Chapitre 13

L'ingénierie des besoins

Se lancer à corps perdu dans l'implémentation d'un problème peut coûter très cher comme on s'en est aperçu à la fin des années soixante.

Un logiciel a pour enjeu de satisfaire des utilisateurs, et notre quotidien en est fortement imprégné ; c'est le cas des domaines des banques, des transports, des communications, de la santé, etc. Le fonctionnement d'une entreprise dépend fortement du fonctionnement – correct ou non – des logiciels. Le logiciel est un produit conçu et fabriqué pour répondre à des besoins préalablement définis.

La réalisation d'un logiciel est un processus long, complexe et coûteux. La particularité de ce produit est d'être un objet immatériel, malléable, qui ne s'use pas, Par contre, il peut devenir obsolète par rapport au contexte technique, aux concurrents, etc. Enfin, les défaillances proviennent d'erreurs humaines (peu prévisibles).

Dans ce chapitre, nous allons d'abord présenter le contexte des logiciels professionnels, ensuite le processus de développement d'un tel logiciel et enfin approfondir l'aspect de l'ingénierie des besoins avec le cas d'APB.

13.1 Contexte

L'*Ingénierie des Systèmes* vise à transformer les besoins des utilisateurs en spécifications formalisées d'une future application. Elle est concernée par des activités visant à produire un logiciel qui doit répondre à ces besoins.

Donnons d'abord quelques définitions sur la terminologie utilisée, ensuite nous présenterons quelques caractéristiques du logiciel et enfin nous terminerons cette section par les raisons de succès ou d'échec de la mise en œuvre des logiciels.

13.1.1 Les concepts de base (ou la terminologie)

Pour, par exemple, le « système APB », on doit distinguer le *système d'information* du *système informatique*. Le premier est « *la partie du réel constituée d'informations organisées, d'événements ayant un effet sur ces informations, et d'acteurs qui agissent sur ces informations ou à partir de ces informations, selon des processus visant une finalité et utilisant les technologies de l'information* » [MHL-08].

Remarquons que le système d'information (SI) se situe à un niveau plus global que le logiciel or souvent lorsqu'on utilise l'acronyme SI, c'est pour le système informatique. L'informatique n'est qu'un élément parmi d'autres éléments constituant le système d'information. Aussi, nous retiendrons la définition suivante : « *un* **système d'information** *est un ensemble de ressources (humaines, organisationnelles, matérielles et logicielles) permettant de gérer (saisir, stocker, traiter, restituer) les informations utiles aux décideurs et aux opérationnels d'une organisation* » [Rei-00].

Le **système informatique** est, de ce point de vue, un sous-ensemble du système d'information, constitué d'un ensemble de composants matériels, logiciels et humains qui coopèrent de façon organisée dans le but d'atteindre un objectif commun. Dans le cas d'APB, le but est de satisfaire les souhaits des lycéens de poursuivre telle ou telles études.

Le **logiciel** représente une partie du système informatique, il peut être défini comme « *un ensemble de programmes nécessaires au fonctionnement d'un système informatique* ». C'est un produit composé de codes sources et d'exécutables, de programmes tests, de fichiers de configuration, de documents internes et externes à destination de l'équipe de développement et des utilisateurs. Seul le code exécutable produit le fonctionnement attendu par les utilisateurs.

À travers ces trois définitions, nous constatons que système, logiciel et programme, bien qu'étroitement liés, représentent des éléments bien distincts.

Nous allons maintenant présenter les caractéristiques d'un logiciel.

13.1.2 Le logiciel et ses caractéristiques

Un logiciel est un produit dont la particularité est d'être fabriqué en un seul exemplaire puis utilisé par simple recopie. Il existe une grande variété de logiciels [Lon-15]. On distingue les *logiciels spécifiques* des *progiciels*. Les premiers sont développés dans un contexte particulier alors que les seconds sont vendus en grand nombre et adaptés au contexte lorsque nécessaire. Le logiciel APB ne rentre dans aucun des deux cas : c'est un logiciel spécifique qui a été développé sur la base d'un logiciel spécifique, comme décrit dans la section 6.5.

Par ailleurs, il y a trois classes de logiciels : (1) les logiciels « autonomes » comme les traitements de texte et les tableurs, (2) les logiciels qui gèrent des processus de gestion ou des processus industriels et (3) les logiciels embarqués comme dans les transports ou les télécommunications. On peut dire que le logiciel APB rentre dans le cadre d'un processus de gestion classique.

On parle souvent d'*applications Web* : ce sont des logiciels hébergés sur un serveur Web et manipulables *via* les réseaux grâce à un navigateur Web. C'est le cas de l'application APB.

