Chapitre I - Introduction et conseils au lecteur

Cette partie introductive situe la place de l'algorithmique dans le développement logiciel et fournit au lecteur des conseils : conseils pour bien analyser un problème et concevoir un algorithme, conseils pour rédiger un algorithme, conseils pour prévoir des jeux d'essais.

Les conseils donnés dans cette partie découlent de notre volonté de faire profiter le lecteur de notre expérience. C'est en voyant des générations d'étudiants se pencher sur des énoncés, en discutant avec eux et en comprenant leurs difficultés que nous avons élaboré ces conseils.

Ils doivent être vus comme une aide, non comme une contrainte. Il est possible de passer outre, mais ce ne sera pas sans risque (de se tromper), sans efforts improductifs, sans errements... Nous ne saurions trop conseiller au lecteur de lire ce chapitre avant d'aborder des exercices du livre.

Nous pensons également que **celui-ci ne devra pas hésiter à y revenir lorsqu'il rencontrera une difficulté, dans l'analyse d'un problème, dans la recherche d'une solution d'un exercice...**

Malgré toute notre volonté, nous n'avons pas pu éviter, dans ce chapitre, certaines références en avant et il est fort possible qu'un lecteur débutant ait parfois du mal avec certains concepts qui sont utilisés avant même que nous les ayons définis, expliqués... Rappelons quand même que ce livre ne remplace pas un ouvrage de cours. Que le lecteur ne se décourage surtout pas s'il ne comprend pas tout ce chapitre en première lecture. Il pourra toujours revenir sur la difficulté une fois résolus les exercices se rapportant aux concepts incompris. Ne pas tout comprendre du premier coup est, somme toute, « normal ».

Contenu du chapitre

1 - Place de l'algorithmique dans le développement logiciel

Dans ce paragraphe, nous abordons le développement du logiciel en général puis celui d'un programme en particulier. Nous évoquons également les différents acteurs intervenant durant ce développement. Nous terminons en présentant les notions de *validation* et de *vérification* à l'aide de *jeux d'essais*.

1.1 - Développement du logiciel en général

Un logiciel peut être défini comme un ensemble de programmes qui contribuent à résoudre un problème. Développer un tel ensemble est un processus qui peut être long, qui doit être minutieux et qui se déroule en plusieurs étapes :

• Il y a d'abord l'*analyse des besoins*, phase durant laquelle un analyste va rencontrer le demandeur, dialoguer avec lui et décrire le plus précisément possible ce qui est attendu. On est dans l'espace du problème et on répond à la question « Quoi ? ». Cette étape se termine par la fourniture d'un *cahier des charges* (et la signature d'un contrat). Des *jeux d'essais de validation* sont également prévus.

• Il y a ensuite la *conception* d'une solution, phase durant laquelle celle-ci est construite par un analyste (à partir du cahier des charges) en même temps qu'il met en place des outils informatiques. On est dans l'espace de la solution et on apporte une réponse à la question « Comment ? ». Cette étape se termine par la fourniture d'un *document d'analyse*, des *algorithmes* et des *jeux d'essais de vérification*.

