CHAPITRE 1

Introduction à la modélisation objet

Les techniques de programmation n'ont cessé de progresser depuis l'époque de la programmation par cartes perforées à nos jours. Cette évolution a toujours été dictée par le besoin de concevoir et de maintenir des applications toujours plus complexes.

La programmation par cartes perforées, switch ou câblage (de 1800 à 1940) a ainsi fait place à des techniques plus évoluées, comme l'assembleur (1947), avec l'arrivée de l'ordinateur électronique né des besoins de la guerre. Des langages plus évolués ont ensuite vu le jour comme Fortran en 1956 ou Cobol en 1959. Jusque-là, les techniques de programmation étaient basées sur le branchement conditionnel et inconditionnel (goto) rendant les programmes importants extrêmement difficiles à développer, à maîtriser et à maintenir.

La programmation structurée (Pascal en 1970, C en 1972, Modula et Ada en 1979…) a alors vu le jour et permis de développer et de maintenir des applications toujours plus ambitieuses. L'algorithmique ne se suffisant plus à elle seule à la fin des années 1970, le génie logiciel est venu placer la méthodologie au cœur du développement logiciel. Des méthodes comme Merise (1978) se sont alors imposées.

La taille des applications ne cessant de croître, la programmation structurée a également rencontré ses limites, faisant alors place à la programmation orientée objet (Simula 67 en 1967, Smalltalk en 1976, C++ en 1982, Java en 1995…). La technologie objet est donc la conséquence ultime de la modularisation dictée par la maîtrise de la conception et de la maintenance d'applications toujours plus complexes. Cette nouvelle technique de programmation a nécessité la conception de nouvelles méthodes de modélisation.

Au milieu des années 1990, trois méthodes s'imposaient dans le domaine de la modélisation objet : OMT, Booch et OOSE. UML (*Unified Modeling Language* en anglais, soit langage de modélisation objet unifié) est né de la fusion de ces trois méthodes et est accepté par l'OMG (*Object Management Group*) en novembre 1997 dans sa version 1.1. La version d'UML en cours en 2009 est UML 2.2 qui s'impose plus que jamais en tant que langage de modélisation objet standardisé pour la modélisation des logiciels.

La première section de ce chapitre (section 1.1) a pour objectif de présenter brièvement la problématique du génie logiciel et de la modélisation. La section 1.2 présente un éventail des cycles de vie les plus courants d'un logiciel. Cette section est à rapprocher du chapitre final (chapitre 10) qui aborde le problème de la mise en œuvre d'UML. La section 1.3 permet d'avoir un aperçu des principales différences qui opposent la programmation structurée à l'approche orientée objet. Cette section introduit également les concepts fondamentaux de la pensée objet. La dernière section 1.4 introduit enfin les différents diagrammes du langage de modélisation UML.

Sommaire

1 Génie logiciel et modélisation

1.1 Problématique du génie logiciel

✓ L'informatisation

L'informatisation est le phénomène le plus important de notre époque. Elle s'immisce maintenant dans la plupart des objets de la vie courante et ce, que ce soit dans l'objet proprement dit [1], ou bien dans le processus de conception ou de fabrication de cet objet.

Actuellement, l'informatique est au cœur de toutes les grandes entreprises. Le système d'information d'une entreprise est composé de matériels et de logiciels. Plus précisément, les investissements dans ce système d'information se répartissent généralement de la manière suivante : 20% pour le matériel et 80% pour le logiciel. En effet, depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement. Ce matériel est relativement fiable et le marché est standardisé. Les problèmes liés à l'informatique sont dorénavant essentiellement des problèmes de logiciel.

✓ Le logiciel

Le terme français *logiciel* a été introduit par l'arrêté du 22 décembre 1981 relatif à l'enrichissement du vocabulaire de l'informatique (Journal officiel du 17 janvier 1982) qui en donne la définition suivante :

> ensemble des programmes, et éventuellement la documentation, relatifs au fonctionnement d'un ensemble de traitements de l'information.

Il s'agit donc d'un ensemble de programmes qui permet à un système informatique d'assurer une tâche particulière (logiciel de comptabilité, logiciel de gestion des prêts, jeux…).

Les logiciels, suivant leur taille, peuvent être développés par une personne seule, une petite équipe, ou un ensemble d'équipes coordonnées. Le développement de grands logiciels par de grandes équipes pose d'importants problèmes de conception et de coordination. Or, le développement d'un logiciel est une phase absolument cruciale qui monopolise l'essentiel du coût d'un produit [2] et conditionne sa réussite et sa pérennité.