La réalisation d'un logiciel est un processus long, complexe et coûteux. Parmi les causes multiples d'échec, certaines sont dues à la nature même du produit : le logiciel est un objet immatériel représenté par une structure d'information. Enfin, les défaillances proviennent d'erreurs humaines car il est facile à modifier avec des conséquences parfois graves. De plus les erreurs sont peu prévisibles au départ : elles sont détectées lors de son fonctionnement. Enfin, le logiciel peut devenir vite obsolète, d'où la nécessité de prévoir son évolutivité.

13.1.3 La crise du logiciel et les problèmes de l'ingénierie du logiciel

Un système n'est de de qualité que s'il répond aux objectifs pour lesquels il a été décidé de le mettre en place. De façon plus précise, on peut dire qu'un logiciel de qualité est celui qui répond aux besoins des utilisateurs, non seulement d'un point de vue fonctionnel, mais aussi du point de vue de son apprentissage et de son utilisation.

Or de nombreuses études ont montré l'insatisfaction des utilisateurs et les raisons sont diverses. Dès 1968, l'expression « crise » est évoquée lors d'une conférence consacrée à l'ingénierie des logiciels par MACILROY [McI-76]. Il y présentait un article intitulé « *Mass produced Software Components* » dans lequel il propose une solution à la crise du logiciel fondée sur la création en masse des composants logiciels. Son idée est que, face aux difficultés rencontrées à produire du logiciel, la solution est d'arriver à industrialiser le processus de sa fabrication.

De ce constat est apparue la nécessité de définir des méthodes conduisant à la création d'un système. Les premières méthodes sont nées au début des années 1970 [Ros-72].

Depuis l'informatisation des administrations, on n'a cessé de constater que les logiciels développés ne répondent pas aux besoins des usagers. Souvent les fonctionnalités incluses dans le logiciel sont inadaptées. Des chercheurs comme BELL et TAYER [BT-76] ont constaté que l'ambiguïté des documents définissant les besoins ont une influence sur la qualité du logiciel.

Diverses études rendent compte des raisons des échecs et des réussites des projets. Parmi ces études, les rapports du *Standish Group*, comme [Sta-01], sont parmi les plus connus, souvent cités et régulièrement mis à jour. Les résultats de leurs études, fondées sur des enquêtes lancées à grande échelle ont permis de constater que :

— 20% des projets se terminent dans le temps imparti et dans les délais avec un système livré conforme au cahier des charges ;
— 50% des projets se terminent soit dans le temps, soit dans les délais avec un système livré légèrement différent de celui défini dans le cahier des charges ;
— 30% des projets sont abandonnés pour diverses raisons.

Le tableau suivant rend compte des résultats des enquêtes menées entre 1995 et 2013.

Année	Succès	Mitigé	échec
1995	16%	53%	31%
2000	28%	49%	23%
2006	35%	46%	19%
2010	37%	42%	21%
2013	39%	43%	19%

Une analyse des facteurs de succès et d'échec est disponible dans [BB-14]. Parmi les facteurs de succès, celles inhérentes aux exigences représentent 37,1%. Ces études citent principalement : l'implication des utilisateurs (15,9%), le support du management exécutif (13,9%), une définition claire des exigences (13%).

Parmi les facteurs d'échec, celles inhérentes aux exigences représentent 44%, parmi lesquelles on peut citer des exigences incomplètes (13,1%), un manque d'implication des utilisateurs (12,4%).

Une estimation des coûts selon l'origine a été effectuée et, selon Barry BOEHM [Boe-81], la correction des erreurs coûte 200 fois plus chère en phase de tests qu'en phase d'analyse des besoins. Il est admis que « *la partie la plus difficile de la construction d'un logiciel consiste à définir précisément ce qu'il faut construire* » [Bro-87].

Ces différentes études et les chiffres établis montrent clairement que le succès ou l'échec d'un développement se joue lors des phases en amont du développement logiciel. La définition complète des exigences, l'implication des parties prenantes sont des éléments essentiels à la réussite d'un projet informatique.

13.2 Les activités de développement d'un logiciel

Le développement d'un logiciel est une activité intellectuelle nécessitant la participation de plusieurs types d'acteurs, à savoir le client (le maître de l'ouvrage), les développeurs du produit (le maître d'œuvre) et les utilisateurs du produit. Tous ces acteurs, considérés comme les parties prenantes (*stakeholders* en anglais), participent à la production d'un logiciel initiée dans le cadre d'un projet informatique.

Le cycle de vie d'un système suit une chronologie d'étapes : l'*analyse des besoins*, la *conception*, la *programmation*, les *tests*, le *déploiement* et la *maintenance*. Chaque étape a ses propres règles et conduit à élaborer des résultats précis (documents, diagrammes, programmes, etc.). Chaque étape comporte un ensemble de phases, d'activités, de tâches et, enfin, chaque étape utilise des concepts et des règles appropriées (formalisme).

