les stocks (ou réservoirs) où sont entreposés les matériaux, l'énergie ou l'information constituant les ressources du système qui doivent être transmises ou réceptionnées.

Aspect fonctionnel d'un système

D'un point de vue fonctionnel, un système comprend :

1. Des flux de matériaux, d'énergie ou d'informations, qui empruntent les réseaux de relations et transitent par les stocks. Ils fonctionnent par entrées/sorties (ou inputs/outputs) avec l'environnement ;
2. Des centres de décision qui organisent les réseaux de relations, c'est-à-dire coordonnent les flux et gèrent les stocks ;
3. Des boucles de rétroaction qui servent à informer, à l'entrée des flux, sur leur sortie, de façon à permettre aux centres de décision de connaître plus rapidement l'état général du système ;
4. Des ajustements réalisés par les centres de décisions en fonction des boucles de rétroaction et de délais de réponse (correspondant au temps que mettent les informations « montantes » pour être traitées et au temps supplémentaire que mettent les informations « descendantes » pour se transformer en actions).

1.4 Typologie des systèmes

1.4.1 La typologie de Jacques Lesourne

Cette typologie distingue :

Les systèmes à états : transformations entrées/sorties, sans régulation interne. (Exemple : un moteur de voiture).

Les systèmes à buts : régulation interne intégrée, capacité d'atteindre des objectifs. (Exemple : une chambre avec thermostat).

Les systèmes à apprentissage : incluant mémoire, mécanismes de calcul, et capacité de prise de décision et d'adaptation en fonction des données enregistrées.

Les systèmes à décideurs multiples (structure complexe de plusieurs systèmes à buts, s'organisant de manière spontanée (jeux) ou de façon hiérarchique (organisations). Lorsque les hiérarchies sont enchevêtrées en un système encore plus large et complexe, on parle de société.

1.4.2 La typologie de Jean-Louis Le Moigne

Cette typologie sépare :

Les systèmes-machines, qui relèvent de la mécanique et de l'ingénierie.

Les systèmes vivants (et systèmes artificiels complexes), dans lesquels apparaissent les processus de mémorisation, des centres de décision (ou de commande) et de coordination (ou de pilotage).

Les systèmes humains et sociaux, avec l'apparition de l'intelligence (ou capacité à traiter des informations symboliques), permettant une auto-organisation par des mécanismes abstraits d'apprentissage et d'invention.

1.5 Conception statique d'un système

La conception statique d'un système consiste à décrire la structure du système sans tenir compte de son évolution dans le temps. Elle représente les éléments qui composent le système et les relations entre eux.

1.5.1 Spécification Technique du Besoin (STB)

La STB est un document qui traduit le besoin d'un demandeur en termes d'exigences et contraintes techniques (spécifications). Ce document est approuvé par le demandeur, il est en général écrit conjointement par le demandeur et le concepteur/réalisateur et plus rarement par le demandeur seul.

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

Elle forme un ensemble de documents destinés à fournir la base du système documentaire relatif à tout produit matériel ou logiciel.

Pour établir la STB, on peut s'appuyer, s'il existe, sur le Cahier des Charges Fonctionnel [1].

La STB est initiée dès le début de l'étude de faisabilité (on parle alors de STB préliminaire en fin de phase de faisabilité), elle est progressivement enrichie pendant les études de conception.

Cette STB doit être figée et approuvée avant le début de la phase de réalisation.

Si celle-ci doit malgré tout être modifiée en cours de réalisation, ces modifications devront au préalable être approuvées par toutes les parties concernées et tracées dans un document.

Avant toute conception de logiciel, il est important de rédiger une spécification technique de besoin pour fixer clairement les objectifs, les contraintes et les besoins fonctionnels auprès des ingénieurs, prestataires, bureau d'études externes chargés du développement du logiciel.

La STB consigne les résultats de l'activité analyse des besoins du logiciel. Elle recense tous les besoins identifiés pendant cette activité sous forme de fonctions à remplir et de contraintes. Elle doit se limiter

1. Le CCF définit ce que le système doit faire pour répondre aux attentes des utilisateurs.

à exprimer des exigences et ne pas exposer de solutions pour répondre aux exigences.

La STB est le document de base pour la conception de l'architecture du logiciel où on recherche une solution qui satisfait toutes les exigences et contraintes de la STB.

La STB est aussi la référence pour la définition des essais de validation.

Avant l'apparition des techniques UML (Unified Modeling Language), la STB était basée sur les techniques d'analyse structurée type SADT (Structured Analysis and Design Technique) avec une décomposition arborescente des fonctions.

La STB comprenait 4 grandes parties :

— La mission du logiciel,

— Les exigences fonctionnelles (décomposition arborescente et diagrammes type SADT/SART),

— Les exigences d'interface,

— Les exigences complémentaires (environnement, mise en œuvre, disponibilité, sécurité, contraintes de conception et de réalisation...).

1.5.2 Analyse structurée et modélisation technique

La méthode SADT (Structured Analysis and Design Technique) débute du général (dit "niveau A-0") vers le particulier et le détaillé.

La SADT est une démarche systémique de modélisation d'un système complexe ou d'un processus opératoire.

Cette analyse, décrite par un modèle graphique, procède donc par approche descendante d'une manière que l'on va du plus général, au plus détaillé en s'intéressant aux activités du système. Pour cela il faut

décomposer la fonction Globale du système en modules fonctionnel (Boites).

Ces modules pouvant être eux-mêmes décomposés progressivement par niveau apportant des informations supplémentaires et permettant d'identifier les moyens et les activités utilisés pour réaliser la fonction globale.

Représentation graphique

Cela consiste à détailler le système en le divisant en sous-systèmes.

On décompose ainsi le système, en niveau inférieur (0, 1, 2, ...).

Le niveau 0 contient les boites 1, 2, 3... qui sont elles mêmes décomposées en boites $3_1, 3_2, 3_3$... pour la boite 3 et i_1, i_2, i_3... pour la boite i et ainsi de suite en descendant les niveaux.

Pour organiser la décomposition, il existe une règle : une boite peut être décomposée et dans ce cas en minimum 3 autres boites et au maximum 6.

De plus chaque flèche entrant ou sortant de la boite-mère doit se retrouver dans le diagramme enfant et doit être en relation avec au moins un enfant.

Niveau A-0 (A moins zéro)

Il définit :

La frontière d'isolement du système.

La fonction globale du système qui permet d'apporter de la valeur ajoutée à la matière d'œuvre, voir Figure (1.3).

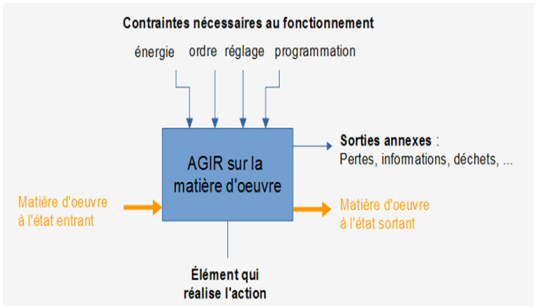


FIGURE 1.3 – conception statique d'un système (Niveau A-0)

Niveau A0 (A zéro)

Chaque boite représente une action que doit réaliser un constituant du système pour lui permettre de satisfaire la fonction globale, voir Figure (1.4).

Ce niveau permet d'observer les flux d'énergie et d'information entre les différentes boites nommées A1, A2, A3...

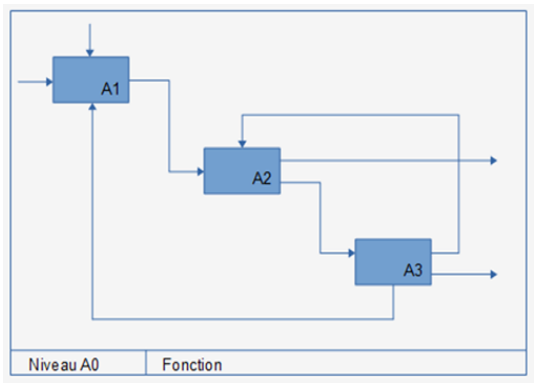


FIGURE 1.4 – conception statique d'un système (Niveau A0)

Niveau inférieur

La méthode est valide à condition de veiller à la cohérence des informations aux changements de niveaux, voir Figure (1.5).

Tout ce qui franchit les frontières d'une boite doit se retrouver à l'identique au niveau suivant.

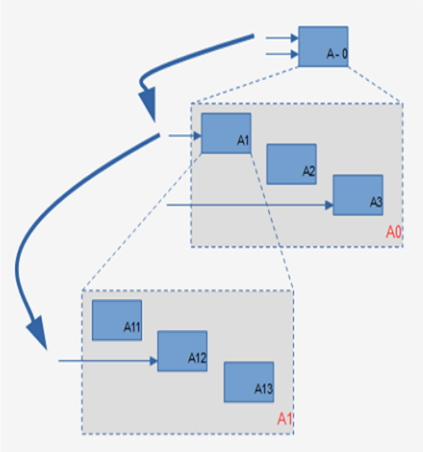


FIGURE 1.5 – conception statique d'un système (Niveau inférieur)

Le diagramme obtenu ressemble à celui illustré sur la Figure (1.6).

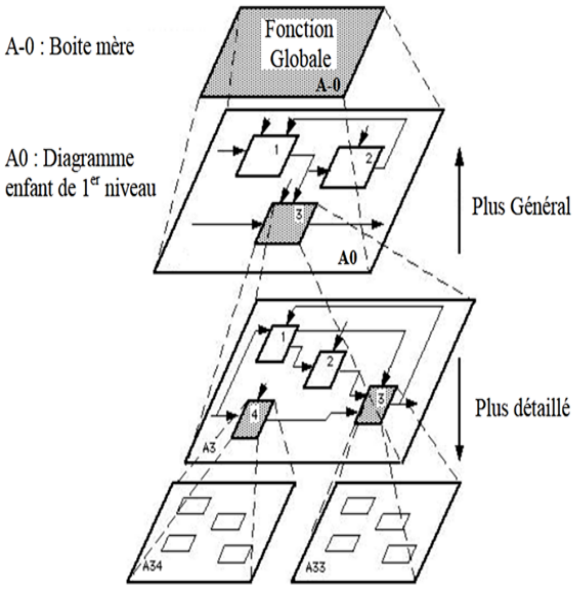


FIGURE 1.6 – Le diagramme obtenu

On ne dépasse que très rarement l'analyse de deux niveaux successifs, souvent A-0 puis A0, voir Figure (1.7)

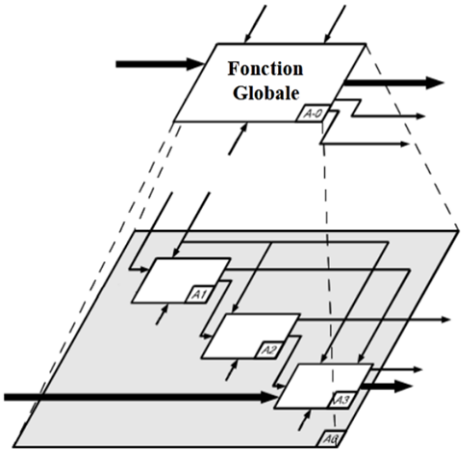


FIGURE 1.7 – Le diagramme du niveau A-0 et A0

1.6 Conception dynamique d'un système

1.6.1 Les réseaux de Pétri

Un réseau de Petri est un modèle mathématique permettant la représentation de systèmes distribués discrets (informatique, industriel), introduit par Petri (1962).

Il est également un langage de modélisation, représenté sous forme d'un graphe biparti orienté, dont on particularise les deux familles de sommets : les places et les transitions.

Les places sont représentées par des cercles, tandis que les transitions sont représentées par des traits ou des rectangles, voir Figure (1.8).

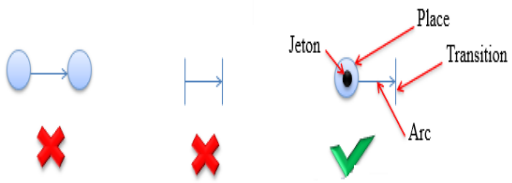


FIGURE 1.8 – Le réseau de Pétri

Comme dans tout graphe biparti, un arc ne relie jamais deux sommets de la même famille.

Une transition sans place d'entrée, Figure(1.9) (a), est une transition dite : **transition source**

Une transition sans place de sortie, Figure(1.9) (b), est une transition dite : **transition puits**

(a) (b)

FIGURE 1.9 – Modélisation d'une transition

Un réseau de Petri est un graphe orienté biparti défini par un quadruplet R = (P, T, Entrée, Sortie), où :

1. P est un ensemble fini de places , $P = P1, P2, P3, \ldots...., Pn$.
2. T est un ensemble fini de transitions, $T = T1, T2, T3, \ldots...., Tn$.
3. Entrée (ou Pré) est une application, Entrée : P x T $\rightarrow$ N, appelée application d'incidence avant. Notée Pré(p, t) ou Entrée(p, t) ou encore I(p, t), contient la valeur entière « n » associée à l'arc allant de la place p à la transition t.
4. Sortie (ou Post) est une application, Sortie : P x T $\rightarrow$ N, appelée application d'incidence arrière. Notée Post(p, t) ou Sortie(p, t) ou encore O(p, t) contient la valeur entière « n » associée à l'arc allant de la transition t à la place p.

Par convention, lorsque le poids n'est pas précisé sur un arc, alors ce poids vaut 1.

1.6.2 Franchissement d'une transition dans un RdP

Le franchissement d'une transition ne peut s'effectuer que si chacune des places en amont de cette transition contient au moins une marque (un jeton), on dit que cette transition est franchissable ou validée, voir Figure (1.10).

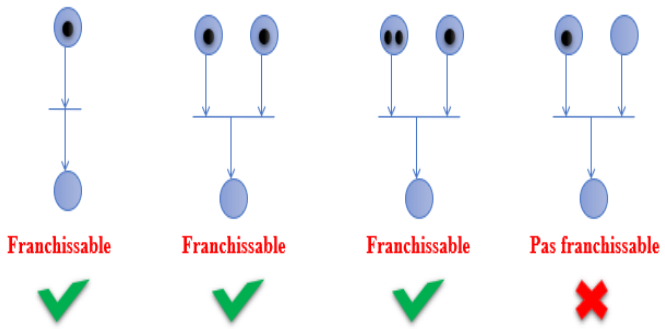


FIGURE 1.10 – Franchissement d'une transition

Le franchissement d'une transition consiste à retirer une marque de chacune des places en amont de la transition Tj et à ajouter une marque dans chacune des places en aval de Tj, voir Figure (1.11).

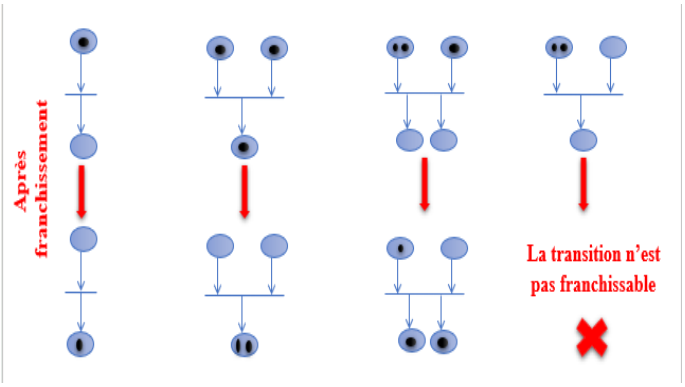


FIGURE 1.11 – Résultat de franchissement

- **Franchissement d'une transition source**

Une transition source est une transition qui ne comporte aucune place d'entrée ; c'est une transition toujours franchissable et le franchissement a lieu lorsque l'événement associé se produit.

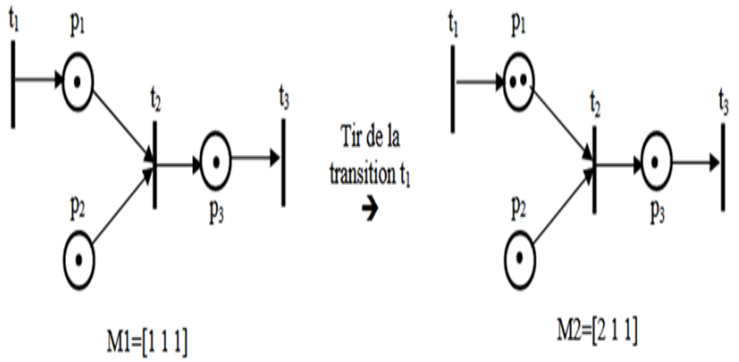


FIGURE 1.12 – Franchissement d'une transition source

• **Franchissement d'une transition puits**

Une transition puits est une transition qui ne comporte aucune place de sortie ; le franchissement d'une transition puits enlève des jetons de toutes les places d'entrée de la transition, voir Figure (1.13).

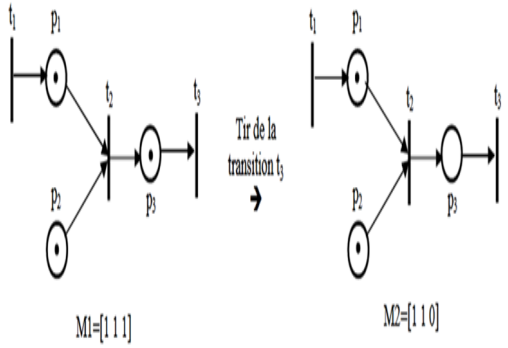


FIGURE 1.13 – Franchissement d'une transition puits

Une transition est dite validée si les places situées en amont (places d'entrée) possèdent chacune un nombre de jetons supérieur ou égal au poids des arcs joignant ces places à la transition.

Cela consiste à enlever de chaque place d'entrée un nombre de jetons égal au poids de l'arc joignant cette place à la transition et à déposer, dans chaque place aval (place de sortie), un nombre de jetons égal au poids de l'arc joignant la transition à chacune de ces places, voir Figure (1.14).

Le franchissement d'une transition est indivisible est de durée nulle.

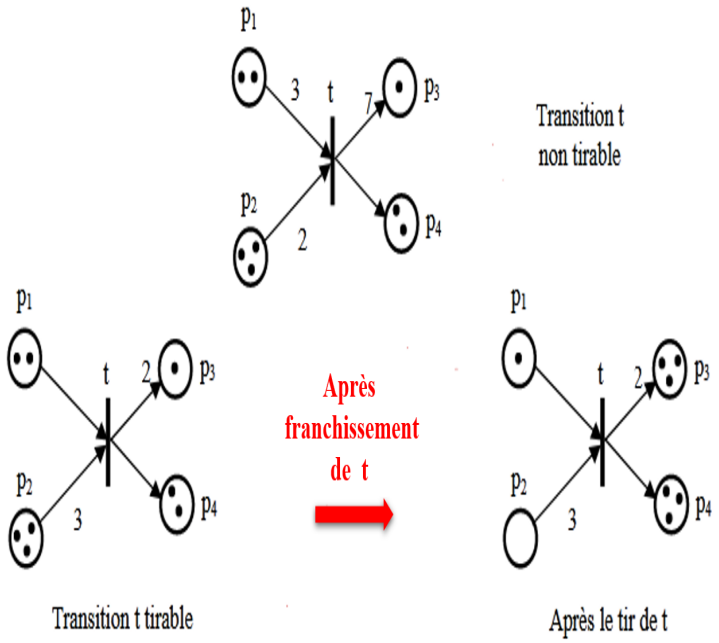


FIGURE 1.14 – Transition validée

Le RdP ne peut évoluer que par franchissement d'une seule transition à la fois (une transition choisie parmi toutes celles qui sont validées à cet instant).

