Analyse des besoins pour le développement logiciel

Recueil et spécification, démarches itératives et agiles

Jacques Lonchamp
Professeur des universités

DUNOD

Illustration de couverture :
Grey abstract communication © iStock.com/nnnnae

5 rue Laromiguière, 75005 Paris
www.dunod.com

ISBN 978-2-10-072994-4

Table des matières

Avant-propos

POURQUOI CET OUVRAGE ?

Le présent ouvrage est l'aboutissement d'une longue pratique de l'enseignement du développement logiciel à divers publics universitaires et de formation continue. Plus précisément, ce sont les insatisfactions éprouvées à la lecture des documents pédagogiques existants, à l'occasion d'une réflexion nationale sur la formation des informaticiens, qui ont constitué l'élément déclencheur de sa rédaction.

Sa première moitié est consacrée aux *styles et processus de développement* des applications informatiques. Sa seconde moitié détaille les techniques utilisables lors de toutes les activités en amont de la conception, qui vont *du recueil des besoins à la spécification de l'application*. Dans la suite du volume, cet ensemble d'activités sera désigné par l'expression « analyse des besoins » ou, encore plus simplement, par le terme « analyse ».

Cet ouvrage reprend les principes d'un premier volume paru dans la même collection et consacré de manière complémentaire à la *conception* des applications (*Conception d'applications en Java/JEE. Principes, patterns et architectures*. Jacques Lonchamp, Dunod, 2014). Les principes communs qui sous-tendent ces deux volumes sont résumés au paragraphe suivant.

Alors qu'il est possible de parler de la conception de manière assez indépendante des styles et processus de développement logiciel, ce n'est pas du tout le cas pour l'analyse : on ne la pratique pas de la même manière dans un développement « agile », fondé sur l'adaptation continue aux évolutions des besoins grâce à un dialogue permanent avec le client et des itérations courtes, et dans un développement « classique », fondé sur le recueil initial exhaustif des besoins et la planification rigide de toutes les activités de production. C'est ce qui justifie le regroupement au sein d'un même volume de la description des styles et

processus de développement avant la présentation des connaissances théoriques et pratiques relatives à l'analyse.

LA CONCEPTION DE L'OUVRAGE

Le positionnement dans le cursus

Un cursus de formation à l'informatique ressemble à une fusée à trois étages.

L'étage *disciplinaire* vise l'acquisition des savoirs conceptuels et techniques de base, grâce à des enseignements ciblés vers des domaines bien délimités.

L'étage *intégratif* doit permettre de tisser des liens entre les savoirs disciplinaires, de les approfondir en conséquence, et de prendre conscience des enjeux réels dans le monde professionnel.

L'étage *professionnalisant*, qui s'appuie sur la compréhension globale des grandes thématiques que procure l'étage précédent, vise l'approfondissement de thèmes techniques spécialisés permettant de passer de la *compréhension* à la *maîtrise effective* de certaines facettes du métier.

Les deux ouvrages proposés, sur la conception d'une part et sur les processus et l'analyse d'autre part, relèvent clairement de la phase « intégrative », pour laquelle le manque de documents pédagogiques est criant.

Une progression logique des apprentissages

Dans le volume sur la conception, les savoirs en conception ont été ancrés dans les connaissances préalables en programmation. En effet, tout programmeur Java utilise au sein du JDK, sans en être nécessairement conscient, les patrons (*patterns*) de conception les plus connus et les plus utiles. Leur compréhension et leur mémorisation se trouvent grandement facilités quand cet ancrage est rendu explicite.

Dans ce second volume consacré aux concepts et méthodes de l'analyse, l'importance de la connaissance préalable des styles et processus de développement a déjà été évoquée. Par ailleurs, beaucoup de concepts de l'analyse prennent du sens à travers la compréhension de ce que peut être une conception et une réalisation de qualité, comme la modélisation de l'application en termes de classes frontières, de classes de contrôle et de classes entité. Une progression logique des apprentissages en développement des applications peut donc se schématiser par :

programmation $\longrightarrow$ *conception* $\longrightarrow$ *styles et processus de développement* $\longrightarrow$ *analyse*.

Pour une « culture générale » de l'analyse

Le côté très parcellaire de beaucoup de documents existants est aux antipodes de ce qu'on peut attendre pour la phase « intégrative » du cursus : l'analyse ne se réduit pas aux seuls besoins fonctionnels, la modélisation au seul langage UML et les processus aux seules approches agiles.

Sans tomber dans le travers encyclopédique des énormes « bibles » du génie logiciel (comme [Pre09] ou [Som04]), cet ouvrage cherche à donner une *présentation synthétique large* des thèmes abordés, une forme de « culture générale » de l'analyse, tout en approfondissant les points essentiels des pratiques les plus courantes.

L'importance de la mise en pratique

Beaucoup d'ouvrages de synthèse ne proposent ni exercices d'application ni études de cas, au contraire de ceux centrés sur des thématiques spécialisées. Or toute faiblesse sur ce point risque d'accréditer l'idée auprès des étudiants que ces documents se limitent à un *discours théorique*, éloigné de la pratique de terrain, et qui peut donc être négligé.