Passons en revue chacune de ces étapes.

13.2.1 Étape d'« analyse des besoins »

Cette étape concerne les activités de *recueil, d'analyse et de spécification des besoins.*

Au démarrage d'un projet informatique, le *client*, c'est-à-dire le demandeur du système, et les utilisateurs finaux ont une idée, souvent peu claire de ce qu'ils veulent. La phase de *recueil des besoins* consiste à étudier le contexte métier (étude du processus en cours, règles, lois et standards), l'environnement existant et les contraintes d'utilisation.

La *spécification des besoins* consiste à représenter l'application dans son environnement, les fonctionnalités qu'elle doit rendre (les *besoins fonctionnels*) et sous quelles contraintes elle doit les rendre (*besoins non fonctionnels*). Une première architecture fonctionnelle se dessine à ce niveau.

13.2.2 Étape de « conception »

Elle consiste à construire l'architecture du futur système et à décrire ses composants ; il s'agit à ce stade de construire ce que l'on appelle la *solution informatique.*

Deux aspects doivent être définis : l'architecture de l'application et la description détaillée de la conception. La première consiste à définir les composants de l'application, les relations entre les composants et avec l'environnement du système.

La conception détaillée de l'application doit mettre en évidence la réalisation des fonctionnalités par les composants, la structuration des données et la description des *Interfaces Homme-Machine* (IHM). Il s'agit de définir l'architecture technique et l'infrastructure qui va supporter l'architecture fonctionnelle définie dans la phase précédente.

13.2.3 Étape de « programmation et tests »

Cette étape consiste à coder dans un langage informatique les composants du système, à les tester avant de les mettre à disposition des utilisateurs. Les tests sont de différentes natures : il y a les *tests unitaires* relatifs au test des composants indépendamment les uns des autres, les *tests d'intégration* relatifs aux interactions entre les composants, les *tests de validation* par rapport au cahier des charges et enfin les *tests d'acceptation* relatifs à la validation faite par le client du système.

13.2.4 Étape de déploiement

Le déploiement consiste en la mise en place du système développé. Pour cela, diverses activités comme l'installation des sites, la configuration, la préparation de la mise en production ainsi que la formation des utilisateurs doivent être réalisées. Les utilisateurs jouent un rôle important dans cette étape. Il est d'usage de mettre en place un suivi pour remonter les problèmes d'utilisation du nouveau système.

13.2.5 Étape de maintenance

Cette étape concerne le système pendant son fonctionnement. En effet, de nouvelles règles de gestion, un changement dans les critères ou tout simplement une erreur technique non décelée lors des tests nécessitent de faire des modifications. Il est admis que la maintenance des applications requiert de 50 à 70% de l'effort total.

On ne peut terminer ce paragraphe sans faire référence aux *modèles de développement.* Ils représentent les différentes manières d'organiser les étapes précédemment décrites. Ces modèles ont connu des évolutions au fur et à mesure de la pratique. Ils sont classifiés selon trois types de modèles : le *modèle en cascade* (*waterfall* en anglais) [], le *modèle en V* [], le *modèle en spirale* [?]. Actuellement, ce sont les méthodes dites agiles qui ont le vent en poupe. Ce sont plus des principes qui ont été introduits dans le cycle de vie classique et qui préconisent une conception–réalisation itérative et incrémentale [].

13.3 L'ingénierie des besoins

L'ingénierie des besoins est une activité qui se situe en amont du cycle de vie de production d'un logiciel (d'un système). Elle permet de définir les buts, les fonctions et les contraintes réelles des logiciels.

Pourquoi cette étape est-elle très importante ? Parce qu'on veut éviter de construire des systèmes techniquement corrects mais inadaptés aux besoins réels. Il faut se donner les moyens de définir à quoi va servir le système bien en amont de sa programmation.

13.3.1 Besoins et exigences

Les termes « besoin » et « exigence » sont deux termes utilisés pour désigner quelque chose de souhaité ou d'attendu par les utilisateurs. Dans le dictionnaire *Le Robert*, le terme « besoin » est défini comme une « *exigence née de la nature ou de la vie sociale* » alors que le terme « exigence » est défini comme « *ce qui est nécessaire, indispensable ou une contrainte imposée par un métier ou une discipline* ». Ce n'est pas très clair car, dans la première, le terme « exigence » est utilisé pour définir le terme « besoin ».