Le franchissement des transitions et les modifications du marquage qu'il entraîne permettent d'analyser la dynamique du système modélisé.

Exemple

Une molécule d'eau est constituée de deux atomes d'hydrogène et d'un atome d'oxygène. Cette réaction chimique peut être modélisée par un RdP, voir Figure (1.15).

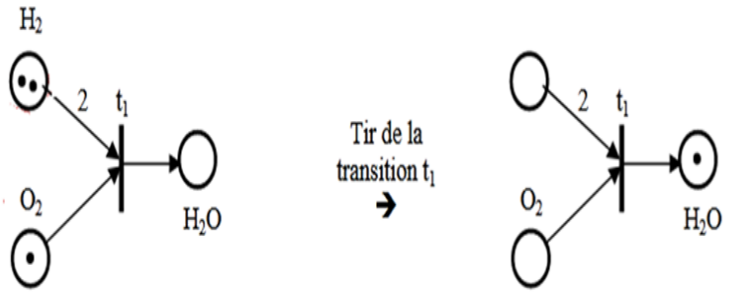


FIGURE 1.15 – Modélisation de la molécule H_2O Par un RdP

1.6.3 Marquage d'un RdP

Le marquage d'un RdP est précisé par la présence à l'intérieur des places d'un nombre fini (positif ou nul), de marques ou de jetons. Une place est donc vide ou marquée, voir Figure (1.16).

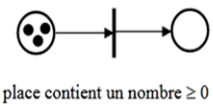


FIGURE 1.16 – Marquage d'un RdP

Lorsque la place représente une condition logique (exemples : machine à l'arrêt, convoyeur en panne), la présence d'un jeton indique que cette condition est vraie ; fausse dans le cas contraire.

Une place donc peut représenter une ressource du système (un stock par exemple), elle peut contenir plusieurs jetons (exemple :le stock, où le nombre de jetons peut indiquer le nombre de pièces stockés).

Le marquage initial, M_0, d'un RdP correspond à la distribution initiale des jetons dans chacune des places du RdP, qui précise l'état initial du système. Dans ce cas, on parle du RdP Marqué par opposition à un Rdp non marqué, c'est-à-dire pour lequel le marquage initial n'est pas précisé.

On note M(p) le nombre de jetons contenu dans la place p pour le marquage M.

Dans l'exemple illustré dans la Figure (1.17), si l'état initial correspond à la répartition des jetons suivante, alors le marquage initial : $M_0 = [M_0(p1), M_0(p2)] = [3,\ 0]$.

Un marquage peut être représenté sous forme d'un vecteur colonne :

$$M_0 = [M_0(p1), M_0(p2)] = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

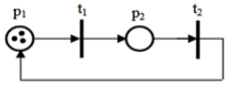


FIGURE 1.17 – Marquage initiale d'un RdP

Remarque : La présentation faite des Réseaux de Petri est une représentation graphique, alors qu'il est possible d'en faire une représentation mathématique grâce à l'algèbre linéaire.

Cette notation permet de représenter un RdP de manière plus formelle mais moins lisible que la représentation graphique. Ainsi, *Pre* et **Post** sont représentés par des matrices à **q** lignes (nombre de places), n colonnes (nombre de transitions).

1.6.4 Graphe de marquage

L'évolution temporelle d'un RdP peut être décrite par un graphe de marquage représentant l'ensemble des marquages accessibles et d'arcs correspondant aux franchissements des transitions faisant passer d'un marquage à l'autre pour un marquage initial M_0, voir Figure (1.18).

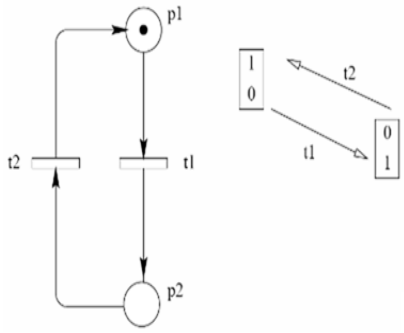


FIGURE 1.18 – Graphe de marquage d'un RdP

1.6.5 Matrice d'incidence

La matrice d'incidence noté C d'un RdP est donnée comme suit :

$$C = Post - Pre$$

Exemple 1 (voir Figure (1.19)) :

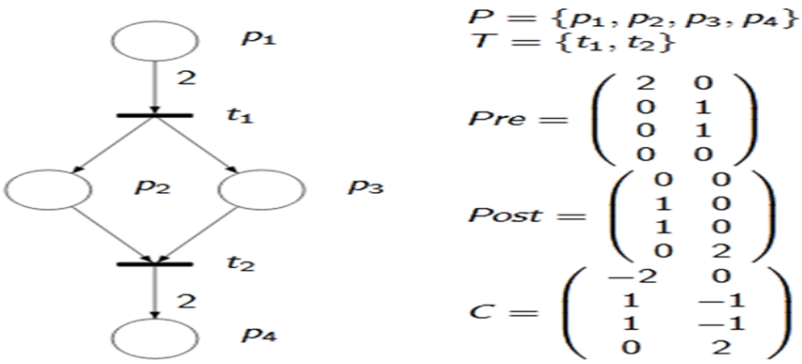


FIGURE 1.19 – Matrice d'incidence Exemple 1

Exemple 2 (voir Figure (1.20)) :

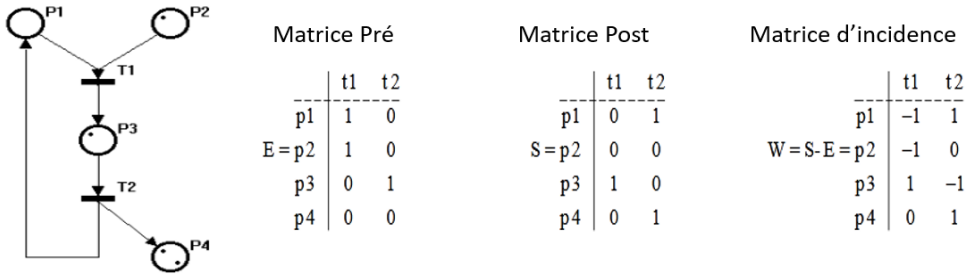


FIGURE 1.20 – Matrice d'incidence Exemple 2

Interprétation : La table (1.2) représente l'interprétation du marquage sur un réseau de Pétri.

Place d'entrée	**Transition**	**Place de sortie**
Précondition	Événement	Postcondition
Donnée d'entrée	Traitement	Donnée de sortie
Ressources nécessaires	Job ou activité	Ressources libérées
Buffer d'entrée	Processus	Buffer de sortie

TABLE 1.2 – Correspondance entre places et transitions

1.6.6 Séquence de franchissement

Le franchissement, à partir de M0, de T1 puis T2 conduit au marquage M2. On appellera T1T2 séquence de franchissement et on no-

tera :

$$M_0 \xrightarrow{T_1T_2} M_2 \quad \text{ou encore} \quad M_0 \xrightarrow{S} M_2 \quad \text{avec } S = T_1T_2$$

Une séquence de franchissement à partir d'un marquage M1 est représentée par la suite des transitions Ti, Tj, ... telles que le franchissement de chacune d'elles conduit à un marquage avec :

$$S : M_1 \xrightarrow{T_i} M_2 \xrightarrow{T_j} ... \xrightarrow{T_r} M_q$$

On dit que la séquence $S = T_i, T_j, ..., T_r$ est applicable à partir de M1, on écrit :

$$M_1 \xrightarrow{S} M_q$$

Exemple

La Figure (1.21) présente un exemple de RdP avec un marquage initiale M_0. La séquence $S = T_1, T_2, T_3$ est applicable à partir de M_0 et conduit au marquage

Après $T1 \rightarrow 01100$

Après $T2 \rightarrow 00110$

Après $T3 \rightarrow 00011$

$$M_0 = (1, 0, 0, 0, 0) \xrightarrow{S=T_1,T_2,T_3} M_4 = (0, 0, 0, 1, 1)$$

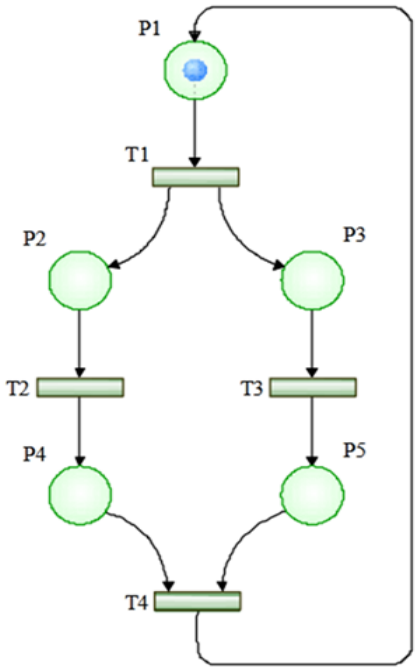


FIGURE 1.21 – Exemple d'un RdP

1.6.7 Propriétés d'un RdP

1. RdP Borné

Une place Pi est dite bornée par un marquage initiale M0 s'il existe un entier naturel K tel que pour tout marquage accessible à partir de M0, le nombre de marque dans une place Pi <= K.

$$\exists\, \alpha_i \in \mathbb{N} \;/\; \forall M \in M_0^*,\; M(P_i) \leq \alpha_i$$

Un RdP est borné pour un marquage initial Mo, si toutes ses places sont bornées pour Mo.

2. RdP sauf (ou binaire)

un RdP est sauf (ou binaire) pour un marquage initial Mo, si pour tout marquage accessible, chaque place contient au plus un jeton.

Remarque : Un réseau sauf est borné.

Exemple :

Le RdP illustré sur la Figure (1.22) est un RdP sauf /borné par 1

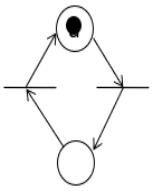


FIGURE 1.22 – Exemple d'un RdP sauf

3. Blocage

Un blocage (état puits) est marquage tel que aucune transition n'est validée.

Exemple 1 :

Soit le Rdp présenté sur la Figure (1.23), avec le $M_0 = (1, 0, 2)$

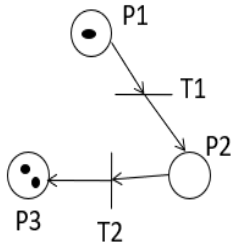


FIGURE 1.23 – RdP exemple 1

Après $T1 \rightarrow M1 = (0, 1, 2)$ Après $T2 \rightarrow M2 = (0, 0, 3) \rightarrow M2$ Marquage de blocage

Un Rdp est dit sans blocage pour un marquage initial M_0, si aucun marquage accessible à partir de M_0 n'est un blocage.

Exemple 2 :

Soit le Rdp présenté sur la Figure (1.24), avec le $M_0 = (1, 0, 2)$

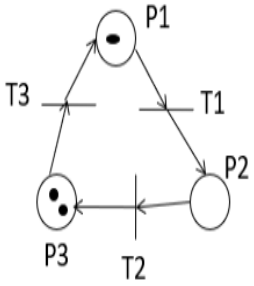


FIGURE 1.24 – RdP exemple 2

Après $T1 \rightarrow M1 = (0, 1, 2)$ Après$T2 \rightarrow M2 = (0, 0, 3)$ Après $T3 \rightarrow M0 = (1, 0, 2) \rightarrow$ RdP sans blocage

4. Vivacité

Une transition T_j est vivacité pour un marquage initial M_0 si pour tout marquage accessible à partir de M_0, il existe une séquence de franchissement S qui contient T_j à partir de M_i.

Un Rdp est dit vivant pour un marquage initial M_0, si toutes les transitions sont vivantes, voir Figure (1.25).

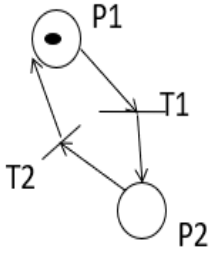


FIGURE 1.25 – Vivacité d'un RdP

Exemple :

$M_0 = (1, 0)$ Après $T1 \rightarrow M_1 = (0, 1)$ Après $T2 \rightarrow M_0 = (1, 0)$

T_1 est vivante car pour aller de M_1 à M_0 on passe par T_2, T_1, T_2 et pour aller à de M_0 à M_1 on passe par T_1. T_2 est aussi vivante donc le RdP vivant

Une transition Tj est quasi-vivante pour un marquage initial M_0 s'il existe une séquence de franchissement S qui contient T_j à partir

de M_i.

Un Rdp est dit quasi-vivant pour un marquage initial M_0, si toutes les transitions sont quasi-vivantes.

1.6.8 RdP particuliers

D'un point de vue structurel, on distingue :

1. Graphe d'états

Un RdP est un graphe d'états si et seulement si toute transition a exactement une place d'entrée et une place de sortie, voir Figure (1.26).

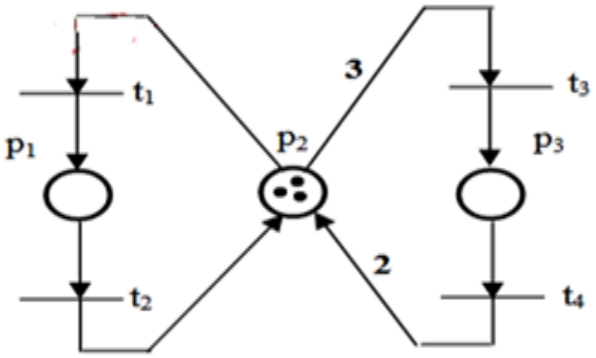


FIGURE 1.26 – Graphe d'état

2. Graphe d'événements

Un RdP est un graphe d'événement si et seulement si toute place a exactement une transition d'entrée et une transition de sortie, voir Figure (1.27).

FIGURE 1.27 – Graphe d'événement

3. RdP sans conflit

Un RdP dans lequel toute place a au plus une transition de sortie. Un conflit (ou conflit structurel) correspond à l'existence d'une place P1 qui a au moins deux transitions de sortie T1, T2,...On notera <P1, T1, T2,....>, voir Figure (1.28).

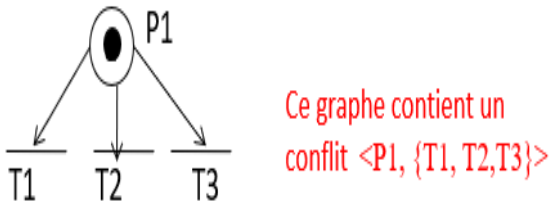


FIGURE 1.28 – Graphe avec conflit

4. RdP à choix libre

Un RdP à choix libre est un RdP dans lequel pour tout conflit <P1, T1, T2,....> aucune des transitions T1, T2,...ne possède une autre place d'entrée que P1, voir Figure (1.29).

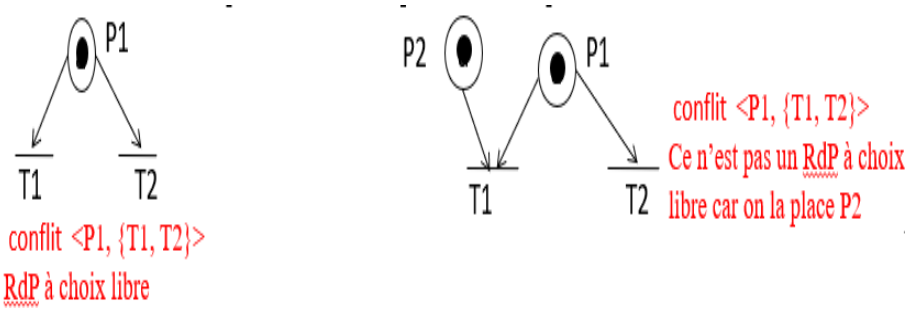


FIGURE 1.29 – Graphe à choix libre

5. RdP simple

C'est un RdP dans lequel chaque transition ne peut être concernée que par un conflit au plus.

Exemple : S'il existe une transition T1 et deux conflits <P1, T1, T2,...> et <P2, T1, T3,...>, alors le RdP n'est pas simple.

6. RdP pur

C'est un RdP dans lequel il n'existe pas de transition ayant une place d'entrée qui soit également place de sortie de cette transition.

Le graphe illustré sur la Figure (1.30) est un graphe impur.

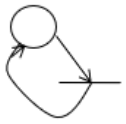


FIGURE 1.30 – RdP impur

1.7 Synthèse

Ce chapitre a exploré les fondements de la conception des systèmes en s'appuyant sur l'approche systémique. Cette dernière consiste à analyser un système en le considérant comme un ensemble d'éléments interagissant dans un environnement spécifique. Contrairement aux approches analytiques classiques, l'approche systémique privilège une vision globale qui englobe les relations, l'organisation et la complexité.

Un système est défini par ses frontières, sa mission, ses ressources, ses interactions et ses fonctions. L'étude d'un système peut être abordée suivant deux aspects complémentaires : un aspect structurel (éléments, relations, frontières, stocks) et un aspect fonctionnel (flux, décisions, rétroactions, ajustements).

La conception statique d'un système met l'accent sur sa structure, en particulier à travers la Spécification Technique du Besoin (STB), qui formalise les exigences fonctionnelles et contraintes. L'analyse structurée, notamment avec la méthode SADT, permet de décomposer progressivement le système du niveau global vers des niveaux plus détaillés.

La conception dynamique, quant à elle, est illustrée par les réseaux de Pétri, qui offrent un formalisme graphique et mathématique pour modéliser les systèmes à événements discrets.

Ainsi, la combinaison des approches statique et dynamique permet une compréhension complète des systèmes, d'un point de vue structurel et évolutif dans le temps.

Chapitre 2: Introduction au Génie Logiciel

Table des matières

2.1 Qu'est-ce qu'un logiciel ?

Un logiciel est un ensemble d'entités nécessaires au fonctionnement d'un processus de traitement automatique de l'information selon la spécification fournie.

L'ensemble des critères que doivent satisfaire un logiciel, son fonctionnement interne et ses interactions avec son environnement représente la spécification du logiciel.

2.2 Crise du logiciel

Le terme de « *Génie logiciel* » a été introduit à la fin des années soixante lors d'une conférence tenue pour discuter de ce que l'on appelait alors (*la crise du logiciel « software crisis »*).

Le développement de logiciel était en crise : les coûts du matériel chutaient alors que ceux du logiciel grimpaient en flèche.