Au contraire, cet ouvrage propose près de 70 exercices d'application tous corrigés, associés à chacun des chapitres, et deux études de cas finales très détaillées. La majorité des exercices ont été glanés sur le Web et retravaillés. Que leurs auteurs originaux, souvent impossibles à déterminer, soient ici remerciés collectivement.

POUR QUELS LECTEURS ?

Cet ouvrage pédagogique s'adresse prioritairement aux étudiants de deuxième et troisième année des cursus spécialisés en informatique, quelle que soit leur nature (DUT/LP, L2/L3, deuxième et troisième années d'écoles d'ingénieurs).

Il peut bien entendu être également utile à tous les praticiens de l'informatique qui souhaitent rafraîchir ou consolider leurs connaissances en processus de développement et analyse des besoins.

LE PLAN DE L'OUVRAGE

La première partie s'attache à présenter de manière synthétique les différentes facettes du développement logiciel en insistant plus particulièrement sur les activités de l'ingénierie des besoins en amont de la conception des applications qui sont au cœur de l'ouvrage.

- Le **chapitre 1** définit le logiciel et la problématique de son développement au sein des organisations.
- Le **chapitre 2** est une introduction à l'ensemble des activités du développement logiciel.
- Le **chapitre 3** discute de la modélisation, qui est à la base de toute la démarche d'analyse et de conception, et donne quelques rappels sur la notation UML.
- Le **chapitre 4** présente les différentes organisations du développement logiciel, appelées communément « modèles de cycle de vie ».
- Le **chapitre 5** entre dans le détail des « méthodes » de développement objet les plus en vogue actuellement : *(Rational) Unified Process* – (R)UP, *eXtreme Programming* – XP, *Scrum*.

Les trois parties suivantes du livre décrivent les principales techniques et notations utilisables concrètement pendant l'analyse.

La deuxième partie est consacrée aux techniques de la modélisation métier.

– Le **chapitre 6** introduit la problématique de la modélisation métier et de l'utilisation du langage UML à cette fin.
– Le **chapitre 7** détaille la modélisation des processus métier.
– Le **chapitre 8** décrit la modélisation des entités métier ou modélisation du domaine.
– Le **chapitre 9** introduit la spécification formelle d'expressions avec le langage OCL (*Object Constraint Language*) qui sert souvent à préciser les modèles de classes du domaine. Mais ce langage peut bien entendu être utilisé dans d'autres contextes et avec d'autres modèles UML.

La troisième partie est dédiée aux techniques de la modélisation des besoins.

– Le **chapitre 10** présente l'utilisation des *user stories* (scénarios client).
– Le **chapitre 11** décrit l'utilisation des cas d'utilisation (*use cases*).
– Le **chapitre 12** aborde l'utilisation d'autres modèles UML en complément des cas d'utilisation : les diagrammes de séquences « système » et les diagrammes d'activités des cas.

La quatrième partie est consacrée aux techniques de la modélisation des applications.

– Le **chapitre 13** présente le modèle des classes d'analyse.
– Le **chapitre 14** traite des modélisations UML complémentaires au modèle des classes d'analyse sous la forme de diagrammes de séquences et de diagrammes d'états.
– Le **chapitre 15** aborde les aspects liés aux interfaces homme-machine (IHM) à travers le modèle de navigation.

La cinquième et dernière partie du livre présente en détail deux études de cas.

La première porte sur l'analyse d'un site de commerce électronique selon les préconisations du processus unifié (R)UP.

– Le **chapitre 16** décrit l'unique itération de la phase d'initialisation.
– Le **chapitre 17** décrit la première itération de la phase d'élaboration.

La deuxième étude de cas porte sur l'analyse d'un jeu de plateau (type Monopoly).

– Le **chapitre 18** décrit informellement les règles du jeu, puis donne une spécification de l'application correspondante sous forme de *user stories*.
– Le **chapitre 19** donne une spécification de l'application en termes de cas d'utilisation et de diagramme de cas.
– Le **chapitre 20** décrit le modèle des classes du domaine et illustre les diagrammes d'états associés aux entités complexes du jeu.

Chapitre 1

Introduction

1.1 LE LOGICIEL

1.1.1. L'importance du logiciel

Beaucoup d'aspects de notre quotidien ne peuvent être imaginés aujourd'hui sans logiciel. C'est le cas entre autres des domaines du transport, du commerce, des communications, de la médecine, des finances ou des loisirs. Cette omniprésence du logiciel fait que nos vies et le fonctionnement de nos sociétés dépendent fortement de sa *qualité*.

Les erreurs logicielles peuvent causer de vrais désastres, économiques ou sanitaires. Un exemple emblématique est l'explosion de la première fusée Ariane 5 en 1996, qui a coûté plus d'un demi-milliard de dollars, à cause du dépassement de capacité d'une variable lors d'un calcul. Dans un registre différent, tout le monde se souvient de la grande peur du « bug de l'an 2000 » et de son coût astronomique, estimé entre 300 et 5000 milliards de dollars !

