Chapitre V

Méthodes d'analyse et de conception

1. INTRODUCTION

1.1. Pourquoi une méthode d'analyse ?

La résolution de n'importe quel problème nécessite une, voire plusieurs phases de réflexion plus ou moins longues en fonction de la complexité et du type du problème. Lors d'un projet informatique, cette réflexion doit pouvoir être comprise et reprise par toutes les personnes travaillant sur le projet.

C'est pour cette raison que des méthodes d'analyse et de conception ont été définies. Certaines disparaissent laissant la place à d'autres méthodes plus adaptées, d'autres évoluent dans le temps en fonction des différentes technologies. Chaque méthode a ses qualités et ses défauts. Il est donc parfois utile et même nécessaire en fonction de l'étape d'analyse du projet d'appliquer des méthodes différentes.

1.2 Enjeux et risques de la conception

Un système d'information (SI) est un ensemble organisé de ressources (matériels, logiciels, personnel, données et procédures) qui permet de regrouper, de classifier, de traiter et de diffuser de l'information sur un phénomène donné.

Le système d'information analysé doit être :

- techniquement exploitable
- pertinent dans le contexte de l'entreprise
- ergonomiquement adapté

Les causes d'échec sont multiples

- mauvaise analyse des besoins
- performance insuffisante
- implication insuffisante des utilisateurs

Nécessité d'une méthodologie rigoureuse de conception et de mise en œuvre

2. Méthodes de conception :

La modélisation du logiciel est l'activité la plus scientifique de la rédaction du cahier des charges. On applique ici, de façon intensive, le principe de séparation des problèmes en sous-problèmes et ses instanciations que sont l'abstraction et la modularité.

Il existe de nombreuses méthodes de conception d'un Système d'Information

2.1 Approches structurées

- SADT
- SART
- Automate à état finie
- DFD
- Diagrammes états-transitions
- Réseau de pétrie
- MERISE (Pour la manipulation des bases de données)

2.2 Approches orientée objets

L'approche orientée objet modélise un système sous la forme d'un ensemble d'objets possédant un état **propre et interagissant par le biais d'envoi de messages.** C'est l'approche la plus courante dans l'industrie du logiciel contemporaine.

Exemple d'approches orientés objet :

- OMT
- OOSE
- Booch
- UML

2.3 Méthodes agiles

Les méthodes agiles se définissent comme des méthodes de développement de logiciel qui permettent de réduire les risques d'échec, produire le résultat attendue et de livrer des systèmes logiciels de qualité dans les meilleurs délais. Pour ceci, ces méthodes en plus de la conception définissent la méthode et les outils de travail ainsi que la relation entre les différentes personnes intervenant dans le projet.

Exemple de méthodes agiles

- RAD
- RUP
- XP
- Scrum

2.1 Approches structurées

SADT (Structured Analysis and Design Technique)

La méthode SADT est une méthode d'analyse hiérarchique et descendante

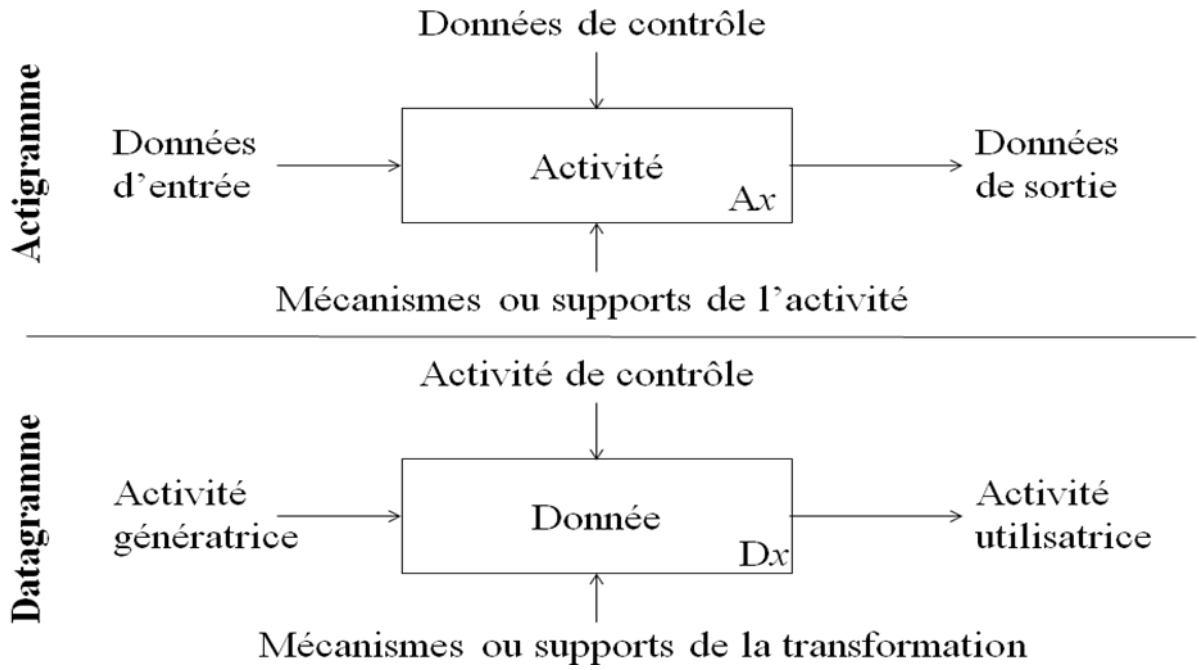
La méthode SADT utilise 2 types de diagrammes