Il fallait de nouvelles techniques et de nouvelles méthodes pour contrôler la complexité inhérente aux grands systèmes logiciels.

2.2.1 Analyse de l'existant

Constat du développement logiciel fin années 60 :

— Délais de livraison non respectés,

— Budgets non respectés,

— Ne répond pas aux besoins de l'utilisateur ou du client,

— Difficile à utiliser, maintenir, et faire évoluer.

Une étude réalisée par le département de la Défense des États-Unis 1995 sur les logiciels produits dans le cadre de 9 gros projets militaires est illustrée sur la Figure (2.1 (a)).

La Figure (2.1 (b)) représente une étude réalisée par Standish group sur l'utilisation des fonctionnalités implantées dans un logiciel.

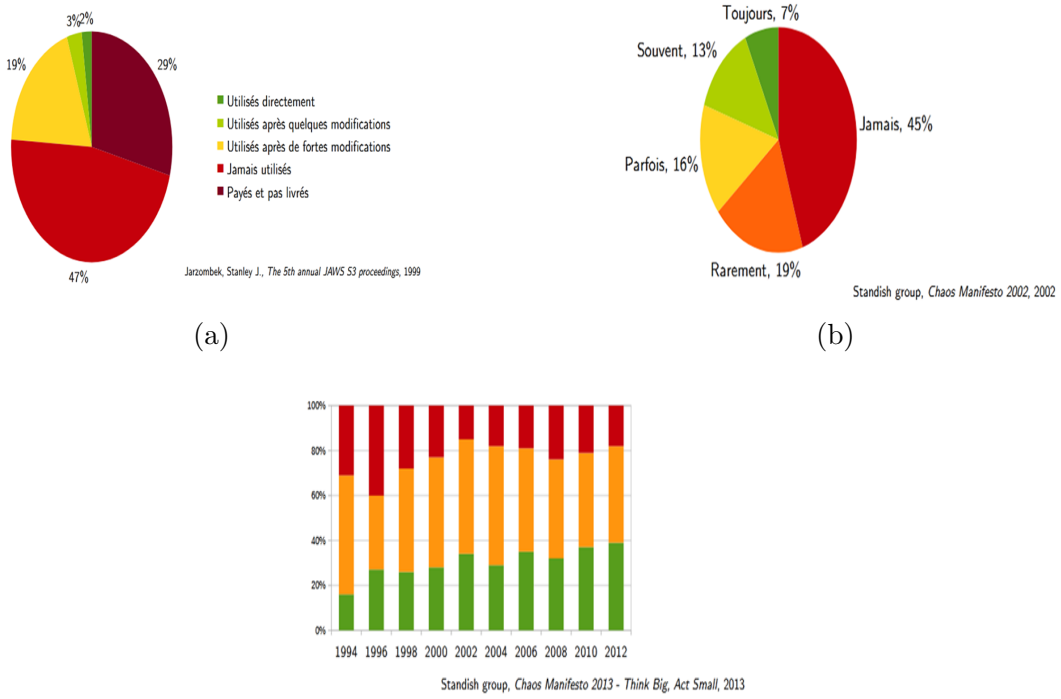
La Figure (2.1 (c)) représente une étude réalisée par Standish group, comme suit :

— Projets réussis : achevés dans les délais et pour le budget impartis, avec toutes les fonctionnalités demandées.

— Projets mitigés : achevés et opérationnels, mais livrés hors délais, hors budget ou sans toutes les fonctionnalités demandées

— Projets ratés : abandonnés avant la fin ou livrés mais jamais utilisés.

Selon Standish group, Chaos Report 2015, *La satisfaction du client et la valeur du produit sont plus grandes lorsque les fonctionnalités livrées sont bien moins nombreuses que demandé et ne remplissent que les besoins évidents.*



(a) (b)

(c)

FIGURE 2.1 – Analyse des logiciels existants

2.2.2 Raisons de la faible qualité des logiciels

La faible qualité des logiciels peut être causée par plusieurs facteurs liés à la conception, au développement et à la gestion du projet.Parmi ces facteurs, on peut distingué :

— La tâche complexe : Taille et complexité des logiciels.

— Le manque de méthodes de conception.

— La négligence et manque de méthodes et d'outils des phases de validation/vérification.

— la mauvaise compréhension des besoins par la négligence de la phase d'analyse des besoins du client (voir Figure (2.2)).

— Le manque d'implication du client dans le processus.

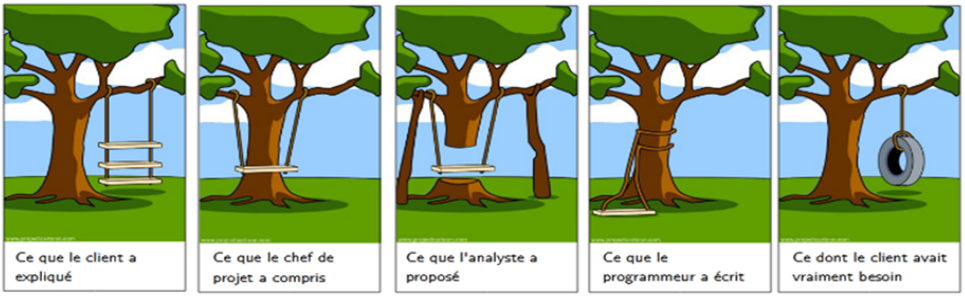


FIGURE 2.2 – Faible qualité des logiciels

La crise du logiciel était perçue à travers ces symptômes :

— La qualité du logiciel livré était souvent déficiente : Le produit ne satisfaisait pas les besoins de l'utilisateur, il consommait plus de ressources que prévu et il était à l'origine de pannes.

— Les performances étaient très souvent médiocres (temps de réponse trop lents).

— Le non-respect des délais prévus pour le développement de logiciels ne satisfaisant pas leurs cahiers des charges.

— Les coûts de développement d'un logiciel étaient presque impossible à prévoir et étaient généralement excessifs.

— L'invisibilité du logiciel, ce qui veut dire qu'on s'apercevait souvent que le logiciel développé ne correspondait pas à la demande (on ne pouvait l'observer qu'en l'utilisant « trop tard »).
— Défaillances logicielles : La maintenance du logiciel était difficile, coûteuse et souvent à l'origine de nouvelles erreurs.

2.3 Une solution : le Génie Logiciel

L'idée est d'appliquer les méthodes classiques d'ingénierie au domaine du logiciel.

2.3.1 Ingénierie

L'ingénierie (ou génie) : est l'ensemble des fonctions allant de la conception et des études à la responsabilité de la construction et au contrôle des équipements d'une installation technique ou industrielle (Génie civil, naval, mécanique,...)

2.3.2 Qu'est-ce que le Génie Logiciel ?

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

C'est l'ensemble des activités de conception et de mise en œuvre des produits et des procédures visant à rationaliser la production du logiciel et son suivi, en utilisant des principes, méthodes et procédures scientifiques bien définis (voir Figure 2.3).

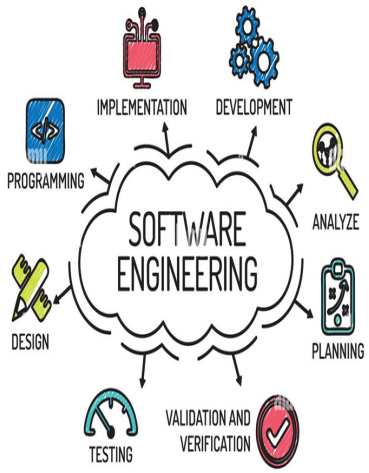


FIGURE 2.3 – Software engineering

Le résultat de l'ingénierie logicielle est un produit logiciel fiable est efficace.

Le terme du *Génie Logiciel* est composé de deux mots, le logiciel et l'ingénierie.

Le logiciel : est considéré comme une collection de code des programmes exécutables, des bibliothèques associées et de documentations. Ainsi, lorsque le logiciel, est conçu pour une exigence spécifique, est appelé un *produit logiciel*.

L'ingénierie (Génie) : consiste à développer des produits, en utilisant des principes et méthodes scientifiques bien définis.

Le génie logiciel englobe les tâches suivantes (voir Figure 2.4) :

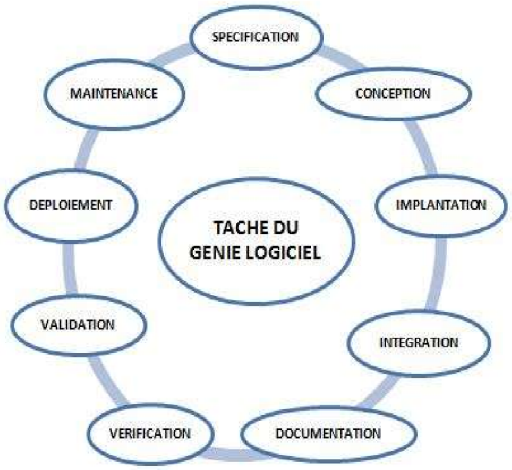


FIGURE 2.4 – Les tâches du génie logiciel

• **La Spécification** : capture des besoins, cahier des charges, spécifications fonctionnelles et techniques.

• **La Conception** : analyse, choix de la modélisation, définition de l'architecture, définition des modules et interfaces, définition des algorithmes.

• **L'Implantation** : choix d'implantations, codage du logiciel dans un langage cible.

• **L'Intégration** : assemblage des différentes parties du logiciel

• **La Documentation** : manuel d'utilisation du logiciel, et aide en ligne.

• **La vérification** : tests fonctionnels, tests de la fiabilité, tests de la sûreté

• **La Validation** : recette du logiciel, conformité aux exigences du cahier des charges.

• **Le Déploiement** : livraison, installation, formation pour son utilisation.

• **La Maintenance** : corrections, et évolutions.

2.3.3 La qualité exigée d'un logiciel

Fiabilité, sûreté et sécurité des logiciels de :

— Transports automobile, ferroviaire, aéronautique.

— Contrôle de processus industriels, nucléaire, armement.

— Médical : imagerie, appareillage, télé-surveillance.

— e-commerce, carte bancaire sans contact, passeport électronique.

Raisons économiques :

— Coût de la correction, du rappel des appareils défectueux.

— Coût de l'impact sur l'image, de l'arrivée tardive sur le marché.

— Coût en vies, coût de l'impact écologique.

La qualité du logiciel est définie par son aptitude à satisfaire les besoins des utilisateurs (spécification), voir Figure (2.5).

FIGURE 2.5 – Principaux facteurs influençant la qualité du logiciel

Pour juger la qualité d'un logiciel, il faut vérifier les critères suivants :

— Validité : répondre aux besoins des utilisateurs.

— Facilité d'utilisation : prise en main et robustesse.

— Performance : temps de réponse, débit, fluidité...

— Fiabilité : tolérance aux pannes.

— Sécurité : intégrité des données et protection des accès.

— Maintenabilité : facile à corriger ou à évoluer le logiciel.

— Portabilité : changement d'environnement matériel ou logiciel.

2.3.4 Les principes du Génie Logiciel

• **Rigueur et formalité**

— Le logiciel doit offrir une solution précise.

— Développement sérieux, soigné, non approximatif.

— Pas besoin d'être toujours formel, mais les techniques formelles peuvent être très utiles, par exemple, pour la vérification.

• **Séparation des questions**

Consiste à considérer séparément différents aspects d'un problème pour en maîtriser la complexité. Il Peut prendre différentes formes :

— Séparation dans le temps (cycle de vie).

— Séparation des qualités à optimiser (exemple : correction avant performance)

— Séparation des vues d'un système (vue des données, vue du contrôle).

— Séparation du système en parties (modularité).

• **Modularité (séparation des parties logiques)**

— Ce principe qui consiste à décomposer un système en sous-systèmes plus simples (pour maîtriser complexité).

— Caractéristiques d'une bonne conception modulaire :

  — Regroupement des aspects logiques semblables

  — Indépendance des modules

• **Abstraction (séparation des aspects)** Consiste à mettre les détails de côté pour ne considérer que les aspects jugés importants d'un système.

Il existe différents niveaux d'abstraction :

Exemple : circuit électronique

— Niveau 1 : Équation mathématique

— Niveau 2 : Circuit logique des composants

— Niveau 3 : Plan physique des composants réels au sein du circuit intégré.

• **Anticipation du changement** Consiste à prévoir les changements de façon à faciliter la gestion des évolutions (modifications).

Nature des changements :

— Perfectifs (nouveaux besoins)

— Correctifs

— Adaptatifs

Nature des problèmes : localisation des erreurs, gestion des versions multiples du logiciel.

• **Généralité**

Consiste à résoudre un problème plus général que le problème spécifique qui est adressé.

• **Construction incrémentale (raffinement)**

Consiste à construire une solution étape par étape en s'approchant de plus en plus du but.

Exemple : réaliser les fonctions essentielles d'un système puis ajouter aspects secondaires.

2.4 Synthèse

Le présent chapitre a introduit les concepts fondamentaux du génie logiciel en mettant l'accent sur les enjeux liés au développement des logiciels.

La crise du logiciel apparue vers la fin des années 60, caractérisée par des dépassements de délais et de budgets, une faible qualité

des produits, ainsi qu'une inadéquation avec les besoins des utilisateurs, a conduit à l'émergence du génie logiciel. Cette crise s'explique notamment par la complexité croissante des systèmes, le manque de méthodes structurées et une mauvaise analyse des besoins.

Pour remédier à ces problèmes, le génie logiciel propose une approche méthodique et rigoureuse inspirée des disciplines d'ingénierie. où il englobe l'ensemble des activités du cycle de vie du logiciel, allant de la spécification des besoins jusqu'à la maintenance, en passant par la conception, l'implémentation, l'intégration, les tests, le déploiement et la documentation.

La qualité du logiciel constitue un objectif central du génie logiciel. Elle est évaluée selon plusieurs critères tels que la validité, la fiabilité, la performance, la sécurité, la maintenabilité et la portabilité.

Enfin, le génie logiciel repose sur des principes fondamentaux permettant de maîtriser la complexité des systèmes, notamment la rigueur, la séparation des préoccupations, la modularité, l'abstraction, l'anticipation des changements, la généralisation et la construction incrémentale. Ces principes assurent une meilleure organisation du développement et contribuent à la production de logiciels de qualité, fiables, évolutifs et adaptés aux besoins des utilisateurs.

Chapitre 3: Le cycle de vie du logiciel

Table des matières

3.1 Qu'est ce qu'un cycle de vie de logiciel ?

Le cycle de vie d'un logiciel représente l'ensemble d'activités successives, organisées en vue de la production d'un logiciel, voir Figure(3.1).

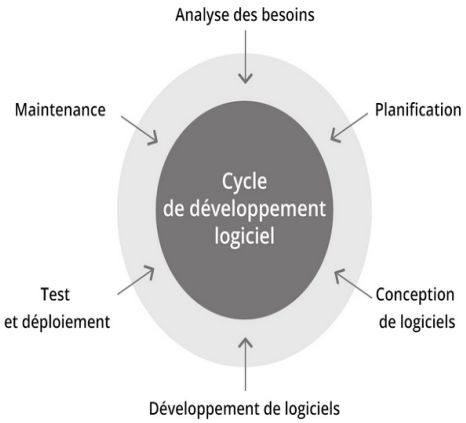


FIGURE 3.1 – Processus de Développement du Logiciel (PDL)

Le choix du processus est en fonction de plusieurs contraintes :

— Future produit,

— Taille des équipes (aspect humain),

— Temps,

— Aspect technique, ...

D'où la nécessité de disposer des modèles de développement généraux qui décrivent l'enchainement et les interactions entre les activités.

3.2 Les étapes du cycle de vie

Le développement d'un logiciel passe par les activités suivantes :

— Analyse des besoins

— Spécification

— Conception

— Programmation

— Vérification et Validation

— Livraison

— Maintenance

Pour chaque activité : exploitation des documents de l'activité précédente et élaboration de nouveaux supports (texte, programme, traces, ...) pour l'activité qui suit.

3.2.1 Analyse des besoins

Le but de l'analyse des besoins est d'étudier le domaine d'application ainsi que l'état actuel de l'environnement du future système afin de déterminer les frontières, les ressources disponibles, et les contraintes d'utilisation, voir Figure (3.2).

FIGURE 3.2 – Analyse des Besoins

Les données de cette activité sont fournies par des experts du domaines d'application et par les futures utilisateurs.

Le résultat de cette étape est un ensemble de documents décrivant les aspects les plus pertinents de l'environnement du future système, son rôle, et son utilisation.

Document principal : cahier des charges.

Les données résultantes de cette étape constitue les hypothèses

d'entrée pour la prochaine phase.

3.2.2 Spécification

Le but de la spécification est d'établir une première description du future système.

Les données de cette activité sont les résultats de l'Analyses des Besoins ainsi que des considérations techniques et de la faisabilité informatique.

Le résultat de cette étape est une description de ce que doit faire le logiciel sans évoquer les décisions prématurées de relation.

Cette phase résulte un document de spécification globale (un manuel).

3.2.3 La conception du logiciel

Cette activité consiste à enrichir la description du logiciel avec des détails de l'implémentation afin d'aboutir à une description très proche du programme.

Elle permet la définition des structures et des algorithmes, et se divise en deux étapes :

1. **Étape de la description architecturale** : Cette étape a pour but de décomposer le logiciel en composants plus simple en précisant les interfaces et les fonctions de chaque composant.

   On obtient à l'issus de cette étape une description architecturale de logiciel (arborescence) et un ensemble de spécification de ses divers composants.

2. **Étape de la conception détaillée** : Cette étape fournit pour chaque composant, une description de la manière dont les fonctions du composant sont réalisées (algorithme, structure de données, ...).

Les frontières entre l'activité de spécification et l'activité de conception est souvent floue. En effet, il n'est pas raisonnable que spécifier un système indépendamment de la considération de la faisabilité (le fait que la conception soit possible démontre la faisabilité du système), voir Figure (3.3).

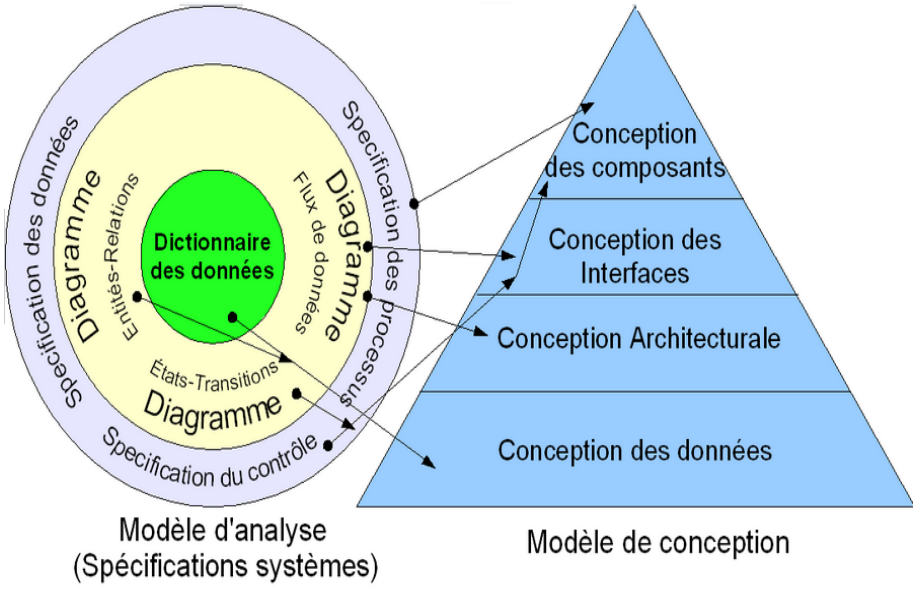


FIGURE 3.3 – Modèle de conception

La qualité d'une conception de logiciel se réfère à l'ensemble des caractéristiques qui déterminent si le logiciel est bien conçu, fiable, maintenable et efficace.