Dans la communauté, on différencie la notion de « besoin » de la notion d'« exigence ». La norme AFNOR X50-150 [?] définit le besoin comme « *une nécessité ou un désir éprouvé par un utilisateur* ». Cette norme a évolué en 1991 en AFNOR X50-151 dans laquelle a été formalisé le contenu d'un cahier des charges fonctionnel []. Le principe de cette norme est de distinguer entre le besoin exprimé (ou le service attendu) de la solution technique (architecture logicielle) ; elle s'applique à tous les projets industriels et plus spécifiquement à l'informatique et aux systèmes d'information. Par exemple, dans le cas d'APB, l'un des besoins émis est que « le 2 avril est la date limite de confirmation des vœux », l'exigence serait « aucun ajout (ou modification) de vœux ne peut être effectué à partir du 2 avril ».

On retient la définition de l'exigence donnée par [BB-14] comme « *un énoncé qui traduit un besoin et/ou des contraintes (techniques, de coût, de délais, etc)* ». Cet énoncé est rédigé dans un langage naturel, semi-formel ou formel.

Nous utiliserons le terme « exigence » dans la suite, car on comprend que les besoins retenus lors de la phase d'analyse des besoins deviennent les exigences à mettre en œuvre dans le système.

13.3.2 Les catégories d'exigences

On distingue principalement deux catégories d'exigences : les *exigences sur le produit* et les *exigences sur le processus*. Les premières décrivent les fonctionnalités que le système doit réaliser. Les secondes concernent le processus à suivre pour obtenir le produit (le système).

Les exigences sur le produit se décomposent en *exigences fonctionnelles* et *exigences non fonctionnelles*. Les premières décrivent ce que le système fait alors que les secondes décrivent les propriétés (sécurité, performance, accessibilité, etc.) que le système doit avoir pour produire un service de qualité. On peut imaginer aisément les exigences fonctionnelles : « Saisir les vœux des lycéens » ou « Contrôler le nombre de vœux émis » ou « Classer les vœux » ou « Confirmer le classement par le lycéen » sont des fonctionnalités du système. On peut citer comme exigence non fonctionnelle « la disponibilité du système 24h durant » ou « le temps de réponse pour accéder à un dossier ne doit pas dépasser 5 s ».

13.3.3 Processus d'ingénierie des exigences

Nous avons vu que l'étape d'analyse des besoins concerne les activités de recueil, d'analyse et de spécification des besoins.

1. La *phase de recueil des besoins* consiste à identifier et à comprendre le problème à résoudre ; il s'agit notamment de répondre aux questions suivantes : à qui est destinée l'application ? Pourquoi ? Quel problème (ou service) doit-elle résoudre ? Quand et comment doit-elle fonctionner ?

   Tous ces éléments sont spécifiés dans un *cahier des charges client*. Ils sont exprimés de façon informelle en langage naturel ou semi-formel, sous la forme de hiérarchies de buts, ou de scénarios (*user stories* en anglais). Ce document n'indique pas comment réaliser les besoins mais répond aux questions du quoi et du pourquoi.

2. La phase « analyse et spécifications des besoins » consiste à analyser les besoins de façon plus approfondie et à produire un document de spécification des exigences. L'analyse des besoins consiste à structurer, à raffiner et à valider les besoins. La structuration a pour objet de modéliser les processus métier (qui fait quoi ? quand ? et où ?), le domaine (les concepts métier) et le modèle de l'application. La validation par les gens du métier doit permettre de contrôler que le modèle obtenu est conforme aux besoins attendus. Une activité concerne la négociation qui doit conduire aux choix des besoins à implanter. À cela s'ajoute l'activité d'évolution permettant de corriger les erreurs et/ou de prendre en

compte de de nouveaux besoins. Un exemple concernant APB en est la prise en compte des bacs français passés à l'étranger.

Le document de spécifications des besoins constitue le contrat entre maîtrise d'ouvrage et maîtrise d'œuvre. La norme IEEE Std 830 [?] propose un plan de ce document.

La spécification des besoins peut être informelle, semi-formelle ou formelle.

— Les besoins spécifiés de façon informelle sont exprimés en langue naturelle. Bien que lisibles par les utilisateurs, ils peuvent présenter l'inconvénient d'être interprétés de façon différente par différentes personnes, pour cause d'ambiguïté ou d'incomplétude.

— Les besoins exprimés dans un langage semi-formel peuvent être de trois formes différentes : buts et scénarios, pseudo-code (sous forme algorithmique) ou diagrammes UML [Roq-06]. Nous verrons dans le paragraphe suivant une application du langage UML au cas d'APB.