- **Actigrammes** décrivant les séquences d'activités
- **Datagrammes** décrivant les transformations de données

Ces diagrammes utilisent des boîtes similaires composées

- d'entrées
- de sorties
- de contrôles
- de mécanismes

Formalisme graphique



Démarche

1. On commence par le diagramme de plus haut niveau A-0 (*A moins zéro*) représentant la finalité du système.
2. Ensuite, on descend dans les niveaux en traçant le diagramme de niveau A0 (*A zéro*) puis A1 et ainsi-de-suite en respectant la hiérarchie des niveaux. On décrit de cette manière les sous-fonctions du système ce qui permet d'en raffiner la perception et la structure.

SART (Structured Analysis for Real Time)

SART est une méthode permettant de modéliser un système temps réel complexe. Il s'agit d'une adaptation de SADT pour le temps réel.

La méthode SART intègre trois aspects fondamentaux d'une méthode de spécification :

- L'aspect fonctionnel (ou transformation de données) :
  - représente la transformation que le système opère sur les données et
  - spécifie les processus qui font ces transformations.
- L'aspect évènementiel (piloté par les évènements) :
  - représente les évènements qui conditionnent l'évolution du système et
  - spécifie la logique de contrôle qui produit des actions et des évènements en fonction d'événement d'entrée et fait changer l'état du système
- L'aspect informationnel (données) : spécification des données sur les flots ou dans les stockages.

Système informatiques temps réel :

Les systèmes informatiques temps réel sont des systèmes qui prennent en compte les contraintes temporelles. C'est-à-dire que le système doit délivrer des résultats exacts et dans des délais imposés.

Un système temps réel n'est pas un système « qui va vite / rapide» mais un système qui satisfait des contraintes temporelles (les contraintes de temps dépendent de l'application et de l'environnement alors que la rapidité dépend de la technologie utilisée, celle du processeur par exemple).

Automate à état finie

C’est une technique simple utilisé pour décrire des systèmes synchrones à état finit. Il consiste en :

- un ensemble fini d’états
- un ensemble fini d’entrées
- une fonction de transition. Une certaine entrée dans un certain état le fait passer à un autre état.

Graphiquement une machine à états finis est représentée par un graphe (appelé diagramme d’états)

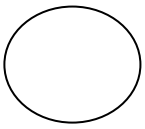
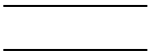
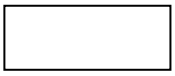
Les limitations de ce modèle sont évidentes. Dans les systèmes complexes le nombre des états peuvent être énorme et la modélisation complète est donc irréalisable. La seule possibilité est d’abstraire de nombreux détails. Par ailleurs, il s’agit d’un modèle synchrone ; c’est à dire qu’à tout instant un état global unique doit être défini et une seule transition peut survenir. Il est donc très mal adapté à la description de systèmes asynchrones, où plusieurs composants évoluent en parallèle de manière assez autonome.

Etat 

Transition 

Diagramme de flux de données (DFD)

Un DFD est un diagramme qui représente les spécifications d'un système en tant que processus (fonction). Les DFD montrent comment chaque processus transforme ses entrées en sorties (flot entrant, flot sortant). Les données peuvent être persistantes (dans des stockages) ou circulantes (flots de données).