✓ La crise du logiciel

En 1995, une étude du *Standish Group* dressait un tableau accablant de la conduite des projets informatiques. Reposant sur un échantillon représentatif de 365 entreprises, totalisant 8380 applications, cette étude établissait que :

- $16,2\%$ seulement des projets étaient conformes aux prévisions initiales,
- $52,7\%$ avaient subi des dépassements en coût et délai d'un facteur 2 à 3 avec diminution du nombre des fonctions offertes,

1. Par exemple, aujourd'hui, 90% des nouvelles fonctionnalités des automobiles sont apportées par l'électronique et l'informatique embarquées.
2. Comparativement à sa production, le coût du développement d'un logiciel est extrêmement important. Nous verrons par la suite que la maintenance coûte également très cher.

– 31,1% ont été purement abandonnés durant leur développement.

Pour les grandes entreprises (qui lancent proportionnellement davantage de gros projets), le taux de succès est de 9% seulement, 37% des projets sont arrêtés en cours de réalisation, 50% aboutissent hors délai et hors budget.

L'examen des causes de succès et d'échec est instructif : la plupart des échecs proviennent non de l'informatique, mais de la maîtrise d'ouvrage (c.-à-d. le client). Pour ces raisons, le développement de logiciels dans un contexte professionnel suit souvent des règles strictes encadrant la conception et permettant le travail en groupe et la maintenance du code. Ainsi, une nouvelle discipline est née : le *génie logiciel*.

✓ Le génie logiciel

Le génie logiciel est un domaine de recherche qui a été défini (fait rare) du 7 au 11 octobre 1968, à Garmisch-Partenkirchen, sous le parrainage de l'OTAN. Il a pour objectif de répondre à un problème qui s'énonçait alors en deux constatations : d'une part, les logiciels n'étaient pas fiables, d'autre part, il était incroyablement difficile de réaliser dans des délais prévus des logiciels satisfaisant leur cahier des charges. Le terme français de *génie logiciel* a été introduit par l'arrêté ministériel du 30 décembre 1983 relatif à l'enrichissement du vocabulaire de l'informatique (Journal officiel du 19 février 1984) et désigne :

> l'ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi.

La notion de suivi (c.-à-d. de maintenance) occupe une place importante dans le génie logiciel. Une enquête effectuée aux USA en 1986 auprès de 55 entreprises révèle à ce sujet que 53% du budget total d'un logiciel est affecté à la maintenance. Jaulent (1992) propose une définition plus pratique du terme *génie logiciel* :

> procédures, méthodes, langages, ateliers, imposés ou préconisés par les normes adaptées à l'environnement d'utilisation afin de favoriser la production et la maintenance de composants logiciels de qualité.

Le génie logiciel s'intéresse particulièrement à la manière dont le code source d'un logiciel est spécifié puis produit. Plus largement, le génie logiciel est concerné par l'intégralité du cycle de vie des logiciels (cf. section 1.2) de l'analyse du besoin à la maintenance, en passant par les étapes d'élaboration des spécifications, de conceptualisation, de développement et de tests. Les projets relatifs à l'ingénierie logicielle sont généralement de grande envergure et dépassent souvent les 10 000 lignes de code. C'est pourquoi ces projets nécessitent une équipe de développement bien structurée. La gestion de projet se retrouve naturellement intimement liée au génie logiciel.

1.2 Pourquoi et comment modéliser ?

✓ Qu'est-ce qu'un modèle ?

Un *modèle* est une représentation abstraite et simplifiée (c.-à-d. qui exclut certains détails), d'une entité (phénomène, processus, système…) du monde réel en vue de le

décrire, de l'expliquer ou de le prévoir. Modèle est synonyme de théorie, mais avec une connotation pratique : un modèle, c'est une théorie orientée vers l'action qu'elle doit servir. Concrètement, un modèle permet de réduire la complexité d'un phénomène en éliminant les détails qui n'influencent pas son comportement de manière significative. Il reflète ce que le concepteur croit important pour la compréhension et la prédiction du phénomène modélisé. Les limites du phénomène modélisé dépendant des objectifs du modèle.

Ci-dessous sont énumérés quelques exemples de modèles.

Modèle météorologique - À partir de données d'observation (satellite…), un modèle météorologique permet de prévoir les conditions climatiques pour les jours à venir.