Dans ce contexte, une bonne conception ne se limite pas à ce que le logiciel fonctionne correctement : elle englobe aussi sa structure, sa facilité d'évolution et sa capacité à répondre aux besoins des utilisateurs.

3.2.4 L'implémentation (Programmation)

Cela consiste à passer le résultat de la conception détaillée à un ensemble de programmes ou de composants de programmes, voir Figure (3.4).

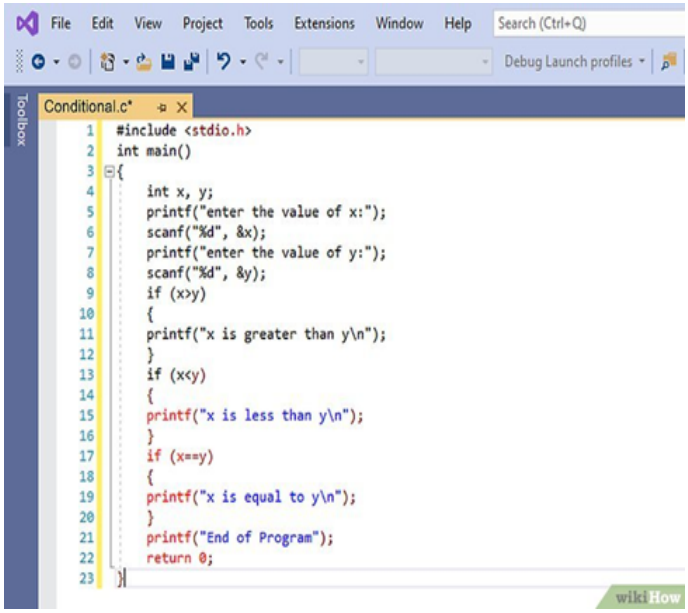


FIGURE 3.4 – Exemple de programme informatique

L'implémentation est la phase la plus outillée du processus de développement du logiciel. Elle constitue 15 à 20 % de l'effort nécessaire pour le développement d'un logiciel, alors que la phase de spécification et la phase de conception constituent 40 % de cet effort.

3.2.5 Gestion des configurations et intégrations

La configuration permet la gestion des composants du logiciel, d'en maitriser l'évolution et les mettre à jour tout au long du processus de développement du logiciel.

L'intégration consiste à assembler toutes parties de composants pour obtenir un système exécutable.

3.2.6 Vérification et validation

La phase de validation est la réponse à la question suivante : *A-t-on décrit le bon système ?* C'est à dire le système qui répond à l'attente des utilisateurs et aux contraintes de l'environnement (l'analyse des besoins et la spécification sont liées à la phase de validation.

La phase de vérification est la réponse à la question suivante : *Le développement est-il correcte à la spécification du départ ?* C'est-à-dire s'assurer que les descriptions successives du logiciel satisfont la conception architecturale et détaillée.

La Figure (3.5), représente la différence entre la validation et la vérification.

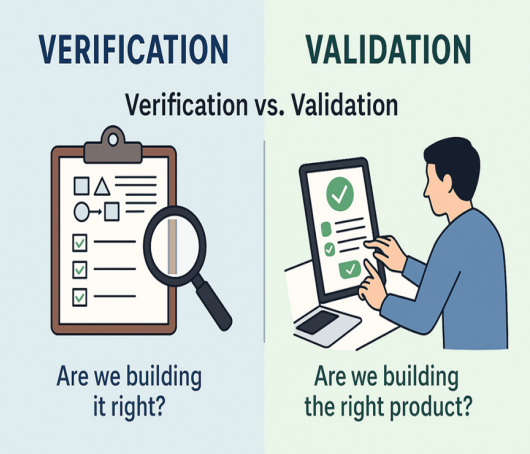


FIGURE 3.5 – validation Vs Vérification

Après la vérification, on passe à une phase de test qui consiste à rechercher des erreurs dans le programme.

Cette recherche peut se faire par examen ou analyse de test (analyse statique) ou alors par des exécutions sur un sous ensembles fini de données possibles (test dynamique).

Il existe plusieurs types de test dans le PDL :

— **Test unitaire** : tester des composants isolées.

— **Test d'intégration** : tester un ensemble de composants qui

viennent d'être rassemblées.

— **Test système** : tester le système sur son future environnement d'exploitation.

3.2.7 La maintenance

En génie logiciel, La maintenance de logiciel désigne les modifications apportées à un logiciel, après sa mise en œuvre, pour (voir Figure (3.6)) :

— Corriger les fautes,

— Améliorer l'efficacité

— Adapter celui-ci à un environnement modifié (Maintenance adaptative).

— Ou autres caractéristiques,

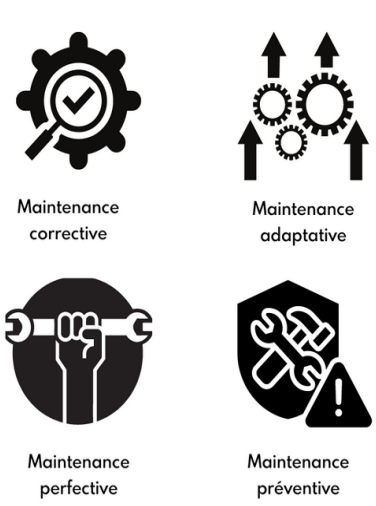


FIGURE 3.6 – Types de maintenance logiciel

Dans la maintenance, on peut distinguer :

— La maintenance adaptative consiste à faire évoluer une application logicielle lorsque son environnement change.

— La maintenance corrective : traiter les erreurs (bugs).

— La maintenance évolutive : intégration de nouveaux changements.

— La maintenance préventive : faciliter les opérations de maintenance à venir

3.3 Synthèse

Le cycle de vie du logiciel représente l'ensemble des activités organisées permettant de concevoir, développer, livrer et maintenir un produit logiciel. Le choix du modèle de développement dépend de plusieurs facteurs tels que la nature du produit, la taille de l'équipe, les contraintes temporelles et les aspects techniques.

Le processus de développement se décompose en plusieurs étapes fondamentales en commençant par l'analyse des besoins, qui consiste à comprendre le domaine d'application, identifier les contraintes et définir les attentes des utilisateurs. Le résultat de cette étape est le cahier des charges qui sera une entrée pour l'étape suivante.

La phase de spécification formalise ensuite les besoins en décrivant ce que doit faire le système, sans entrer dans les détails techniques. Cette étape est suivie par une étape de conception, qui transforme ces spécifications en une architecture logicielle détaillée.

L'implémentation correspond à la traduction de la phase de conception en code source dans un langage de programmation précis. Cette étape représente une part relativement limitée de l'effort global par rapport aux phases d'analyse et de conception.

La gestion de configuration et l'intégration permettent de maîtriser les différentes versions du logiciel et d'assembler les composants pour obtenir un système fonctionnel.

Les activités de vérification et de validation jouent un rôle crucial dans l'assurance qualité. La validation d'un logiciel consiste à vérifier si le système répond aux besoins des utilisateurs, tandis que la vérifi-

cation s'assure de la conformité du développement aux spécifications.

Enfin, la maintenance constitue une phase essentielle du cycle de vie, intervenant après la mise en service du logiciel. La maintenance peut être de nature corrective, adaptative, ou évolutive.

Chapitre 4: Les modèles de développement d'un logiciel

Table des matières

Différentes approches ont été proposées pour gérer les processus de développement associés à la production de logiciels.

Ces approches, appelées cycles de vie, permettent de rationaliser les activités qui interviennent tout au long du processus de développement et de mieux gérer les acteurs qui y participent.

Partant du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de conception, un des objectifs de la mise en place des cycles de vie est de détecter ces erreurs au plus tôt et ainsi de mieux maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts occasionnés, dans la mesure où ils prévoient des phases de vérification/validation "tôt" dans le cycle.

4.1 Le cycle de vie en " Cascade "

Le cycle de vie en " Cascade " est un modèle de gestion linéaire qui divise le processus de développement en phases de projet successives.

Il part du principe que la construction nécessite, en général, un enchaînement logique. Par exemple la couverture d'une maison ne peut pas être effectuée sans avoir préalablement fait les fondations.

Il définit une démarche de développement séquentiel où chaque phase conduit à la production d'un ou plusieurs livrables qui doivent être validés avant d'être utilisés lors de la phase suivante (une vérification doit être faite pour assurer la conformité de la solution retenue).

Chaque étape est reliée à l'étape suivante pour représenter l'enchainement, et à l'étape précédente pour représenter les corrections par un retour en arrière, voir Figure (4.1).

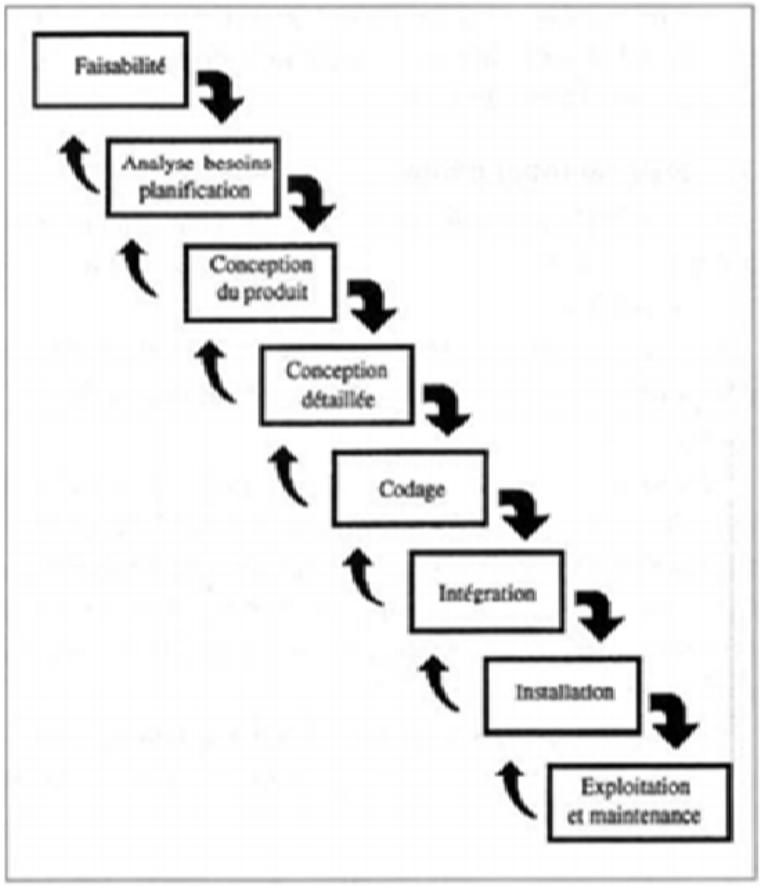


FIGURE 4.1 – Cycle de vie en cascade

Avantages du cycle de vie en cascade

— Plan simple, et facile à comprendre.

— Cette première définition a permis la normalisation des cadres conceptuels et terminologiques des différentes activités.

Inconvénients du cycle de vie en cascade

— Hypothèses souvent irréalistes que l'on peut dès le départ définir complètement et en détail.

— Ne reflète pas la façon dont le code est réellement développé.

— Manque de flexibilité pour les imprévus.

4.2 Le cycle de vie en " V "

Le cycle de vie en " V " 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, voir Figure (4.2).

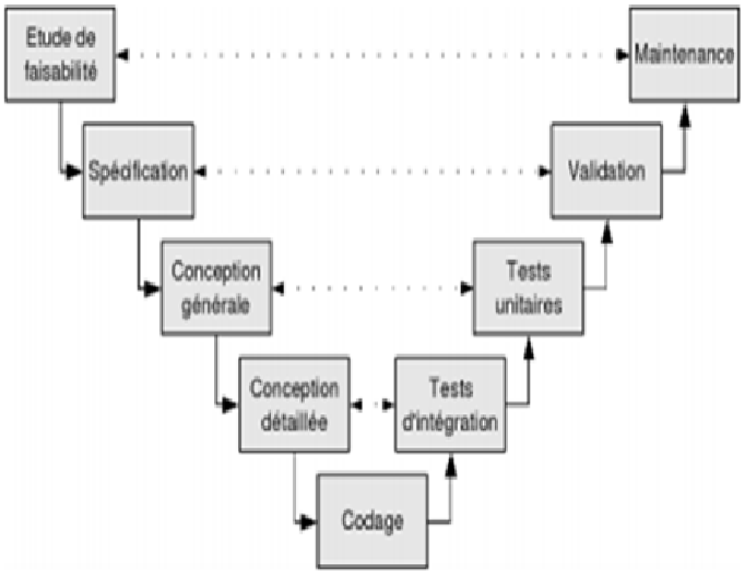


FIGURE 4.2 – Le cycle de vie en " V "

Cette méthode implique toutes les étapes du cycle de mise en œuvre d'un produit ou d'un logiciel. Elle permet de définir le processus d'un projet en neuf étapes regroupées en trois phases :

— La conception (partie descendante),

— La réalisation (codage et programmation),

— La validation (partie montante).

Les phases de la partie montante doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés afin d'améliorer le logiciel.

Il définit une démarche de développement séquentiel où chaque phase conduit à la production d'un ou plusieurs livrable(s) qui doivent

être validés avant d'être utilisés lors de la phase suivante (une vérification doit être faite pour assurer la conformité de la solution retenue).

Chaque étape est reliée à l'étape suivante pour représenter l'enchainement, et à l'étape précédente pour représenter les corrections par un retour en arrière.

Le cycle de vie en " V " permet de montrer comment les activités de «test» sont liées à celles de l'analyse et de conception». Néanmoins, il présente un manque de «feedback» et pas de résultats intermédiaires dont on peut discuter avec le client.

4.3 Le cycle de vie en spirale

Le cycle de vie en spirale est une approche itérative du cycle de développement en " V ". Ainsi, il en reprend l'essentiel des concepts en s'articulant autour de quatre phases importantes :

— La détermination des objectifs,

— La détermination des risques,

— Le développement et les tests,

— La planification de l'itération suivante.

Il est important d'évaluer correctement le risque à chaque itération sachant que toute erreur sera d'autant plus difficile à corriger que le développement sera avancé.

La Figure (4.3) permet de visualiser le modèle de cycle de vie en Spirale.

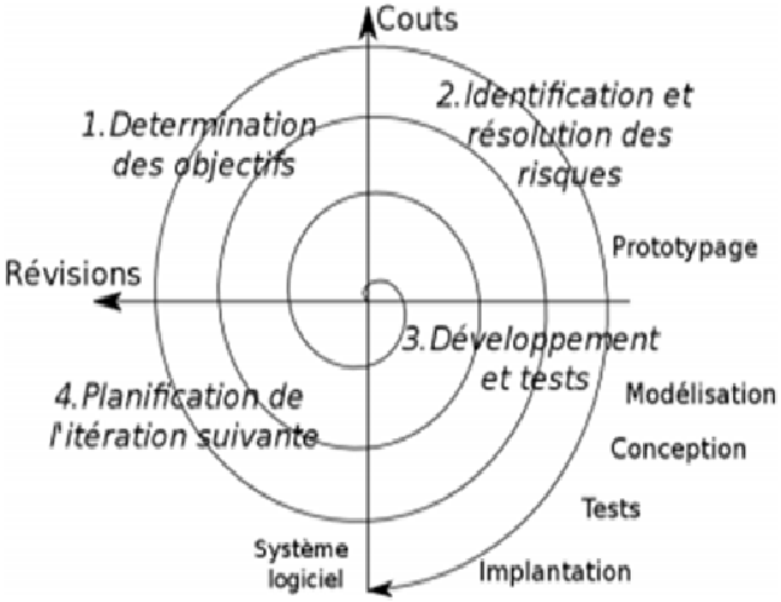


FIGURE 4.3 – Le cycle de vie en spirale

Le modèle de cycle de vie en spirale, permet d'établir le modèle de développement le plus adéquat en regard du risque, mais la mise en œuvre de ce modèle demande des compétences et un effort important.

4.4 Le modèle par prototypage

Le modèle de prototypage est souhaitable pour les projets où les besoins ne sont pas clairement définis et sont susceptibles de changer avec le temps.

À chaque étape, un ou plusieurs prototypes sont soumis au client pour évaluation/révision, voir Figure (4.4).

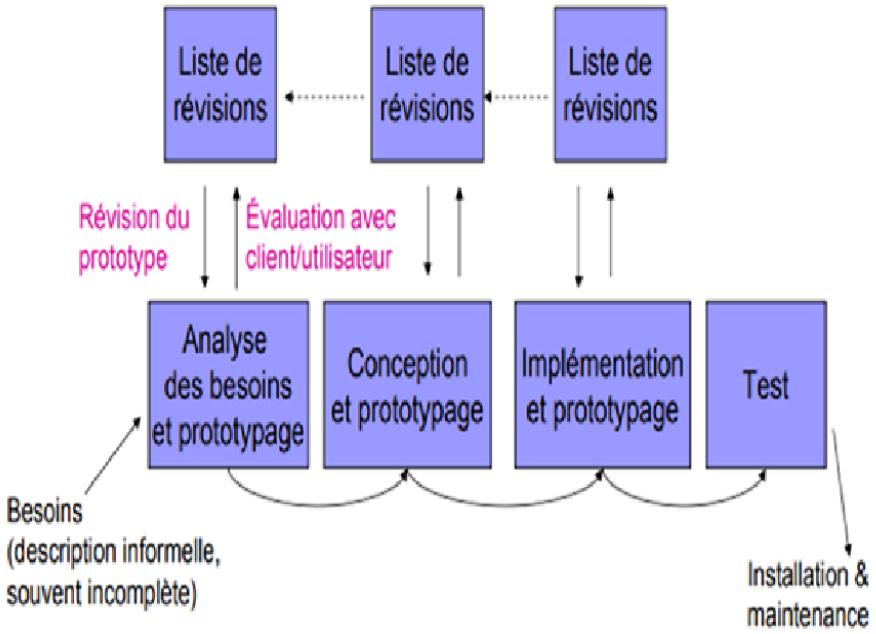


FIGURE 4.4 – Le modèle par prototypage

Ce modèle permet d'examiner et d'explorer certains aspects du système pour évaluer et choisir les meilleures stratégies/solutions.