- La représentation graphique classique distingue :
  - les fonctions par des cercles 
  - les stockages par des boîtes ouvertes 
  - les flots par des flèches 
  - les entités externes par des rectangles 

La capacité du diagramme de flux de données à illustrer des niveaux supérieurs ou inférieurs d'un système est une autre caractéristique très utile. Sur un DFD, il est possible de décomposer les processus de niveau supérieur en DFD détaillés et distincts de niveau inférieur, on peut ainsi décomposer les processus de DFD détaillés en diagrammes supplémentaires pour obtenir de multiples niveaux d'abstraction.

Réseau de pétries (RDP)

1. Introduction

Cette technique est particulièrement bien adaptée pour décrire le comportement des systèmes asynchrones avec des évolutions parallèles (Spécifications dynamiques des systèmes).
Graphiquement un RdP est constitué :

- d'un ensemble fini de places (des cercles), 
- d'un ensemble fini de transitions (des barres),
- d'un ensemble fini de flèches, connectant soit des places à des transitions, soit des transitions à des places mais jamais une place à une place ou une transition à une transition. 

- Une place correspond à une variable d'état du système modélisé.
- Une transition correspond à une action qui va entraîner l'évolution des variables d'état du système. Les transitions modélisent en général des actions effectuées par le système.
- La présence d'un jeton dans une place signifie que la condition nécessaire à la réalisation d'une action est satisfaite.
- La transition a lieu si toutes les places en entrée ont un jeton
- Une fois que la transition est franchie les places en entrée perdent leur jeton et les places en sortie gagnent des jetons.

2. Représentation graphique

2.1 Places, transitions et arcs (figure.1)

Un réseau de Petri est:

- un graphe,
- formé de deux types de nœuds appelés places et transitions reliés par des arcs orientés,
- les arcs sont bipartis, c'est-à-dire qu'un arc relie alternativement une place à une transition et une transition à une place.

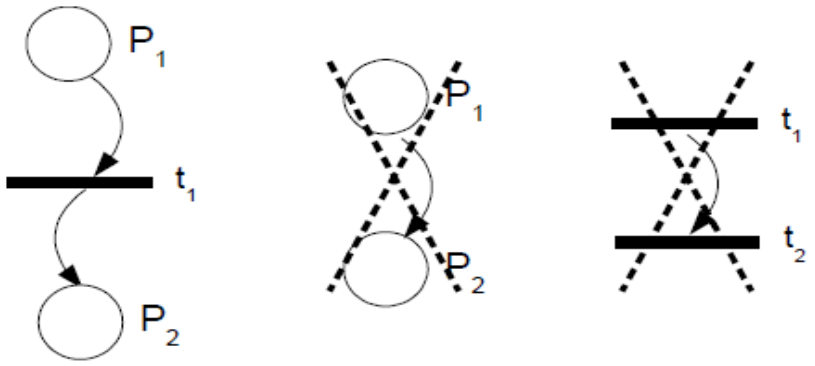


Figure.1 Représentation graphique d'un réseau de Petrie.

2.2 Marquages (figure.2)

- Chaque place d'un réseau de Petri peut contenir une ou plusieurs marques (on parle aussi de jetons).

- La configuration complète du réseau, avec toutes les marques positionnées, forme le marquage et définit l'état du réseau (et donc l'état du système modélisé).

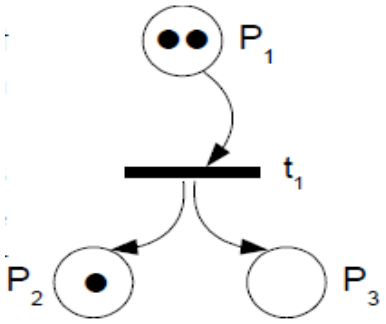


Figure.2 Marquage d'un réseau de Petrie.

2.3 Franchissement de transitions (figure 3)

- Les réseaux de Petri intègrent un formalisme permettant de passer d'un marquage à un autre : c'est le franchissement des transitions.
- Une transition est franchissable si chacune des places en entrée comporte au moins un jeton.
- Pour les transitions franchissables, on définit le franchissement effectif selon les règles suivantes :
  - le franchissement est une opération indivisible (atomique),
  - un jeton est consommé dans chaque place en entrée,
  - un jeton est produit dans chaque place en sortie