— Enfin, les besoins spécifiés en langage formel utilisent un formalisme mathématique (langage B, langage Z, réseaux de Petri,...). Les spécifications obtenues peuvent être analysées de façon automatique. Ces langages sont utilisés pour les logiciels critiques (aéronautique, santé, etc.), qui ont besoin d'être simulés avant d'entrer en fonctionnement.

En général, le document relatif aux recueils est écrit en langue naturelle alors que celui des spécifications est écrit en un langage semi-formel.

13.4 Les modèles de spécification : l'application APB

La modélisation tient un rôle important en développement de logiciel. Chaque étape décrite ci-dessus produit des modèles. Un *modèle* est une représentation abstraite du futur système, il permet de mieux comprendre le fonctionnement du système et il est un vecteur de communication entre la MOA et la MOE. Un modèle est une abstraction contenant un ensemble restreint d'informations représentant un système selon un point de vue [?]. Pour élaborer un modèle du système, on utilise un langage commun aux parties prenantes.

Depuis l'avènement des méthodes, on distingue la modélisation fonctionnelle de la décomposition « objet ». Dans l'approche fonctionnelle, c'est la fonction qui définit l'architecture du système. Cette architecture est obtenue par raffinements successifs de la fonction (le système) jusqu'à l'obtention de fonctions élémentaires à implémenter [Ros-72]. Dans la décomposition objet, l'analyse est fondée sur les objets du réel qui contribuent à la réalisation d'un service offert aux utilisateurs. Le système est défini comme une collection d'objets qui interagissent entre eux pour réaliser une fonction répondant au service demandé. Les objets regroupent les données et les fonctions [].

13.4.1 Une approche objet : UML appliquée au cas APB

UML est un langage de modélisation visant à comprendre et décrire des besoins, à spécifier et documenter des systèmes, à présenter des architectures logicielles, à concevoir des solutions et à communiquer différents points de vue. Il s'agit d'un langage graphique permettant d'élaborer des éléments de modélisation et des éléments de visualisation, en proposant 13 types de diagrammes.

Le langage UML permet de décrire un système selon trois vues : fonctionnelle, structurelle et comportementale [Roq-06].

- La **vue fonctionnelle** a pour objet de décrire les fonctionnalités du système ainsi que son environnement (les acteurs). Plusieurs diagrammes sont produits :
  - le diagramme de cas d'utilisation montre les acteurs et les fonctions du système ;
  - le diagramme de description des cas d'utilisation : c'est un ensemble de diagrammes graphiques permettant de documenter les cas d'utilisation ;

— le diagramme d'activité décrit le (les) processus regroupant les activités, les acteurs et les objets au sein de l'organisation.

Avant la construction des diagrammes, une expression initiale des besoins avec la vision du projet, son positionnement, doit être exprimée sous forme d'objectifs à satisfaire. Des méthodes dites `Gore` (*Goal-Oriented Requirements Engineering*) ont fait leur apparition [?] depuis le début des années 90. Elles préconisent de définir le système sous forme d'objectifs à satisfaire avant d'entreprendre la définition de l'architecture du système.

Des hiérarchies d'objectifs, on peut déterminer le diagramme de cas d'utilisation (DCU). Le DCU décrit les besoins d'une façon simple de façon à être compréhensible par les utilisateurs. Il permet de spécifier les acteurs (environnement du système) et les cas d'utilisation (système).

Un *acteur* représente un *rôle* joué par une entité externe (humain, dispositif matériel ou autre système) qui *interagit* avec le système étudié. Un *cas d'utilisation* décrit le système étudié du point de vue de l'utilisateur. Il modélise un service rendu par le système à un utilisateur.

Dans le cas d'APB, le système devra regrouper toutes les fonctionnalités nécessaires à la gestion des vœux, à la gestion des catalogues de formation proposés par les établissements, au traitement des dossiers par les filières ainsi qu'à la génération des classements et enfin aux affectations (appariement candidat–établissement).

En prenant appui sur les documents cités dans le chapitre 6, relatifs à l'interface APB pour le responsable de la filière et l'interface APB pour le candidat, on peut imaginer le diagramme partiel suivant.