Types de prototypes

— **Prototype jetable** : maquette exploratoire développée pour mieux comprendre les besoins du client, évaluer différentes solutions, etc. Développé rapidement et jeté ensuite.

— **Prototype évolutif (ou réutilisable)** : maquette destinée à être complétée/optimisée dans les prochaines étapes du développement jusqu'à l'obtention du produit final.

Le modèle par prototypage améliore la compréhension mutuelle du problème entre client, développeur et utilisateur. Aussi, il favorise les activités de vérification et de validation intermédiaires. Cependant, cette approche peut pousser à accélérer le processus et à s'arrêter à un prototype incomplet.

4.5 Le modèle par incrément

Le cycle par incrément est une approche incrémentale qui se base sur la division du système en sous-systèmes (un par fonctionnalité), voir Figure (4.5).

Première version : système partiel.

Chaque nouvelle version (incrément) ajoute une nouvelle fonctionnalité.

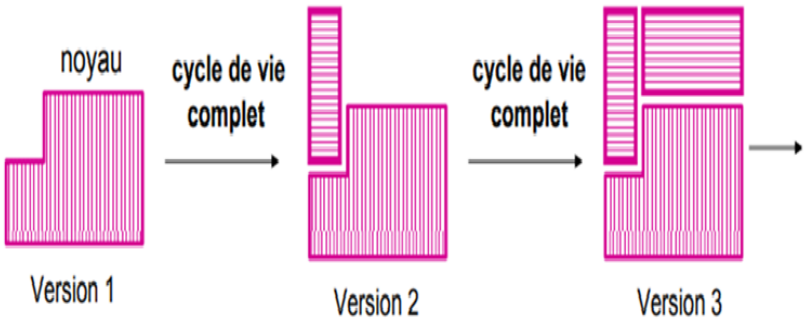


FIGURE 4.5 – Le modèle par incrément

Cette approche présente le risque de la remise en cause du noyau (fonctionnalités de base) au cours du développement. Néanmoins, elle permet de :

— Produire des versions partielles du système.

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

— Ajouter graduellement de qualités et fonctionnalités.

4.6 synthèse

Les modèles de développement logiciel, également appelés cycles de vie, constituent des approches méthodologiques permettant d'or-

ganiser, planifier et maîtriser les différentes phases du processus de développement. Leur objectif principal est d'améliorer la qualité du logiciel, de réduire les coûts et de détecter les erreurs le plus tôt possible.

La table (4.1) présente une comparaison entre les différents modèles du processus de développement du logiciel.

TABLE 4.1 – Comparaison des modèles de développement logiciel

Modèle	**Avantages**	**Inconvénients**
Cascade	Simple, Structuré, Facile à comprendre	Rigide, Peu flexible, Difficile de revenir en arrière
Modèle en V	Bonne qualité, Validation à chaque étape	Peu flexible, Peu de feedback utilisateur
Spirale	Gestion des risques, Grande flexibilité	Complexe, Coûteux, Besoin d'expertise
Prototypage	Clarifie les besoins, Améliore la communication	Risque de prototype incomplet, Dérive du projet
Incrémental	Livraison rapide, Détection précoce des erreurs	Risque d'incohérence globale

Le modèle en cascade est une approche séquentielle où chaque phase doit être terminée et validée avant de passer à la suivante. Simple et structuré, il facilite la compréhension et la gestion du projet, mais il manque de flexibilité et s'adapte mal aux changements en cours de développement.

Le modèle en V constitue une évolution du modèle en cascade, avec une correspondance entre les phases de conception et de test et validation. Ce modèle renforce l'assurance qualité, mais reste rigide et complexe avec peu de retours intermédiaires avec le client.

Le modèle en spirale adopte une approche itérative centrée sur la

gestion des risques. Chaque cycle comprend la définition des objectifs, l'analyse des risques, le développement et la planification. Bien qu'il soit puissant dans la gestion des risques avec une grande flexibilité, il est couteux et nécessite une expertise importante et des ressources conséquentes.

Le modèle par prototypage est particulièrement manipulé lorsque les besoins sont mal définis. Il repose sur la réalisation de maquettes (prototypes) permettant de mieux comprendre les attentes des utilisateurs. Il favorise la communication pour clarifier les besoins avec une validation progressive. Néanmoins, ce modèle peut conduire à des solutions incomplètes ou mal structurées.

Le modèle incrémental consiste à développer le système avec des versions successives, où chaque version permet d'ajouter de nouvelles fonctionnalités. Il assure une livraison rapide de versions partielles et une détection précoce des erreurs, mais peut poser des problèmes de cohérence globale du système.

Le choix du modèle de développement dépend du contexte du projet, notamment de la stabilité des besoins, des contraintes techniques et des ressources disponibles. Chaque modèle présente des avantages et des limites, mais leur combinaison peut être nécessaire pour répondre efficacement aux exigences du projet.

Chapitre 5: Conception Orienté Objet

Table des matières

5.1 La terminologie Orientée Objet

La terminologie orientée objet est issue des réseaux sémantiques et des langages de programmation.

Les réseaux sémantiques ont introduit l'idée de représenter les connaissances sous forme de noeuds (objets) reliés par des relations, ce qui a produit la notion d'objet en programmation. Chaque objet regroupe à la fois des données (attributs) et des comportements (méthodes).

Parallèlement, les langages de programmation ont évolué vers une approche plus structurée, permettant de modéliser des systèmes complexes en utilisant des entités autonomes et réutilisables.

Ainsi, la Programmation Orientée Objet (POO) repose sur cette convergence entre représentation des connaissances et évolution des langages, donnant naissance à des concepts clés comme : classe, objet, encapsulation, héritage et polymorphisme.

5.1.1 Qu'est ce qu'un Objet ?

Un objet est abstraction informatique d'une entité du monde réel caractérisée par une identité (identifiant), un état, et un comportement.

Exemple : objet de type véhicule V1 peut être représenté par un groupe de valeurs nommées avec un comportement associé, voir Figure (5.1).

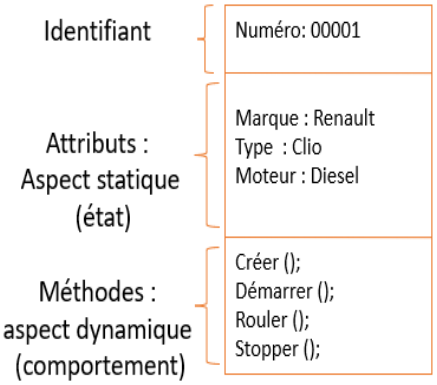


FIGURE 5.1 – Objet de type véhicule

Un identifiant d'objet (IDO) : est une référence système unique et invariante attribuée à un objet lors de sa création permettant de le désigner et de le retrouver tout au long de sa vie.

Un attribut : est une caractéristique d'un objet désignée par un nom permettant de mémoriser une ou plusieurs valeurs.

Encapsulation : le modèle à objet permet d'encapsuler les structures des objets par des opérations appelées méthodes.

5.1.2 Définition d'une classe

Une classe est un ensemble de d'objets ayant les mêmes propriétés (attributs + méthodes).

Exemple :

La Figure (5.2) est un illustration de la Classe Étudiant : identifiant, attributs, méthodes

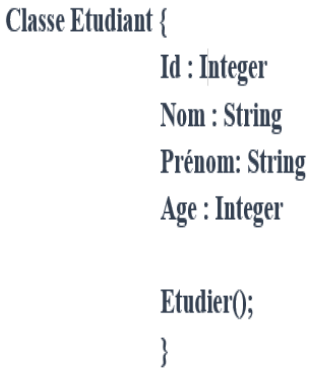


FIGURE 5.2 – Classe Étudiant

Un élément de la classe constitue une instance de la classe (la classe est un moule, et l'objet est le remplissage de ce moule).

Instance de la classe Etudiant [Mohamed, Ali, Sihem, ...]

5.1.3 Lien d'héritage

Un lien d'héritage est la possibilité de définir une nouvelle classe par affinage d'une classe plus générale.

Définition 1 : Lien hiérarchique entre deux classes spécifiant que les objets de la classe supérieure sont plus généraux que ceux de la classe inférieure.

Classe supérieure = super classe ou classe de base.

Classe inférieure = sous-classe ou classe dérivée.

Le parcours du lien de la super classe vers la sous classe correspond à une spécialisation.

Une sous classe reprend les propriétés (attributs + méthodes) des classes plus générales (héritage).

Définition 2 : transmission automatique d'une propriété d'une classe de base vers une sous classe.

Exemple : voir Figure (5.3).

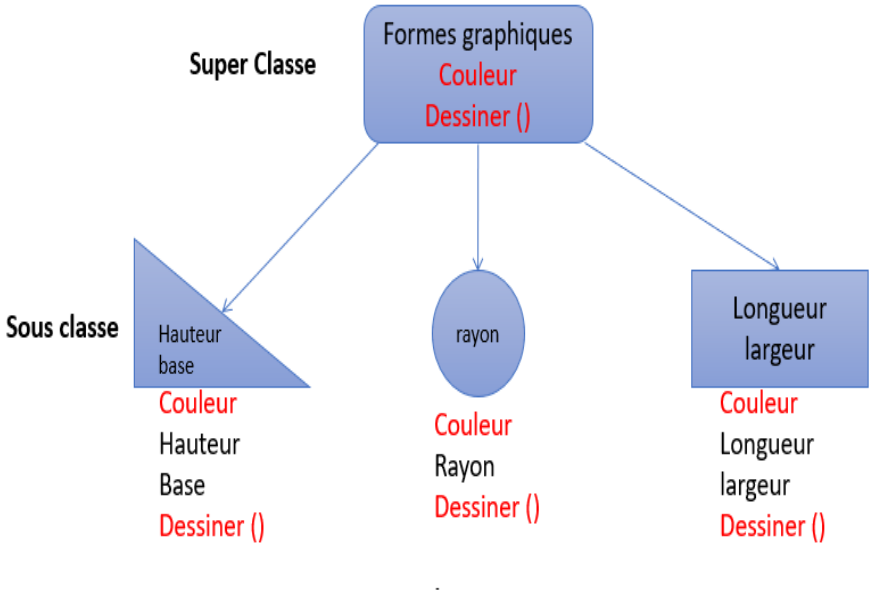


FIGURE 5.3 – Graphe d'héritage

5.1.4 Héritage multiple (polymorphisme)

Le polymorphisme est un type d'héritage dans le quel une classe dérivée hérite de plusieurs classes de niveaux immédiatement supérieur.

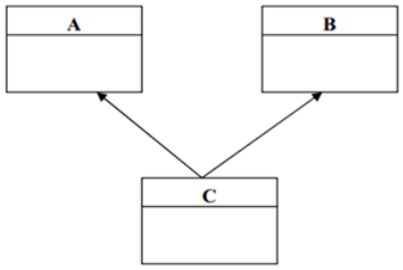


FIGURE 5.4 – Héritage multiple

5.2 Unified Modified Language (UML)

Pour faire face à la complexité croissante des systèmes d'information, de nouvelles méthodes et outils ont été créées.

La principale avancée réside dans la Programmation Orientée Objet (P.O.O).

Face à ce nouveau mode de programmation, les méthodes de modélisation classique (telle MERISE) ont rapidement montré certaines limites.

De très nombreuses méthodes ont également vu le jour comme Booch, OMT ... L'Object Management Group (OMG) a eu comme objectif de définir une notation standard utilisable dans les développements informatiques basés sur l'objet.

C'est ainsi qu'est apparu UML (Unified Modified Language) langage de modélisation objet unifié, qui est issu de la fusion des méthodes Booch, OMT (Object Modelling Technique) et OOSE (Object Oriented Software Engineering).

UML est en effet apparu très tardivement, car l'approche objet se pratique depuis de très nombreuses années déjà. Vers la fin 1997, UML est devenu une norme OMG (Object Management Group).

UML permet de définir et de visualiser un modèle, à l'aide de diagrammes.

Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle.

Chaque type de diagramme UML possède une structure (les types des éléments de modélisation qui le composent sont prédéfinis).

Les différents types de diagrammes UML combinés offrent une vue complète des aspects statiques et dynamiques d'un système.

Par extension et abus de langage, un diagramme UML est aussi un modèle car un diagramme modélise un aspect du modèle global.

Il existe 2 types de vues du système qui comportent chacune leurs propres diagrammes, voir Figure (5.5).

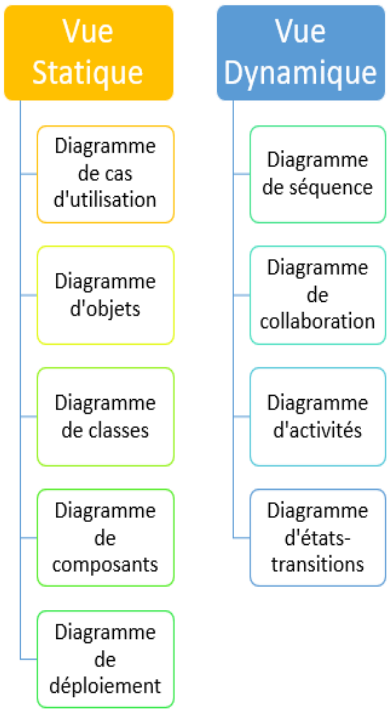


FIGURE 5.5 – Classification des diagrammes UML

Lors de la conception d'un logiciel, on commence généralement par comprendre et analyser les besoins des utilisateurs. Cette étape est exprimée en langage naturel (texte, cahier des charges). UML intervient pour structurer ces besoins sous forme de modèles graphiques standardisés, compréhensibles par les concepteurs et les développeurs, voir Figure (5.6).

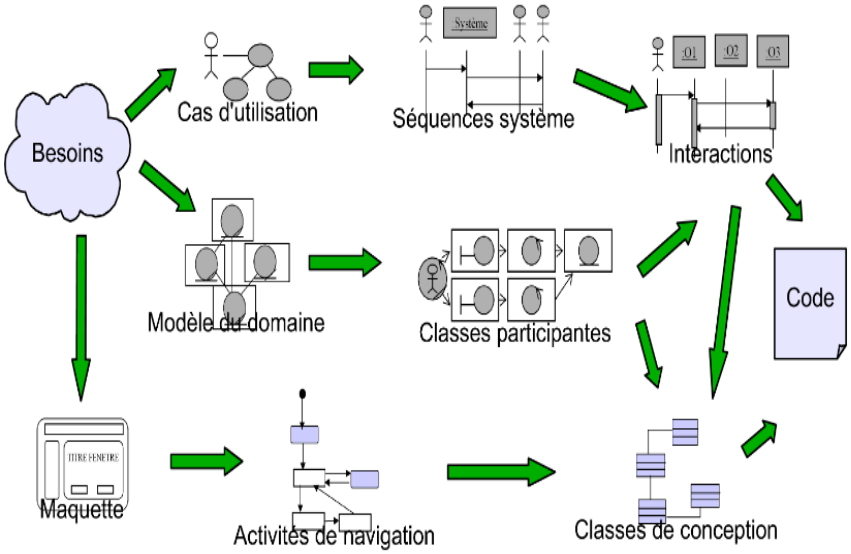


FIGURE 5.6 – schéma synthétique UML

UML agit comme un pont entre l'analyse des besoins et le code final, garantissant une meilleure traçabilité et cohérence du système, en passant progressivement par des modélisation intermédiaires.

5.2.1 Diagramme cas d'utilisation (use case)

Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation, voir Figure (5.7).

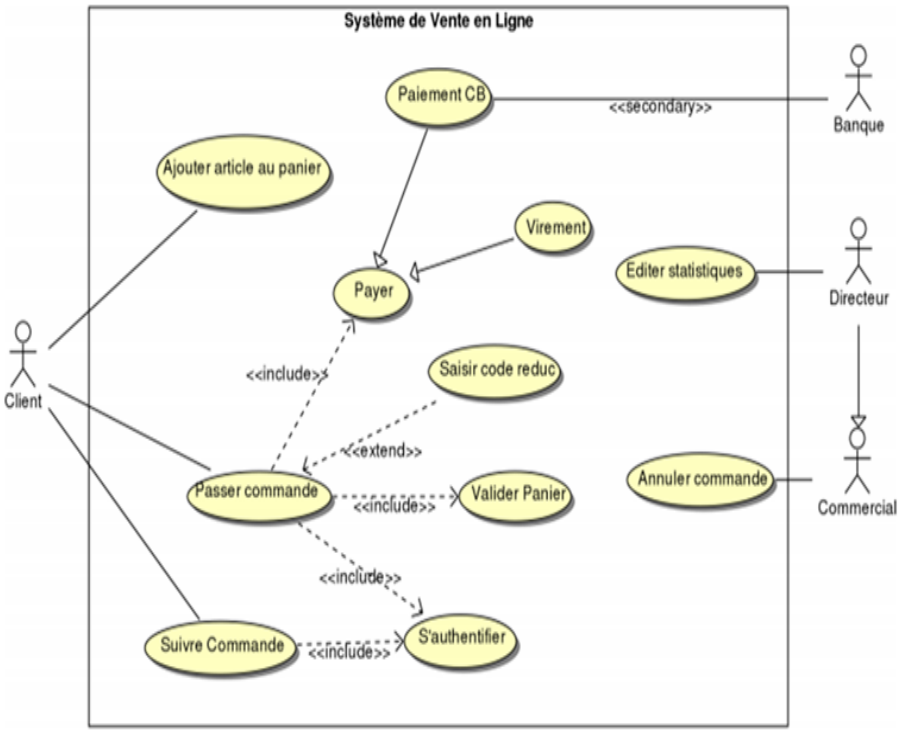


FIGURE 5.7 – Exemple de diagramme de cas d'utilisation

Un cas d'utilisation est un service rendu à l'utilisateur, il implique des séries d'actions plus élémentaires, voir Figure(5.8).

FIGURE 5.8 – Représentation d'un cas d'utilisation

Un acteurs est une entité extérieure au système modélisé, et qui interagit directement avec lui, voir Figure(5.9).

FIGURE 5.9 – Représentation d'un acteur

Un cas d'utilisation est l'expression d'un service réalisé de bout en

bout, avec un déclenchement, un déroulement pour l'acteur qui l'initie, voir Figure(5.10).

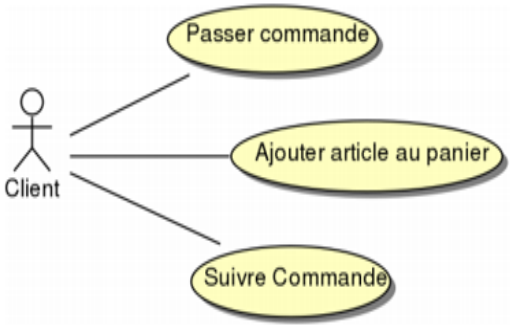


FIGURE 5.10 – Représentation d'un service par un diagramme use case

Inclusion : le cas A inclut le cas B (B est une partie obligatoire de A), voir Figure (5.11 (a)).