Modèle économique - Un modèle économique peut être utilisé pour simuler l'évolution de cours boursiers en fonction d'hypothèses macro-économiques (évolution du chômage, taux de croissance…).

Modèle démographique - Un modèle démographique définit la composition d'un panel d'une population et son comportement, dans le but de fiabiliser des études statistiques, d'augmenter l'impact de démarches commerciales…

Plans - Les plans sont des modèles qui donnent une vue d'ensemble d'un système donné. Par exemple, dans le bâtiment, pour la construction d'un immeuble, il faut préalablement élaborer de nombreux plans comme :
- plans d'implantation du bâtiment dans son environnement ;
- plans généraux du bâtiment et de sa structure ;
- plans détaillés des différents locaux, bureaux, appartements…
- plans des câblages électriques ;
- plans d'écoulements des eaux…

Les trois premiers exemples sont des modèles qualifiés de prédictifs. Le dernier, plus conceptuel, possède différents niveaux de vues comme la plupart des modèles en génie logiciel.

✓ Pourquoi modéliser ?

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence. Un modèle est un langage commun, précis, qui est connu par tous les membres de l'équipe et qui constitue, à ce titre, un vecteur privilégié pour communiquer. Cette communication est essentielle pour aboutir à une compréhension précise d'un problème donné et commune aux différentes parties prenantes (notamment entre la maîtrise d'ouvrage et la maîtrise d'œuvre informatique).

Dans le domaine de l'ingénierie du logiciel, le modèle permet de mieux répartir les tâches et d'automatiser certaines d'entre elles. C'est également un facteur de réduction des coûts et des délais. Par exemple, les plateformes de modélisation savent maintenant exploiter les modèles pour faire de la génération de code (au moins au niveau du squelette) voire des aller-retours entre le code et le modèle sans perte d'information. Le modèle est

enfin indispensable pour assurer un bon niveau de qualité et une maintenance efficace. En effet, une fois mise en production, l'application va devoir être maintenue, probablement par une autre équipe et, qui plus est, pas nécessairement de la même société que celle ayant créé l'application.

Le choix du modèle a donc une influence capitale sur les solutions obtenues. Les systèmes complexes sont mieux modélisés par un ensemble de modèles indépendants. Selon les modèles employés, la démarche de modélisation n'est pas la même.

✓ Maîtrise d'ouvrage ou maîtrise d'œuvre, qui doit modéliser ?

La **maîtrise d'ouvrage** (MOA) est le client qui passe commande d'un produit nécessaire à son activité. Il s'agit généralement d'une personne morale (entreprise, direction, entité d'une organisation…) et non d'une personne physique.

La **maîtrise d'œuvre** (MOE) est une personne morale garante de la bonne réalisation technique du produit commandé par la MOA. Soit la MOE réalise elle-même le produit, soit elle passe commande à un ou plusieurs fournisseurs. Dans ce dernier cas, la MOE coordonne l'action des fournisseurs en contrôlant la qualité technique, en assurant le respect des délais fixés par la MOA et en minimisant les risques. La MOE est responsable de la qualité technique de la solution. Elle doit, avant toute livraison à la MOA, procéder aux vérifications nécessaires (recette).

La modélisation est souvent faite par la maîtrise d'œuvre informatique (MOE). C'est malencontreux, car les priorités de la MOE résident dans le fonctionnement de la plateforme informatique et non dans les processus de l'entreprise. Il est préférable que la modélisation soit réalisée par la maîtrise d'ouvrage (MOA) de sorte que le métier soit maître de ses propres concepts. La MOE doit intervenir dans le modèle lorsque, après avoir défini les concepts du métier, il faut introduire les contraintes propres à la plateforme informatique.

Il est vrai que certains métiers, dont les priorités sont opérationnelles, ne disposent pas toujours de la capacité d'abstraction et de la rigueur conceptuelle nécessaire à la formalisation. La professionnalisation de la MOA a pour but de les doter de ces compétences. Cette professionnalisation réside essentiellement dans l'aptitude à modéliser le système d'information du métier : le maître mot est *modélisation*. Lorsque le modèle du système d'information est de bonne qualité, sobre, clair, stable, la MOE peut travailler dans de bonnes conditions. Dans la réalité, la MOA n'a souvent pas la compétence nécessaire à la modélisation. Les méthodes de développement actuelles sont conscientes du problème. Pour y répondre, les *méthodes Agiles* (cf. section 10.1.3) font, de la collaboration avec le client, une priorité. Pour aller encore plus loin, *eXtreme Programming* (cf. section 10.1.5) va jusqu'à exiger la présence à plein temps pendant toute la durée du projet d'un représentant de la MOA.