1.1.2. La diversité du logiciel

Il existe une grande variété de logiciels et de nombreuses manières de les classer.

Une première différenciation oppose les logiciels « génériques », ou progiciels, qui sont vendus en grand nombre comme des produits de consommation, aux logiciels « spécifiques » qui sont développés pour un contexte et un client particulier.

Une seconde différenciation repose sur la nature de la dépendance entre les logiciels et leurs environnements. Elle distingue trois classes :

– les logiciels « autonomes » (*standalone*), comme les traitements de texte, les logiciels de calcul ou les jeux,
– les logiciels qui gèrent des processus (*process support*), aussi bien industriels que de gestion,
– les logiciels « embarqués » dans des systèmes (*embedded*), comme dans les transports, la téléphonie ou les satellites.

Dans cette dernière classe, on a coutume de parler plutôt de « développement système » que de « développement logiciel », car les problématiques matérielles et logicielles y sont indissociables.

Une troisième différenciation sépare les logiciels isolés des « lignes de produits logiciels ». Celles-ci regroupent une famille de logiciels présentant de fortes similarités et un même domaine d'application, comme des applications pour téléphones mobiles. Elles sont développées à partir d'éléments réutilisables (*assets*) qui peuvent être des exigences, des modèles, des codes (composants), des plans de tests, etc.

Dans les grandes organisations, on particularise souvent ce qu'on appelle les « logiciels du patrimoine » (*legacy software*). Ce sont les applications les plus anciennes, complexes et peu documentées, auxquelles on touche le moins possible tant qu'elles remplissent correctement leurs fonctions en général vitales pour l'organisation.

Enfin les « logiciels web », ou « applications web », sont souvent considérés comme une catégorie à part. Ce sont des logiciels hébergés sur un serveur web et manipulables via les réseaux, Internet ou réseaux locaux, grâce à un navigateur web. On peut citer comme particularités notables :

– les accès concurrents qu'ils subissent, avec une charge difficilement prédictible,
– leurs exigences élevées de performance, de disponibilité et de sécurité,
– la prédominance des données par rapport aux traitements,
– leurs besoins fréquents d'évolution,
– l'importance de l'ergonomie et de l'esthétique dans leur conception.

On comprend que la notion de qualité et les exigences qui s'y attachent peuvent varier considérablement pour toute cette diversité de logiciels.

1.1.3. La complexité du logiciel

Techniquement, le terme « logiciel » désigne un riche ensemble d'artefacts incluant :

– des codes sources et des exécutables,
– des programmes et des données de test,
– des fichiers de configuration,
– des documentations « externes » à destination des utilisateurs,
– des documentations « internes » à destination de l'équipe de développement, etc.

1.2 LE DÉVELOPPEMENT LOGICIEL

1.2.1. Les acteurs du développement

Le développement logiciel est une activité intellectuelle à forte composante humaine (*human-intensive*). En 2013, le secteur du conseil, des études et des services informatiques en France comptait 575 000 emplois, hors fonction publique d'État.

Pendant le développement d'un logiciel, outre les membres de l'équipe de développement proprement dite, les autres acteurs à considérer sont le client (l'investisseur), les utilisateurs finaux et d'éventuels sous-traitants.

1.2.2. Les contextes du développement

Le logiciel peut être développé dans plusieurs contextes assez différents.

– Chez des éditeurs de logiciels qui commercialisent les progiciels qu'ils développent.

– Dans des Sociétés de service en ingénierie informatique (SSII), également nommées Entreprises de services du numérique (ESN), qui répondent aux besoins d'externalisation des projets informatiques, en développant des logiciels pour le compte de leurs clients.

– En interne, dans des organisations de toutes natures, entreprises, administrations et associations.

– Sur Internet, au moins en partie par des développeurs bénévoles, pour les logiciels libres.

1.2.3. Les activités du développement

Schématiquement, le développement logiciel consiste à transformer une idée ou un besoin en un logiciel fonctionnel. L'idée est produite par un client ou *maître d'ouvrage* (MOA). Le logiciel correspondant est développé par un fournisseur ou *maître d'œuvre* (MOE), puis exploité par des utilisateurs finaux. Lorsqu'il s'agit d'entités on parle de *Maîtrise d'ouvrage* (MOA) et de *Maîtrise d'œuvre* (MOE). La MOA peut être assistée en externe et en interne, souvent par d'anciens informaticiens ayant une bonne pratique de la conduite des projets : on parle de MOAD, pour *maîtrise d'ouvrage déléguée*.

FIGURE 1.1 Le développement logiciel

Comme cela sera détaillé dans la suite de l'ouvrage, le développement logiciel comprend en amont une phase centrée sur les *besoins* (ou exigences). Il s'agit pour toutes les parties prenantes d'un projet informatique de s'accorder sur l'ensemble des besoins auxquels devra répondre le système ou logiciel à construire. Le terme d'« ingénierie des besoins » ou « ingénierie des exigences » (*requirements engineering*) caractérise toutes les activités liées