Extension : le cas B étend le cas A (B est une partie optionnelle de A), voir Figure (5.11 (b)).

Généralisation : le cas A est une généralisation du cas du cas B (B est une sorte de A), voir Figure (5.11 (c)).

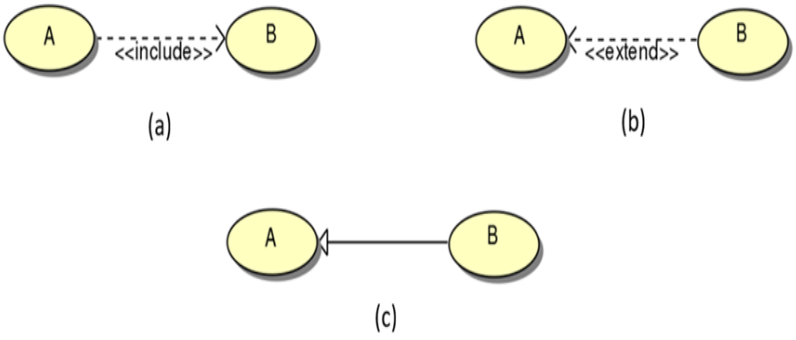


FIGURE 5.11 – Inclusion, Extension, Généralisation

Les inclusions et les extensions sont représentées par des dépendances.

Lorsqu'un cas B inclut un cas A, B dépend de A.

Lorsqu'un cas B étend un cas A, B dépend aussi de A.

On note toujours la dépendance par une èche pointillée B —> A qui se lit « B dépend de A ».

Lorsqu'un élément A dépend d'un élément B, toute modification de B sera susceptible d'avoir un impact sur A.

Les « include » et les « extend » sont des stéréotypes des relations de dépendance.

Les relations entre cas permettent la réutilisabilité du cas « s'authentifier » : il sera inutile de développer plusieurs fois un module d'authentification.

5.2.2 Diagramme de classes

Une classe est une représentation abstraite d'un d'ensemble d'objets, elle contient les informations nécessaires à la construction de l'objet (c'est-à-dire la définition des attributs et des méthodes), voir Figure (5.12).

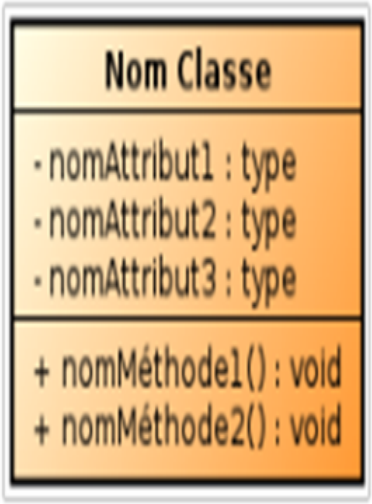


FIGURE 5.12 – Représentation d'une classe

Le diagramme des classes est un diagramme structurel (statique) qui permet de représenter :

— Les classes (attributs + méthodes)

— Les associations (relations) entre les classes.

Le diagramme de classes est le plus important des diagrammes UML. Il est obligatoire lors de la modélisation objet d'un système.

Si la modélisation ne s'intéresse qu'aux relations entre les différentes classes du système (et pas au contenu des classes), nous pouvons ne pas représenter les attributs et les méthodes de chaque classe (nous ne mettons rien dans le deuxième et troisième compartiment).

Nous pouvons détailler la classe en indiquant :

— La visibilité (encapsulation) des méthodes et des attributs.
  — *public* : élément non encapsulé visible par tous.
  — *private* : élément encapsulé visible seulement dans la classe.
  — *protected* : élément encapsulé visible dans la classe et dans les sous-classes.
  — *package* : élément encapsulé visible dans les classes du même paquetage.

— Le type de chaque attribut.

— La signature de chaque méthode.

— Le type de valeur retournée par chaque méthode.

La multiplicité (ou cardinalité) des attributs

La multiplicité indique le nombre de valeur que l'attribut peut contenir (l'attribut est souvent un tableau de valeurs, statique ou dynamique), voir la Figure (5.13).

Cardinalité	Signification
0..1	Zéro ou une fois
1..1 (ou 1)	Une et une seule fois
0..* (ou *)	De zéro à plusieurs fois
1..*	De une à plusieurs fois
m..n	Entre **m** et **n** fois
n..n (ou n)	**n** fois

FIGURE 5.13 – La multiplicité des attributs

La multiplicité se note entre crochets après le type de valeur que contient l'attribut, voir Figure (5.14).

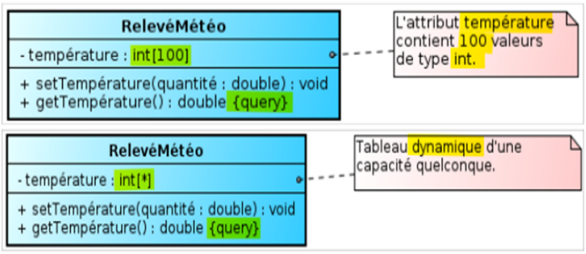


FIGURE 5.14 – Exepmle de multiplicité des attributs

Les relations entre classes

1. La relation de dépendance : La dépendance est la forme la plus faible de relation entre classes. Une dépendance entre deux classes signifie que l'une des deux utilise l'autre.

Elle est représentée par un trait discontinu orienté, reliant les deux classes. La dépendance est souvent stéréotypée « use » pour mieux expliciter le lien sémantique entre les éléments du modèle.

2. Les associations : Cette relation est plus forte. Elle indique qu'une classe est en relation avec une autre pendant un certain laps de temps. La ligne de vie des deux objets concernés ne sont cependant pas associés étroitement (un objet peut être détruit sans que l'autre le soit nécessairement).

L'association est représentée par un simple trait continu, reliant les deux classes. Le fait que deux instances soient ainsi liées permet la navigation d'une instance vers l'autre, et vice versa (chaque classe possède un attribut qui fait référence à l'autre classe).

3. La cardinalité (ou multiplicité) : Les associations sont typiquement destinées à représenter des relations durables entre des classes ; elles sont souvent utilisées pour représenter les attributs de la classe.

Comme nous l'avons vu pour un attribut, il est possible d'utiliser la multiplicité pour indiquer le nombre d'instances (d'une classe donnée) qui sont impliquées dans la relation, voir Figure (5.15).

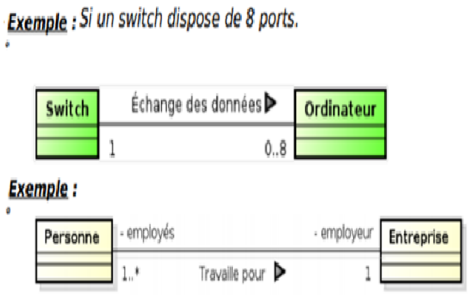


FIGURE 5.15 – Cardinalité entre classes

La cardinalité indique le nombre d'instances de classe étant en relation avec la classe situé à l'autre extrémité de l'association. En l'absence de spécification, la cardinalité vaut 1.

4. Association réflexive (ou récursive) : Une association qui relie une classe avec elle-même est une association réflexive, voir Figure (5.16).

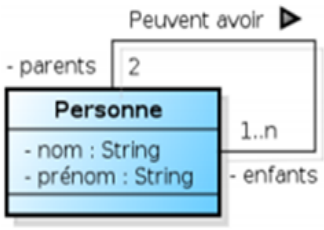


FIGURE 5.16 – Association réflexive

5. Associations reliant plusieurs classes (association n-aire) : l'association n-aire est une association qui relie plus de 2 classes entre elles. L'association n-aire se représente par un losange d'où part un trait allant à chaque classe.

L'association n-aire est imprécise, difficile à interpréter et souvent source d'erreur, elle est donc très peu utilisée. La plupart du temps,

elle est vite remplacée par un ensemble d'associations binaires afin de lever toute ambiguïté, voir Figure (5.17).

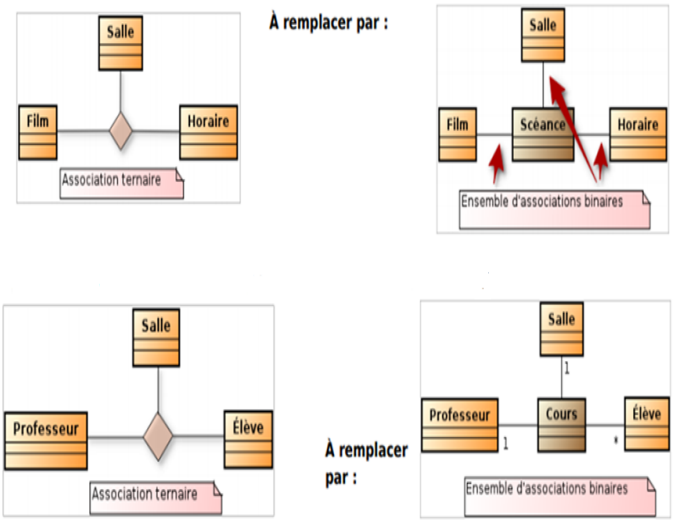


FIGURE 5.17 – Association n-aire

6. La composition : indique qu'un objet A (appelé conteneur) est constitué d'un autre objet B. Cet objet A n'appartient qu'a l'objet B et ne peut pas être partagé avec un autre objet. C'est une relation très forte, si l'objet A disparaît, alors l'objet B disparaît aussi.

Exemple : Un cheval possède une tête et 4 jambes. Elle se représente par un losange plein du coté de l'objet conteneur.

7. L'agrégation : indique qu'un objet A possède un autre objet B, mais contrairement à la composition, l'objet B peut exister indépendamment de l'objet A. La suppression de l'objet A n'entraîne pas la suppression de l'objet B. L'objet A est plutôt à la fois possesseur et utilisateur de l'objet B.

Exemple :Un cheval possède une selle sur son dos. Elle se représente par un losange vide du coté de l'objet conteneur.

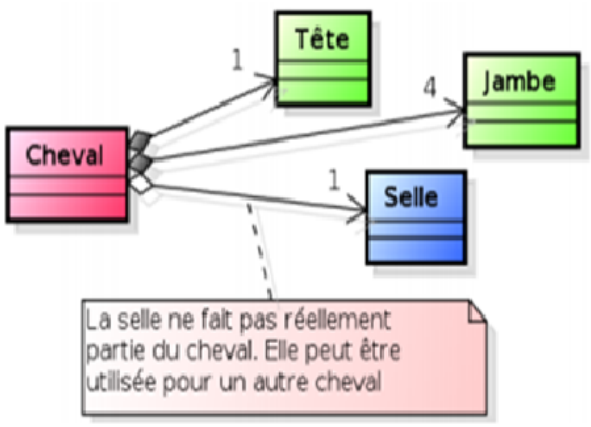


FIGURE 5.18 – Agrégation et Composition

5.2.3 Diagramme d'objet

Le diagramme d'objet est un diagramme instance, c'est un exemple de diagramme de classe avec les objets instanciés à un moment donné.

Un diagramme d'objets se concentre sur les attributs d'un ensemble d'objets et sur la façon dont ils interagissent les uns avec les autres.

Les objets sont relié par des liens, instance des associations entre les classes qui donnent le sens aux objets considérés.

Les liens entre objets représentent les connexions entre les instances de classe à un instant donné.

Autrement dit, un diagramme de classe représente une situation générale alors qu'un diagramme d'objet, représente une situation particulière, voir Figure (5.19).

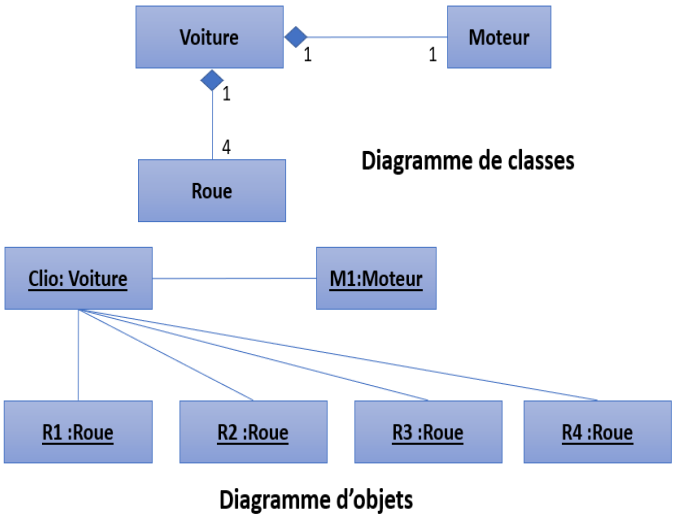


FIGURE 5.19 – Exemple d'un diagramme d'objet

Une instance réflexive peut relié un objet à lui-même. Dans ce cas le lien est représenté par une boucle portée par l'objet en question.

5.2.4 Diagramme de composants

Le diagramme de composants, voir Figure (5.20), permet de présenter :

— Les différents composants d'un système (modules, bibliothèques, programmes, services...),

— Les relations et dépendances entre ces composants,

— La manière dont ils interagissent pour former le système global.

Un diagramme de composants est composé de :

— Composant : partie du système (ex : base de données, interface utilisateur, module de calcul...)

— Interface : point de communication entre composants

— Dépendance : relation montrant qu'un composant utilise un autre

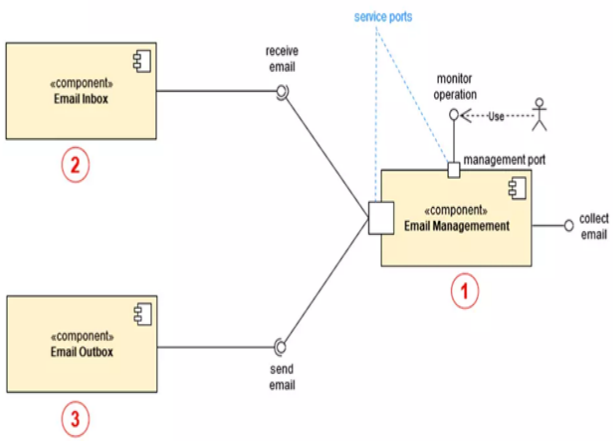


FIGURE 5.20 – Principaux facteurs influençant la qualité du logiciel

5.2.5 Diagramme de déploiement

Le diagramme de déploiement décrit la répartition des programmes (logiciels) sur les différents matériels (serveurs, ordinateurs, appareils mobiles, etc.) ainsi que les connexions entre eux.

Il vise les objectifs suivants :

— Visualiser l'organisation matérielle d'un système

— Montrer où sont installées les applications

— Illustrer les communications entre les équipements

— Aider à comprendre l'infrastructure technique

Il est composé de :

— Nœuds (Nodes) :Représentent les équipements matériels ou les environnements d'exécution

Exemple : serveur, ordinateur, smartphone

— Composants logiciels : Programmes ou applications installés sur les nœuds

— Connexions : Liens qui montrent la communication entre les nœuds (réseau, internet, etc.)

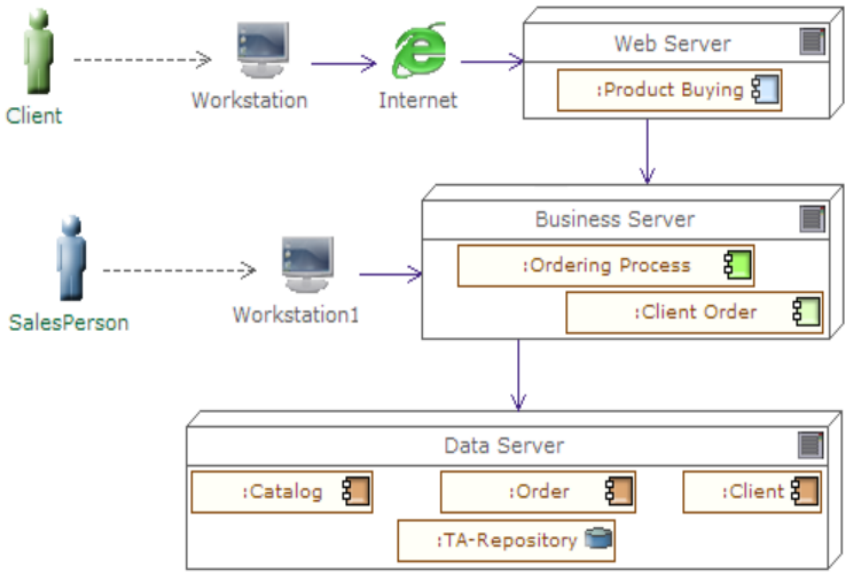


FIGURE 5.21 – Diagramme de déploiement

5.2.6 Diagramme de séquences

Un diagramme de séquences appelé aussi diagramme de scénarios montre les interactions entre les objets (communication).

La représentation de ce diagramme se concentre sur la séquence des interactions d'un point de vu temporel. Le diagramme de séquences met l'accent sur l'aspect temporel.

Les diagrammes de séquences permettent la description : COMMENT les éléments du système interagissent entre eux et avec les acteurs ?

Les objets au cœur d'un système interagissent en s'échangent des messages.

Les acteurs interagissent avec le système au moyen d'IHM (Interfaces Homme-Machine).

La dynamique étudie comment les objets communiquent ensemble, et l'effet de cette communication sur l'état de l'objet. La communication se fait en envoyant ou en recevant des messages. Un message

correspond à un appel d'une opération de l'objet cible du message, voir Figure (5.22).

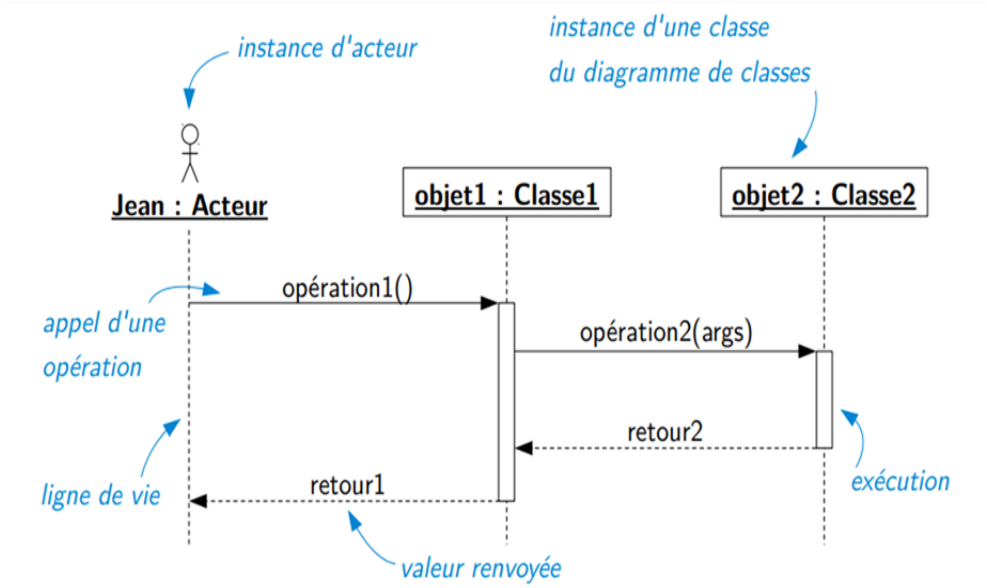


FIGURE 5.22 – Diagramme de séquence

Le diagramme de séquence est une représentation graphique de la chronologie des échanges de messages avec le système ou au sein du système.