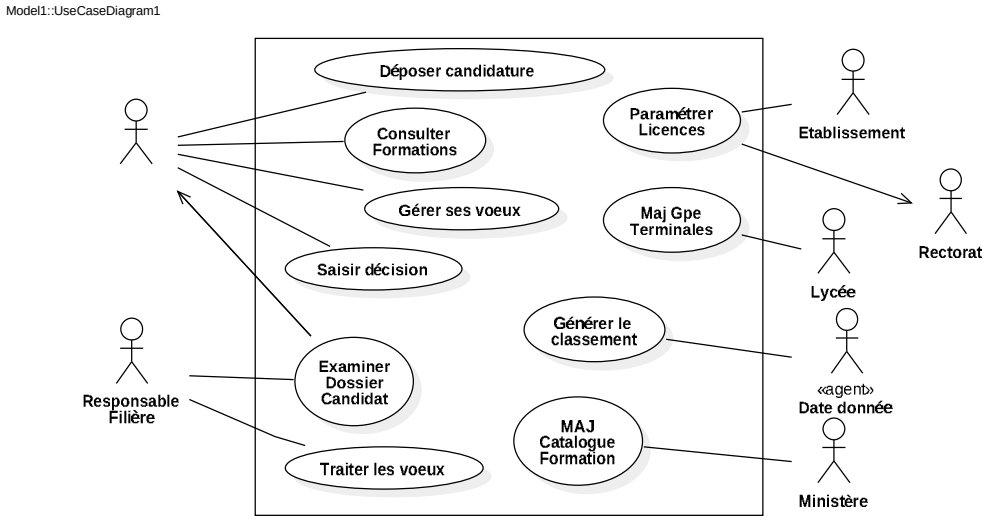


Figure 1.- Diagramme de cas d'utilisation du cas APB

De ce schéma (partiel, rappelons-le), on peut regrouper les fonctionnalités soit par

type d'acteur soit par type de procédure. Par exemple, on peut considérer un paquetage (terminologie d'UML), ou sous-système regroupant les fonctionnalités proposées à l'acteur Candidat ; de même qu'on peut considérer le paquetage 'Mise à jour des catalogues' dans lequel sera pris en compte les traitements en *back office* réalisés par les établissements. Le troisième paquetage peut concerner les examens de dossiers par les filières, etc. Cette décomposition conduit à définir une première architecture fonctionnelle à un haut niveau d'abstraction et, ce faisant, à faciliter la compréhension du système mais aussi sa mise en œuvre par les équipes de développement.

Chaque cas d'utilisation est décrit ensuite par un ensemble de scénarios. Un *scénario* représente une succession particulière d'enchaînements, s'exécutant du début à la fin du cas d'utilisation. L'objet d'une telle représentation permet d'identifier le détail de l'interaction entre l'acteur (celui qui utilise le système) et le système. Il est nécessaire d'identifier tous les cas possibles, d'où l'élaboration d'un scénario nominal, de scénarios alternatifs et de scénarios d'erreur.

La description des scénarios est faite soit en langue naturelle, soit sous forme graphique : le diagramme de séquence système et le diagramme d'activité système.

La description en langue naturelle suit un format préétabli dont voici un exemple :

Nom : S'inscrire dans l'application
Résumé : Ce cas d'utilisation permet à un candidat de s'inscrire dans APB
Acteurs : candidat à l'enseignement supérieur

Description des scénarios
Préconditions :
• Avoir son code INE
Scénario nominal
1. le lycéen saisit son n^o INE et sa date de naissance
2. le système contrôle la période d'inscription
3. le système affiche le formulaire
4. le candidat lycéen remplit le formulaire
5. le système renvoie un numéro et un mot de passe
6. le lycéen valide son adresse
7. le système lui envoie un code confidentiel
8. le lycéen saisit le code reçu dans son dossier personnel
Scénarios alternatifs
A1 : INE non conforme ou date erronée
L'enchaînement A1 démarre au point 1 du scénario nominal.
1a. Les données saisies sont erronées :
Le scénario nominal reprend au point 1.
A2 : Données du formulaire non valides
L'enchaînement A2 démarre au point 4 du scénario nominal.
4a. Le système indique au candidat que les informations saisies ne sont pas conformes
Le scénario nominal reprend au point 3.
A3 : code non reçu
L'enchaînement A3 démarre au point 7 du scénario nominal.
7a : Le candidat ne reçoit pas le code confidentiel.
Le scénario nominal reprend au point 5.

La vue fonctionnelle est une étape importante dans la compréhension des besoins et de leurs spécifications. Les modèles obtenus forment la base principale des exigences prises

en charge par le système. Le diagramme de cas d'utilisation est le modèle qui permet de recueillir, d'analyser les besoins.

- La **vue structurelle** a pour objet de décrire les concepts métier relatifs au sujet étudié. Elle regroupe quatre types de diagrammes :
  - Le diagramme de classes constitue l'ossature de la structure interne du système (donc du logiciel). Il permet d'identifier les objets dit métier qui doivent répondre à la problématique du métier ;
  - Le diagramme d'objet sert à valider le diagramme de classes ;
  - Le diagramme de composant représente l'architecture logicielle, c'est-à-dire tous les composants utilisés au fonctionnement du système, par exemple le système de gestion de bases de données utilisé dans APB est *Oracle*, c'est l'un parmi d'autres composants utilisé pour faire fonctionner le système ;
  - Le diagramme de déploiement décrit l'architecture matérielle : représente respectivement l'architecture logicielle et l'architecture matérielle sur laquelle s'exécutent les composants logiciels.