2 Cycle de vie d'un logiciel

Le *cycle de vie d'un logiciel* (en anglais *software lifecycle*), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.

2.1 Étapes du cycle de vie d'un logiciel

Le cycle de vie d'un logiciel s'articule généralement autour des étapes qui suivent.

Définition des objectifs : Cette étape consiste à définir la finalité du projet et son inscription dans une stratégie globale.

Analyse des besoins et faisabilité : Cette étape concerne l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes. Il s'agit également d'estimer la faisabilité de ces besoins.

Spécifications ou conception générale : Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.

Conception détaillée : Cette étape consiste à définir précisément chacun des sous-ensembles du logiciel.

Codage (Implémentation ou programmation) : C'est la traduction dans un langage de programmation des fonctionnalités définies lors des phases de conception.

Tests unitaires : Ils permettent de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.

Intégration : L'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel grâce à différents tests d'intégration.

Qualification (ou recette) : C'est la vérification de la conformité du logiciel aux spécifications initiales.

Documentation : Elle vise à produire les informations nécessaires pour l'utilisation du logiciel et pour ses développements ultérieurs.

Mise en production : C'est le déploiement sur site du logiciel.

Maintenance : Elle comprend toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

La séquence et la présence de chacune de ces activités dans le cycle de vie dépendent du choix d'un modèle de cycle de vie entre le client et l'équipe de développement. Une méthode de développement permet de prendre en compte, en plus des aspects techniques du cycle de vie, l'organisation et les aspects humains (cf. section 10.1).

2.2 Modèles de cycles de vie linéaires

✓ Cycle de vie en tunnel

Ce modèle de cycle de vie, représenté sur la figure 1.1, cache en fait l'absence de modèle de développement. Ce cycle ne comporte pas d'étape ou de document intermédiaires. Il est très difficile de se faire une idée de son état d'avancement. Ce modèle de développement n'est adapté qu'aux microprojets.

✓ Cycle de vie en cascade

Le modèle de cycle de vie en cascade (cf. figure 1.2) a été mis au point dès 1966, puis formalisé aux alentours de 1970. Le principe de ce modèle est très simple et facilement compréhensible par toutes les parties impliquées dans le projet. Il consiste à enchaîner les phases dans un déroulement séquentiel et linéaire depuis les spécifications jusqu'à la mise en production. Chaque phase correspond à une activité qui se termine à une date précise par la production de certains documents. Ces documents sont utilisés comme source d'information, voire comme spécifications, par la phase suivante. Bien plus tard, ils sont également utilisés pour les revues de validation. À l'issue d'une phase, ces documents sont soumis à un examen approfondi et le passage à la phase suivante ne peut se faire que s'ils sont jugés satisfaisants et validés. L'intention originelle de ce modèle est de ne jamais revenir sur une phase terminée.

Un inconvénient majeur de ce modèle est qu'il ne supporte aucune insuffisance sur les étapes passées et qu'il est incapable de prendre en compte des évolutions au cours de son cycle. Il exclut donc toute allée et venue entre le cahier des charges et la conception du produit. Bien que le modèle original ne comporte pas de possibilité de retour en arrière, celui-ci a été rajouté ultérieurement sur la base qu'une étape ne peut remettre en cause que l'étape précédente, ce qui s'avère insuffisant dans la pratique. Un autre inconvénient majeur de ce modèle est la vérification bien trop tardive du bon fonctionnement du système lors de la phase d'intégration, ou pire, lors de la mise en production.

✓ Cycle de vie en V

Le modèle en V, représenté figure 1.3, est une variante du modèle en cascade qui met l'accent sur les tests. Un plan de test est préparé avant l'implémentation pour chaque niveau de spécifications : toute décomposition doit être accompagnée de la description de sa recomposition, toute spécification d'un composant doit être accompagnée des tests qui permettront de s'assurer de son adéquation. Ceci rend explicite la préparation des dernières phases (validation-vérification) par les premières (construction du logiciel), et permet ainsi d'éviter un écueil bien connu de la spécification du logiciel : énoncer une propriété qu'il est impossible de vérifier objectivement après la réalisation.

Cependant, ce modèle ne résout en rien les défauts majeurs du modèle de cycle de vie en cascade.