— La vie de chaque entité représentée verticalement

— Échanges de messages représentés horizontalement

Le diagramme de séquence fait entrer en action les instances de classes intervenant dans la réalisation d'un cas d'utilisation particulier.

A chaque objet est associé une ligne de vie (en trait pointillés à la verticale de l'objet) qui peut être considéré comme un axe temporel (le temps s'écoule du haut vers le bas).

La ligne de vie indique les périodes d'activité de l'objet (généralement, les moments ou l'objet exécute une de ses méthodes). Lorsque l'objet est détruit, la ligne de vie s'achève par une croix, voir Figure (5.23).

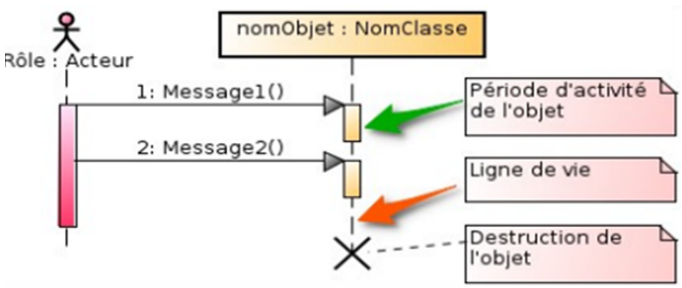


FIGURE 5.23 – La ligne de vie dans un diagramme de séquence

Un message définit une communication particulière entre des lignes de vie. Ainsi, un message est une communication d'un objet vers un autre objet. La réception d'un message est considérée par l'objet récepteur comme un événement qu'il faut traiter (ou pas). Plusieurs types de messages existent :

— L'invocation d'une opération : message synchrone (appel d'une méthode de l'objet cible).

— L'envoi d'un signal : message asynchrone (typiquement utilisé pour la gestion événementielle).

— La création ou la destruction d'une instance de classe au cours du cycle principal.

5.2.7 Diagramme de collaboration

Les diagrammes de collaboration constituent une extension du diagramme d'objets.

Le diagramme de collaboration fait partie des vues dynamiques du système d'information.

Les diagrammes de collaboration montrent les interactions et les liens entre une collection d'objets.

Ils montrent comment les messages sont échangés entre les objets liés et dans quel ordre ces messages se font, à travers la représentation chronologique d'envois de messages, mais le temps n'est pas repré-

senté implicitement. La chronologie des interactions est indiquée par la numérotation de messages pour indiquer leur ordre d'envoi.

Un diagramme de collaboration peut être utilisé pour montrer l'exécution d'une opération ou d'un scénario d'interaction dans le système.

Ce diagramme est équivalent au diagramme de séquences. Néanmoins, l'aspect temporel n'apparaît pas, mais l'aspect chronologique est présent. Le diagramme de collaboration représente ces éléments réalisant un objectif particulier (une fonctionnalité du système).

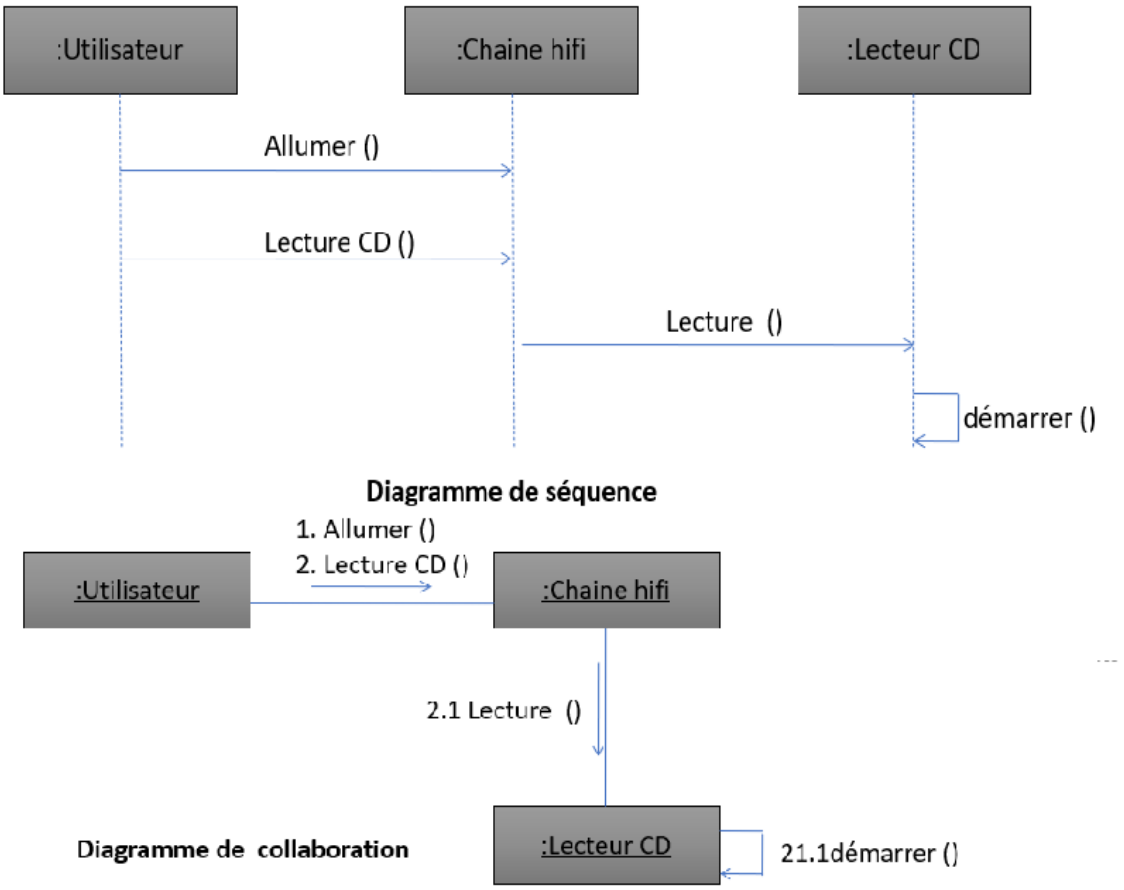


FIGURE 5.24 – Diagramme de collaboration

5.2.8 Diagramme d'activités

Le diagramme d'activité est utilisé pour modéliser les aspects dynamiques d'un système. Il s'agit de représenter les opérations d'un processus et leurs conséquences sur les objets .

La modélisation peut être utilisée pour décrire le déroulement d'un cas d'utilisation ou d'une méthode.

Les diagrammes d'activité affichent le flux de travail d'un point de départ à un point d'arrivée en détaillant les nombreux chemins de décision existant dans la progression des événements contenus dans l'activité.

Ils peuvent être utilisés pour détailler des situations dans lesquelles un traitement parallèle peut avoir lieu lors de l'exécution de certaines activités.

Un diagramme d'activités visualise un graphe d'activités qui modélise le comportement interne d'une méthode, d'un cas d'utilisation ou plus généralement d'un processus de traitement.

Une activité définit un comportement décrit par un séquencement organisé d'unités dont les éléments simples sont les actions.

Le diagramme d'activités comprend :

— **Des activités** (une étape d'exécution) : Une activité représente une exécution d'un mécanisme, un déroulement d'étapes séquentielles. Le passage d'une activité vers une autre est matérialisé par une transition.

— **Des transitions** : qui sont automatiques entre activités. Il est inutile également de préciser les événements. Les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre.

La Figure (5.25) présente un exemple de diagramme d'activité.

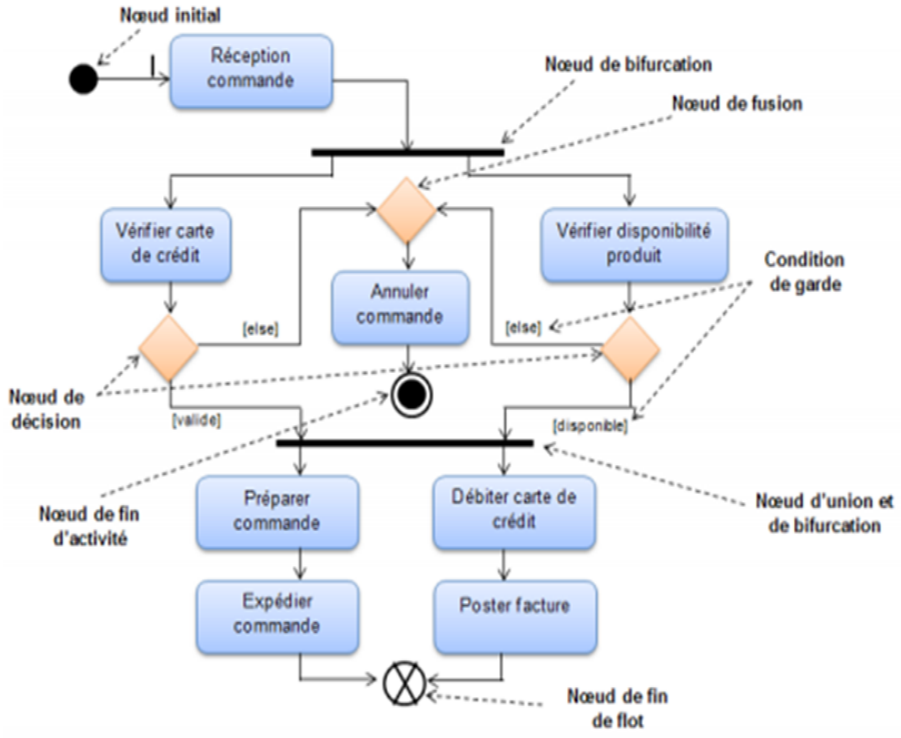


FIGURE 5.25 – Diagramme d'activité

5.2.9 Diagramme d'états-transitions

Un diagramme d'états-transitions (appelé aussi diagramme d'états ou State Machine Diagram en UML) est un diagramme qui permet de représenter le comportement dynamique d'un système en montrant les différents états possibles d'un objet et les transitions entre ces états.

— **État** : Il représente une situation ou une condition dans laquelle peut se trouver un objet. Il est représenté par un rectangle à coins arrondis.

Exemple d'état : En attente, En cours, Terminé.

— **Transition** : Elle représente le passage d'un état à un autre. Elle est représentée par une flèche, et déclenchée par un événement.

Exemple de transition : Le passage de l'état En attente vers l'état En cours.

Le schéma de la Figure (5.26) présente un exemple de diagramme d'état transition pour une montre à cadran numérique.

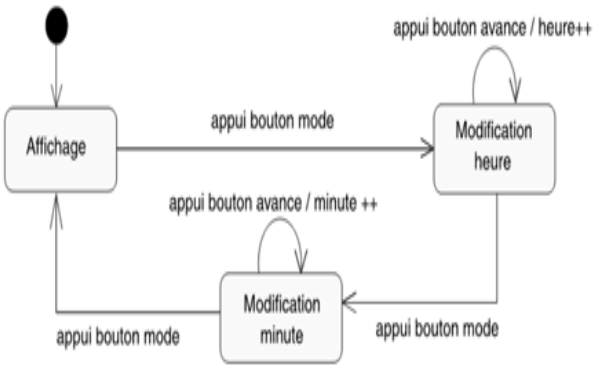


FIGURE 5.26 – Diagramme d'état-transition

5.3 Synthèse

La conception orientée objet est une approche de modélisation et de développement des systèmes informatiques basée sur la représentation du monde réel sous forme d'objets. Un objet est caractérisé par une identité, un état (attributs) et un comportement (méthodes). Les objets sont regroupés en classes, qui constituent des modèles définissant leurs propriétés communes.

La Figure (5.27) illustre une carte mentale pour la modélisation orienté objet en utlisant UML.

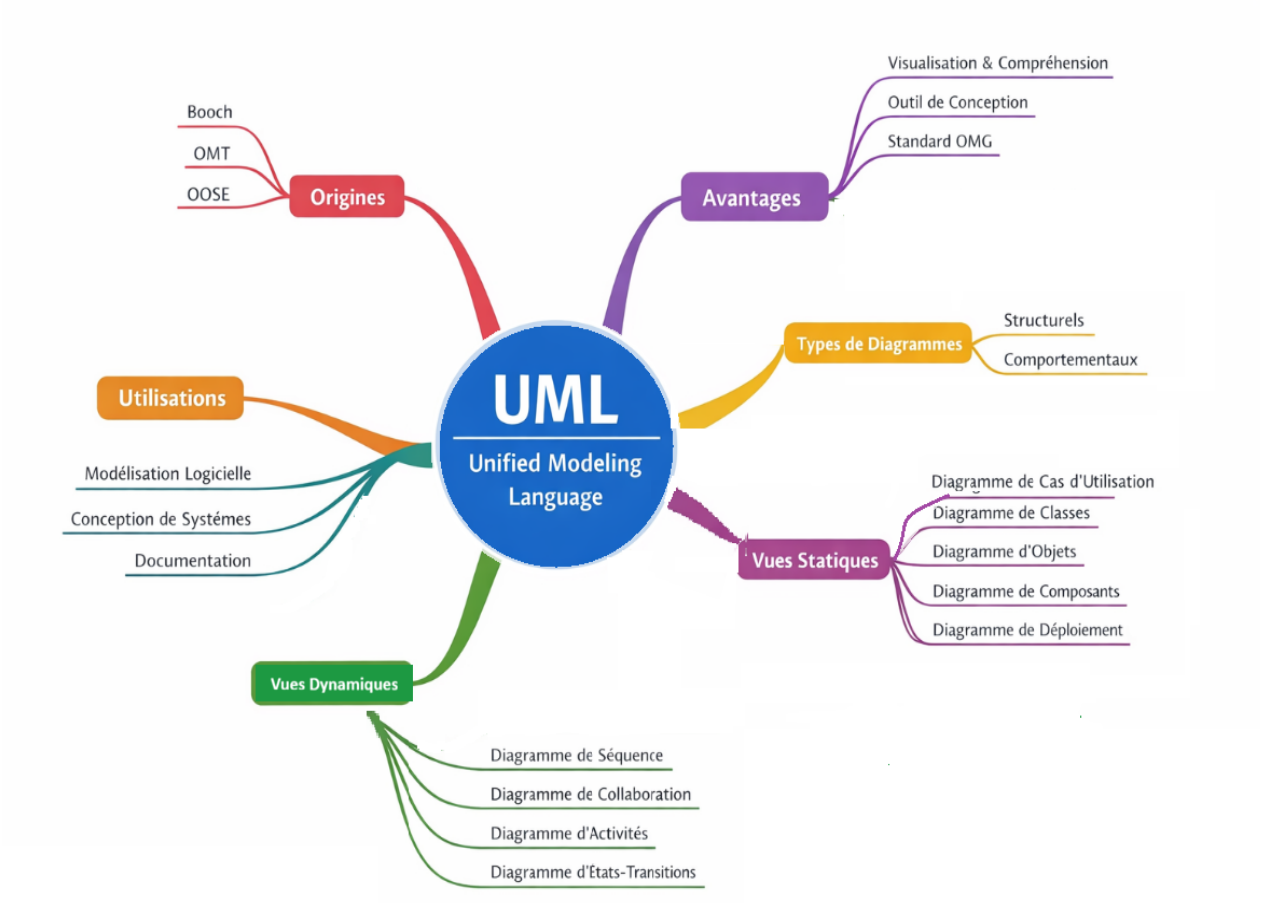


FIGURE 5.27 – Carte mentale UML

Pour modéliser les systèmes orientés objet, le langage UML (Unified Modeling Language) fournit une notation standard permettant de représenter les aspects statiques et dynamiques d'un système à l'aide de diagrammes structurels et comportementaux.

Les diagrammes UML se divisent en deux grandes catégories :

— Les diagrammes statiques : décrivent la structure du système (diagramme de cas d'utilisation, de classes, d'objets, de composants et de déploiement).

— Les diagrammes dynamiques : décrivent le comportement du système et permettent de représenter les interactions entre objets au cours du temps (diagramme de séquence, de collaboration,

d'activités et d'états-transitions).

Ainsi, la conception orientée objet en utilisant UML offre une approche structurée, modulaire et évolutive permettant de mieux comprendre, concevoir et maintenir les systèmes logiciels complexes.

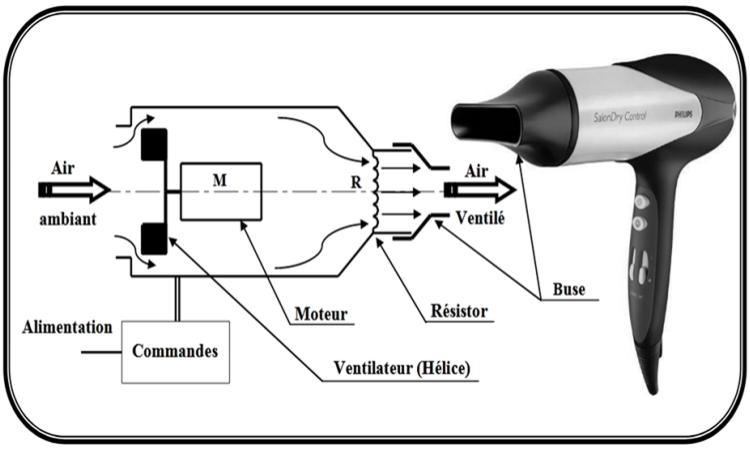
Partie II : Séries d'exercices et Examens types

Cette partie présente des séries d'exercices et examens types pour assimiler les concepts abordés dans le cours et développer les compétences analytiques et méthodologiques chez les étudiants.

Série d'exercices 1

Exercice 1

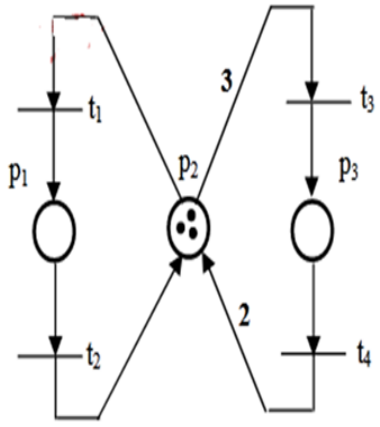
Soit le système technique d'un sèche cheveux suivant.



Donnez la modélisation du système qui correspond.

Exercice 2

Soit le RdP suivant .