    Voici un diagramme de classes simplifié du cas APB :

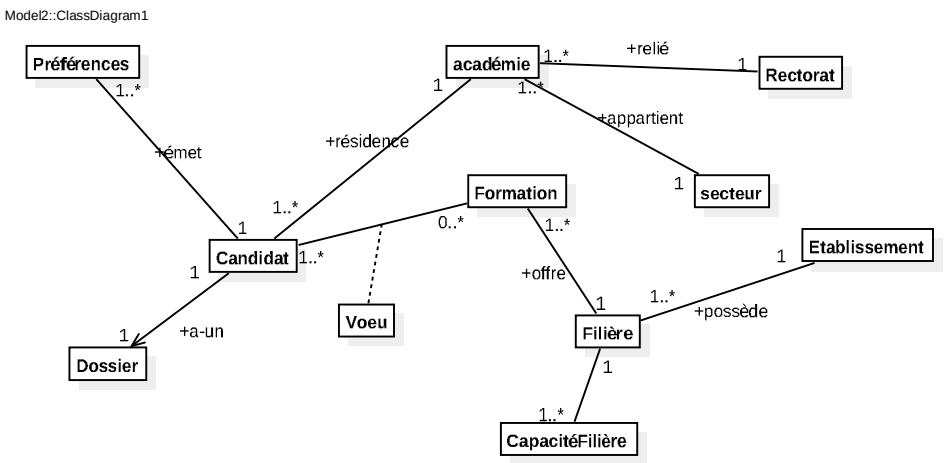
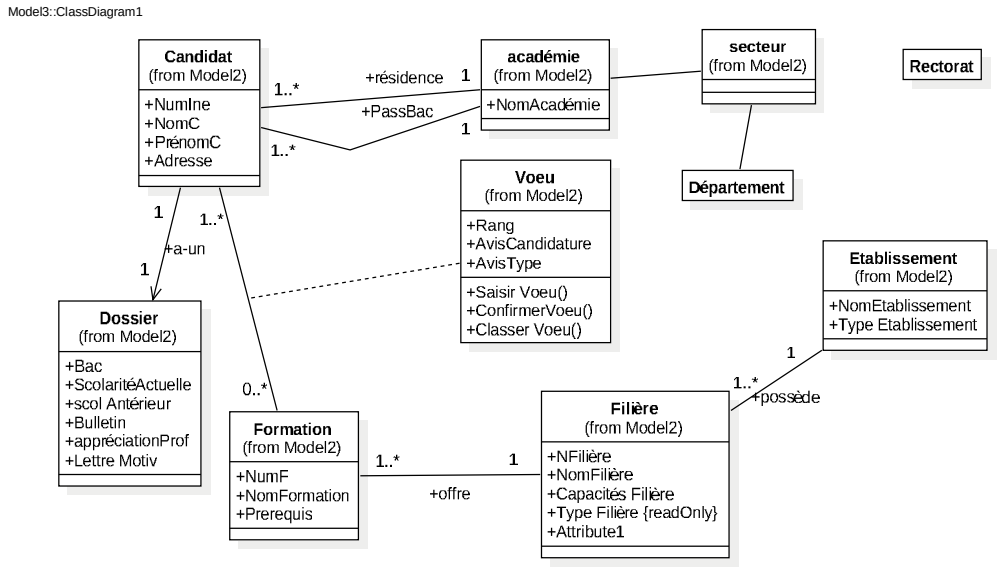


Fig. 2 : Diagramme de classe partiel d'APB



Chaque classe d'objets (représentant des objets de même nature) se définit par deux parties : les *attributs* décrivent les objets alors que les *méthodes* décrivent les opérations possibles sur les attributs. On voit sur la figure ci-dessus la classe ÉTABLISSEMENT, qui décrit tous les établissements concernés dans l'opération APB. L'association entre classes représente une relation sémantique entre classes. Par exemple, les vœux émis par les candidats lycéens sont représentés sous la forme d'une association entre les candidats et les formations sur lesquelles ils ont émis un vœu. La classe VŒU, qui est une classe association, intègre le vœu classé du candidat.

La classe GROUPE DE FORMATION représente les groupes de classement élaborés par le responsable de la formation d'accueil et l'association *Classé* entre les classes CANDIDAT et GROUPE DE FORMATION représente les candidats d'un groupe. L'association *affecté* représente le résultat de l'appariement candidat–formation.