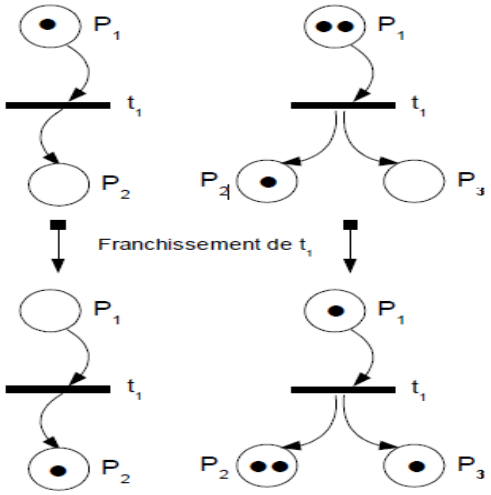


Figure.3 Franchissement de transitions.

2.4 Graphe de marquage (figure. 4)

- On utilise le graphe de marquages quand le nombre de marquages accessibles est fini.

- On peut aussi représenter un réseau de Petrie à l'aide d'un vecteur M. Chaque place contient un nombre entier (positif ou nul) de marques ou jetons. Le nombre de marque contenu dans une place Pi sera noté soit M(Pi) soit mi. Le marquage du réseau à l'instant i, Mi est défini par le vecteur de ces marquages mi c'est à dire Mi = (m1, m2, …,mp où p est le nombre de place). Le marquage initial décrit l'état initial du système modélisé (M0).

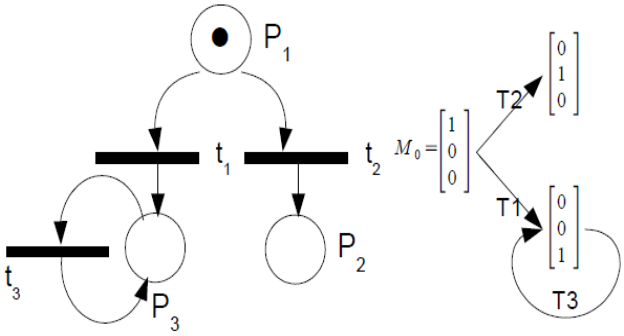


Figure.4 Graphe et vecteur de marquage.

2.2 Approches orientée objets

L'approche orientée objet modélise un système sous la forme d'un ensemble d'objets possédant un état **propre et interagissant par le biais d'envoi de messages.** C'est l'approche la plus courante dans l'industrie du logiciel contemporaine.

Exemple d'approches orientés objet :

- OMT
- OOSE
- Booch
- UML

2.3 Méthodes agiles

Les méthodes agiles se définissent comme des méthodes de développement de logiciel qui permettent de réduire les risques d'échec, produire le résultat attendu et de livrer des systèmes logiciels de qualité dans les meilleurs délais. Pour ceci, ces méthodes en plus de la conception définissent la méthode et les outils de travail ainsi que la relation entre les différentes personnes intervenant dans le projet.

Principes des méthodes agiles

1. Priorité aux personnes et à la communication plutôt qu'aux procédures et aux outils de travail :
   - travail en groupe,
   - communication entre les membres du groupe.

2. Priorité à la collaboration avec le client par rapport à la négociation des contrats :
   - retour en arrière régulier avec le client,
   - développer des solutions réellement adaptées aux besoins des clients,
   - établir des relations de confiance avec le client.

3. Priorité à l'acceptation du changement par rapport à la planification :
   - planification flexible,
   - accepter les modifications des versions déjà validées.

- Exemple de méthodes agiles
  - RAD
  - RUP
  - XP
  - Scrum

Règles à suivre pour le développement d'une application

Le développement d'une application exige une méthodologie et des caractéristiques:

1. procéder par étapes
    - •prendre connaissance des besoins
    - •effectuer l'analyse
    - •trouver une solution informatique
    - •réaliser
    - •tester
    - •installer
    - •assurer le suivi
2. Procéder avec méthode
    - •du général au détail et au technique
    - •fournir une documentation
    - •utiliser des méthodes appropriées
3. Savoir se remettre en question
    - •bonne construction ?
    - •bon produit ?
4. Choisir une bonne équipe
    - •trouver les compétences
    - •définir les missions de chacun
    - •coordonner les actions
5. Contrôler les coûts et délais
    - •aspect économique
    - •bonne maîtrise de la conduite du projet
    - •investissements au bon moment
6. Garantir le succès du logiciel
    - •répondre à la demande
    - •assurer la qualité du logiciel
7. Envisager l'évolution
    - •du logiciel
    - •du matériel
    - •de l'équipe