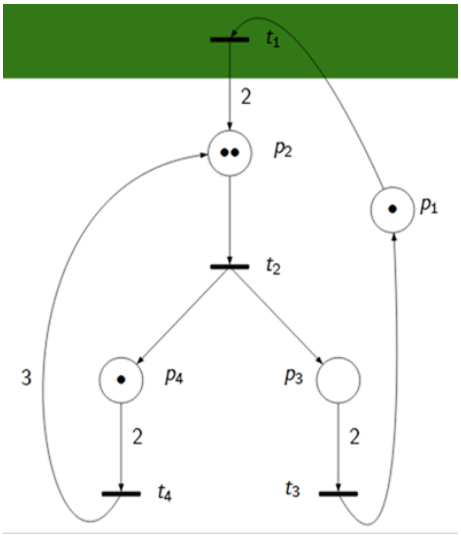
— Indiquer le marquage initial

— Etablir la matrice d'entrée E

— Etablir la matrice de sortie S.

Exercice 3

Soit le RdP suivant :



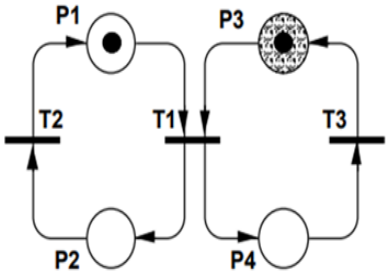
— Indiquer le marquage initial

— Etablir la matrice d'entrée E

— Etablir la matrice de sortie S.

Exercice 4

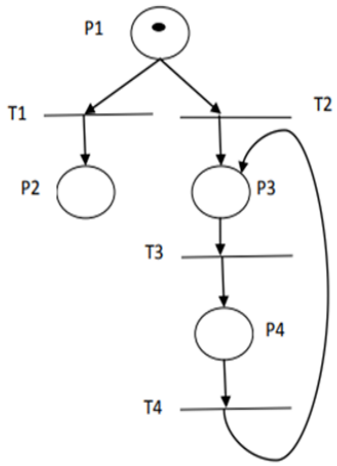
Donnez le graphe de marquage du RdP suivant :



Exercice 5

Construire le graphe de marquage correspondant au RDP suivant avec le marquage

initial M0= (1, 0, 0, 0).



Le Graphe est-il

— Borné ?

— Quasi vivant ?

— Vivant ?

— Sans blocage ?

Série d'exercices 2

Exercice 1

Soit une lampe de chevet (le système est la lampe), les deux interactions liées aux fonctionnalités du système sont l'allumage de la lampe et son extinction.

Un utilisateur peut allumer ou éteindre la lampe, ces deux opérations constituent bien les objectifs fonctionnels de cet utilisateur.

— Représentez les deux cas d'utilisations ?

— Faire intervenir le réseau électrique dans le cas d'utilisation de l'allumage de la lampe ?

Exercice 2

Il s'agit de décrire le cas d'utilisation de l'authentification d'un utilisateur à un système informatique. Cette authentification s'effectue de façon simple par la saisie d'un nom et d'un mot de passe.

— Représentez le cas d'utilisation de « Authentifier » permettent à un utilisateur de se connecter au système, sans spécifier les détails ;

— Introduire, dans le diagramme du cas d'utilisation, la saisie du nom et celle du mot de passe ainsi que la vérification de ces données ;

— Ajouter la saisie d'un code complémentaire après celle du mot

de passe. Ce code complémentaire est optionnel et n'est destiné qu'aux utilisateurs ayant besoin d'une sécurité accrue.

Exercice 3

Soit un document composé d'une ou plusieurs feuilles. La feuille comporte des objets géométriques et des texte.

Les objets graphiques supportent des opérations de type : sélectionner, copier, couper, coller et déplacer.

On suppose les deux objets géométriques suivants : cercle et rectangle.

— Modélisez cette situation par un diagramme de classe (utiliser la notion d'héritage, composition et agrégation).

Exercice 4

Soit un système d'information qui concerne le suivi des personnels d'un ensemble d'agences locales. Chaque agence se trouve dans une région, chaque région est pilotée par une direction régionale.

La direction régionale se charge d'un ensemble d'agences locales.

Une direction régionale est caractérisée par un code et un libellé.

— Modélisez ce système d'information par un diagramme de classe ?

Série d'exercices 3

Exercice 1

Soit en école qui assure des fonctions suivantes :

- Élaborer les catalogues qui décrivent les cours et donnent les dates prévisionnelles des sessions.

- Inscrire les personnes qui désirent participer aux sessions et leur envoyer leur convocation.

- Déterminer les formateurs qui vont animer les sessions.

Certaines sessions peuvent être animées par des vacataires. Certains cours sont organisés en filières, les cours utilisent des documents référencés.

Liste des attributs	
Code cours	N° catalogue
Date catalogue	N° document
Date session	N° session
Durée cours	Nom
État de la session (prévue, annulée, en cours, close)	Organisme extérieur
Intitulé du cours	Prénom
Lieu session	Service
Matricule	Titre document

— Donnez le diagramme de classe correspondant ?

Exercice 2

Chaque hôtel contient minimum 2 chambres qui contiennent chacune une salle d'eau (douche, baignoire).

Un hôtel peut héberger un ou plusieurs personnes puis il peut employer des personnels.

Un hôtel est dirigé par un seul directeur.

Chaque employé est caractérisé par un nom et un prénom.

Une personne peut être soit enfant ou adulte.

Chaque hôtel est caractéristique par une adresse, certain un nombre de chambres puis une catégorie.

Une chambre possède son numéro, un certain nombre de lits et le prix d'hébergement.

On veut savoir la date de réservation d'une chambre par des personnes.

Pour chaque jour de l'année, il faut calculer le prix de location de chaque chambre selon son prix et sa durée d'occupation.

La somme des prix de location nous permet de calculer le chiffre d'affaires de l'hôtel entre deux dates.

— Donnez une diagramme de classes correspondant ?

Exercice 3

Étude d'une caisse de supermarché : le déroulement normal d'utilisation d'une caisse de supermarché est le suivant :

- Un client arrive à la caisse avec ses articles à payer.

- Le caissier enregistre le numéro d'identification de chaque article, ainsi que la quantité

- La caisse affiche le prix de chaque article et son libellé.

- Lorsque tous les achats sont enregistrés, le caissier signale la fin de la vente.

- La caisse affiche le total des achats.

- Le caissier annonce au client le montant total à payer.

- Le client choisit son mode de paiement : on suppose que le client va payer en espèce.

- Le caissier encaisse l'argent, la caisse indique le montant à rendre au client.

- La caisse en registre la vente et l'imprime.

- Le caissier donne le ticket de caisse au client.

— Développez le diagramme de séquence correspondant ?

Exercice 4

Voici la recette pour faire une bonne mousse au chocolat : - Commencer par casser le chocolat en morceaux, puis le faire fondre.

- En parallèle, casser les œufs en séparant les blancs des jaunes.

- Quand le chocolat est fondu, ajouter les jaunes d'œuf.

- Battre les blancs en neige jusqu'à ce qu'ils soient bien fermes.

- Les incorporer délicatement à la préparation chocolat sans les briser.

- Verser dans des ramequins individuels.

- Mettre au frais au moins 3 heures au réfrigérateur avant de servir.

— Établir le diagramme d'activités pour modéliser la recette.

Série d'exercices 4

Exercice 1

Une carte géographique est caractérisée par une échelle, la longitude et la latitude de son coin inférieur gauche, la hauteur et la largeur de la zone couverte par la carte.

La carte comporte un ensemble de données géographiques de natures diverses.

Les villes et les montagnes sont repérées par un point unique. Chaque point a 2 coordonnées x et y calculées par rapport au coin inférieur gauche de la carte.

Un nom est associé à chaque donnée géographique repérée par un point.

Les routes et les rivières sont repérées par des lignes brisées, c'est à dire par un ensemble de points correspondant aux extrémités de ses segments de droite. Les routes et les rivières ont des noms et des épaisseurs de trait.

Les lacs, mers et forêts sont représentés par des régions caractérisées par un nom et une couleur de remplissage. Une région est une ligne brisée refermée sur elle même.

— Proposer le Diagramme de classes correspondant à ce système.

Exercice 2

Le fonctionnement d'une bibliothèque est le suivant : une bibliothèque propose à ses adhérant des œuvres littéraires. Les œuvres peuvent être présentes en plusieurs exemplaires.

Un adhérant peut emprunter jusqu'à'à trois livres.

— Écrire à l'aide d'un diagramme de séquence un scénario nominal d'emprunt.

Exercice 3

Une entreprise souhaite modéliser avec UML le processus de formation de ses employés afin d'informatiser certaines tâches.

Le processus de formation est initialisé quand le responsable de formation reçoit une demande de formation d'un employé. Cet employé peut éventuellement consulter le catalogue des formations offertes par les organismes agréés par l'entreprise.

Cette demande est instruite par le responsable qui transmet son accord ou son refus à l'employé. En cas d'accord, le responsable cherche la formation adéquate dans le catalogue des formations agréées qu'il tient à jour. Il informe l'employé du contenu de la formation et lui soumet la liste des prochaines sessions prévues.

Lorsque l'employé à fait son choix il inscrit l'employé à la session retenue auprès de l'organisme de formation concerné.

En cas d'empêchement, l'employé doit avertir au plus vite le responsable formation pour que celui ci demande l'annulation de l'inscription.

A la fin de la formation, l'employé transmet une appréciation sur le stage suivi et un document attestant sa présence.

Le responsable formation contrôle la facture envoyée par l'organisme de formation.

— Donner le diagramme de séquences associé à cette description.

Examen Type 1

Questions de cours

Répondez aux questions suivantes :

1. Quelle est la principale cause de l'apparition du génie logiciel ?

2. Donnez trois principaux facteurs influençant sur la qualité du logiciel.

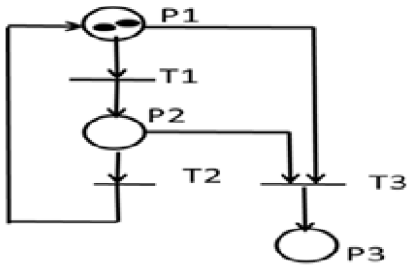
3. Comment peut-on juger la qualité du logiciel ?

4. Expliquez le principe d'anticipation du changement" en GL.

5. Que représente un RdP sans conflit ?

Exercice 1

Soit le RdP suivant :



1. Déterminez la matrice d'incidence de ce RdP.

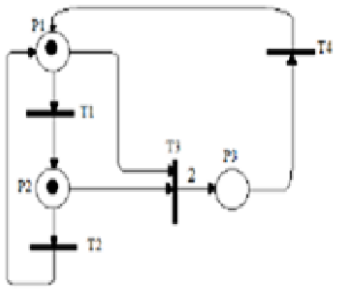
2. T1, T2, T3sont-elles vivantes ou quasi-vivantes ?

3. Le RdP est-il vivant ?

4. Le RdP est-il avec ou sans blocage ?

Exercice 2

Soit le RdP suivant :



1. Donner la matrice d'incidence de ce RdP.
2. Donner le graphe des marquages atteignables de ce RdP.
3. Trouver le marquage obtenu après la séquence de tire S = T1T2T3T4.

Examen Type 2

Questions de cours

Répondez aux questions suivantes :

1. Que représente le processus de développement d'un logiciel ? citez les étapes de ce processus.

2. Expliquez le modèle de développement du logiciel en Cascade.

3. Que représente un polymorphisme en GL ?

4. Quelle est l'utilité d'un diagramme de "use-case" ?

Exercice 1

Le déroulement normal d'utilisation d'un distributeur automatique de billets est le suivant :

le client (porteur de carte) introduit sa carte bancaire.

La machine vérifie alors la validité de la carte et demande le code au client si le code est correct, elle envoie une demande d'autorisation de prélèvement au groupement de banques. Ce dernier renvoie le solde autorisé à prélever.

Le distributeur propose alors plusieurs montants à prélever.

Le client saisit le montant à retirer, et après contrôle du montant par rapport au solde autorisé, le distributeur demande au client s'il désire un ticket.

Après la réponse du client, la carte est éjectée et récupérée par

le client, les billets sont alors délivrés (ainsi que le ticket), le client récupère enfin les billets et son ticket.

1. Modéliser cette situation à l'aide d'un diagramme de séquence en ne prenant en compte que le cas où tout se passe bien.

Exercice 2

Soit A et B, deux diagrammes d'états, l'un avec un état composite (B) et l'autre sans (A).

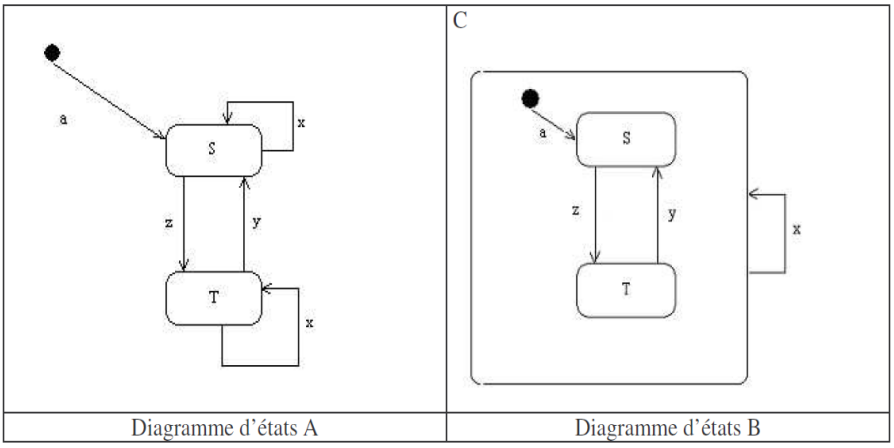


Diagramme d'états A | Diagramme d'états B

— Montrer que les deux diagrammes ci-dessus ne sont pas équivalents en trouvant une séquence d'événements qui n'amène pas dans le même état en A et en B (on supposera que les séquences débutent à l'état initial).

Examen Type 3

Questions de cours

1- Citer les quatre principaux facteurs de qualité d'un logiciel. Proposer deux autres facteurs et dire dans quelles circonstances, ces deux derniers deviennent-ils plus importants que les autres ?

2- Expliquer pourquoi le modèle de cascade ne reflète pas exactement les activités de développement du logiciel.

3- Etablir un tableau montrant pour chacune des étapes du modèle en V du cycle de vie des logiciels, les documents à délivrer en sortie.

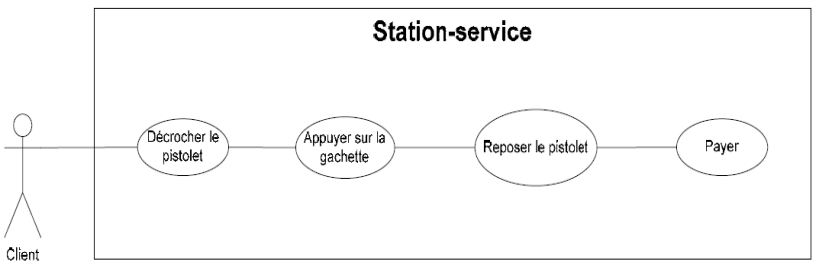
4- Quelles sont les qualités requises dans un cahier de charges ?

5- Montrer, pour chaque modèle de cycle de vie, les domaines d'application les plus appropriés.

6- Comment peut-on réduire l'écart entre les besoins réels d'un client et les besoins exprimés dans le cahier de charges ?

Exercice 1

Quel est le défaut du diagramme suivant :



Exercice 2

Soit un robot qui dispose d'un bras articulé se terminant par une pince. Le fonctionnement du robot est le suivant :

Le robot déplie son bras, attrape une pièce, replie son bras puis relâche la pièce.

— Représentez à l'aide d'un diagramme de séquences, l'échange des messages entre les objets «Robot», «Bras Articulé» et «pince» ?

Bibliographie

[1] Mylène Bapst, Daria Druzhinenko-Silhan et Patrick Schmoll. “Penser l’effraction en approche systémique”. In : *Cahiers de systémique* 6 (2025), p. 5-12.

[2] Zineb BENOTMANE. *Génie Logiciel*. Université Relizene, Polycopié pédagogique, 2023.

[3] Jean Bézivin et Xavier Blanc. “MDA : Vers un important changement de paradigme en génie logiciel”. In : *Développeur référence v2* 16 (2002), p. 15.

[4] Tshikutu Anaclet Bikengela. “COURS DE MODELISATION OBJET : UML”. In : (2024).

[5] David Delfieu, Olivier Henri Roux et Robert Valette. “Réseaux de Petri : Théorie et mise en oeuvre”. In : *Techniques de l’ingénieur Automatique et ingénierie système* S7254 (2025).

[6] Usman Khan Durrani et al. “A decade of progress : A systematic literature review on the integration of AI in software engineering phases and activities (2013-2023)”. In : *IEEE Access* 12 (2024), p. 171185-171204.

[7] DI GALLO Frédéric. “COURS DE GENIE LOGICIEL”. In : *Edition Cnam Bordeaux* 2000 (1999).

[8] Christopher Fuhrman et Yvan Ross. *Analyse et conception de logiciels*. École de technologie supérieure. fabrique REL, 2023.

[9] Issaka Gmakouba. “Gestion de la qualité des projets en génie logiciel : facteurs clés et solutions”. Thèse de doct. Université du Québec à Trois-Rivières, 2024.

[10] Reinhard Haberfellner et al. *Systems engineering*. Springer, 2019.

[11] Hervé P Hillion. “Modélisation et analyse des systèmes de production discrets par les réseaux de Petri temporisés”. Thèse de doct. Paris 6, 1989.

[12] Rucien KINDUKULU, Yanniv Ekoto NDJASAKI et Fiston Bulambo LOKOLE. "Reflexions croisees entre lapproche qualitative en recherche et les methodes agiles de developpement logiciel." In : *International Journal of Engineering Development and Research* 14.1 (2026).

[13] Phillip A LAPLANTE et Mohamad KASSAB. *Requirements engineering for software and systems*. Auerbach Publications, 2022.

[14] Emmanuel LAZARD. *Architecture de l'ordinateur*. Pearson Education France, 2006.

[15] Delphine LONGUET. "Introduction au génie logiciel et à la modélisation". In : (2018).

[16] Rajib MALL. *Fundamentals of software engineering*. PHI Learning Pvt. Ltd., 2018.

[17] Yann MORERE. *Cours de réseau de Petri*. 2002.

[18] Jacques PRINTZ. *Le génie logiciel*. FeniXX, 1995.

[19] Damiano TORRE et al. "How consistency is handled in model-driven software engineering and UML : an expert opinion survey". In : *Software Quality Journal* 31.1 (2023), p. 1-54.

[20] Raphael Grevisse YENDE. "Support De Cours De Génie Logiciel". Thèse de doct. INSTITUT SUPÉRIEUR DE COMMERCE, 2019.