La structure de la base de données à implémenter peut être obtenue à partir de ce diagramme de classes. Celle-ci peut être générée de façon automatique car les règles de passage du diagramme de classes à la base de données sont bien définies ; il existe même des outils permettant d'obtenir le programme de création de la base en langage SQL et ce quel que soit le SGBD utilisé Rajouter la structure de la BD du schéma de classes de la figure 2.

- La **vue comportementale** décrit le système pendant son exécution. Elle permet d'établir le lien entre la vision statique et la vision fonctionnelle et constitue ce que l'on appelle la *dynamique du système*.

  Plusieurs diagrammes sont proposés :

  — Le *diagramme de séquence* montre comment les objets réagissent pour réaliser un cas d'utilisation ;

— Le *diagramme d'états transition* permet d'étudier le comportement interne d'un objet en réaction à des événements discrets ;
— le *diagramme de timing* met l'accent sur les contraintes temporelles.

Dans cette étape, on est en pleine phase de conception technique avant la programmation. Il s'agit de représenter un système pendant son exécution. Pour chaque cas d'utilisation, il faut identifier les objets métier (du diagramme de classes déjà établi) nécessaires à la réalisation du cas d'utilisation. Aux objets métier vont être rajoutées les objets Dialogue (IHM) et les objets Contrôle. Les objets Dialogue représentent l'interface utilisateur et sont en liaison avec les objets contrôle qui contiennent la logique applicative et font la liaison avec les objets métier.

Le *diagramme de séquence* représente la réalisation d'un cas d'utilisation. Lorsque le candidat veut s'inscrire, le système affiche un formulaire (objet dialogue) dans lequel l'acteur candidat saisit les informations liées à son inscription. Les informations saisies sont transportées par envoi de message à l'objet contrôle. Celui-ci envoie à chaque objet métier les informations qui le concernent. Par exemple, les informations du formulaire induisent la création d'un objet informationnel candidat et la génération d'un mot de passe qui est transmis à l'acteur candidat.

Ces différents diagrammes participent à l'identification des opérations de chaque classe. Le diagramme de classe de conception obtenu représente, à l'issue de l'étude de la dynamique, l'architecture technique du système. Des outils peuvent être utilisés pour générer le code à partir du diagramme de classes de conception.

13.5 Conclusion

Dans ce chapitre, on a présenté le contexte dans lequel est initiée la construction d'un système d'information informatisé. Des caractéristiques du logiciel ainsi que les problèmes liés ont été présentés. Une brève description des étapes rentrant dans la construction d'un système a été faite. Nous avons insisté sur l'aspect d'analyse des besoins car c'est l'étape en amont qui est la plus importante, celle qui conduit au succès (ou à l'échec) d'un projet informatique. Il est bien reconnu que l'aspect technique, tout ce qui est autour de la programmation, est à ce jour plus ou moins maîtrisé.

Il est fortement recommandé de penser le système (par la multitude de modèles) avant sa réalisation. La maîtrise d'ouvrage doit être assurée que l'ensemble des lois et règles sont mises en œuvre conformément à l'esprit qui a été à l'origine de leur définition. Il faut mettre en place des règles permettant d'attester de la conformité de la conception aux besoins des utilisateurs et à la réglementation.

13.6 Bibliographie

[BB-14] BADREA, S. et BOULANGER, J. L., **Ingénierie des exigences**, Dunod, 2014.

[Boe-81] BOEHM, Barry, **Software Engineering Economics**, Prentice Hall, 1981.

[Bro-87] **Brooks** Frederick P., *No silver Bullet : essence and accidents of Software Engineering*, **IEEE Computer**, Vol. 20, 1987. Traduction française dans **Le mythe du mois-homme**, International Thomson publishing France, seconde éd., 2001, 276 p.

[BT-76] BELL, T. E. et THAYER, T. A., *Software Requirements : are they really a problem ?*, in **Proc. of Int. Conf. on Software Engineering** (ICSE), 1976.

[Lon-15] LONCHAMP, Jacques, **Analyse des besoins et spécifications**, Dunod, 2015.

[McI-76] MCILROY, M., *Mass-produced Software Components*, in **Software Engineering Concepts and Techniques**, NATO Conference Engineering, J.M. Buxton et al. eds, 1976.

[MHL-08] MORLEY, C., HUGUES, J. et LEBLANC, B., **UML2 pour l'analyse d'un système d'information**, Dunod, 2008.

[Rei-00] REIX, Robert. **Systèmes d'information et management des organisations**, deuxième édition, Vuibert, 2000.

[Roq-06] ROQUES, P., **UML2 par la pratique**, Eyrolles, 2006.

[Ros-72]

[Sta-01] The standish Group, **Extreme Chaos**, Technical report, 2001.