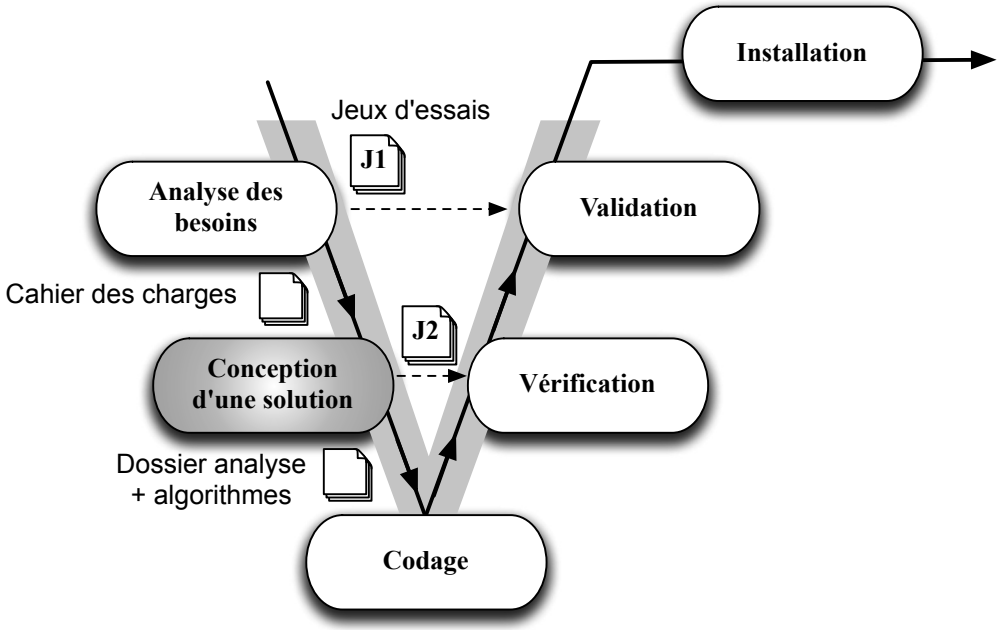
• Il y a ensuite la phase de *codage* pendant laquelle les algorithmes sont traduits sous forme de programmes, ce travail étant réalisé par les programmeurs qui ne se posent, en principe, qu'une seule question : « Comment traduire l'algorithme ? ».

• Il y a ensuite les tests de *vérification*, phase durant laquelle chaque programme va être contrôlé, séparément tout d'abord puis dans son interaction avec les autres, de façon à éliminer toutes les erreurs de conception. On est ici toujours dans l'espace de la solution et on répond à la question « Est-ce bien écrit ? ».

• Ensuite, viennent les tests de *validation*, pendant lesquels on vérifie que le logiciel produit est bien celui qu'attendait l'utilisateur. La question à laquelle il faut répondre est « Est-ce bien cela qu'il fallait faire ? ».

• Il y a enfin l'*installation* du logiciel sur les ordinateurs du demandeur et son exploitation par les utilisateurs.

Cette succession d'étapes (nous l'avons un peu simplifiée) constitue ce que l'on appelle le *cycle de vie* d'un logiciel. Il existe plusieurs représentations de ce cycle, une des plus connues étant ce que l'on appelle le *cycle en V* :



1.2 - Développement d'un programme

Les étapes de création d'un logiciel (un ensemble de programmes) se retrouvent dans la création d'un programme isolé qui doit résoudre un problème ponctuel. Les entrées, contraintes et résultats attendus du problème, nommées *spécifications*, sont consignées dans le cahier des charges. Une fois le problème analysé, la solution conçue doit être expliquée dans le dossier d'analyse et exprimée de manière à préparer le travail du programmeur : c'est là que va apparaître l'*algorithme*.

L'algorithme constitue, en quelque sorte, une description précise de ce que doit exécuter le programme. Il peut être considéré comme un « programme » destiné à une machine abstraite. Il est rédigé dans un langage qu'un ordinateur ne sait pas interpréter mais qui est conçu pour être compris par un être humain et se traduire facilement dans un langage de programmation.

Tout codage d'un programme doit être précédé de la conception de son algorithme.

Un programme est un ensemble d'instructions, exécutées dans un ordre chronologique précis. Il en est de même pour un algorithme.

Un même algorithme pourra être traduit dans différents langages de programmation : Java, C, Pascal...

Cette traduction est encore, pour quelque temps, effectuée à la main par les programmeurs.

1.3 - Acteurs concernés par le développement logiciel

Quels sont les acteurs qui interviennent durant le cycle de vie d'un logiciel ou dans celui d'un programme ? On peut les classer en cinq catégories :

Acteur	Fonction
le **demandeur :** • l'enseignant, • le client.	Est confronté à un problème qui peut être résolu par un ou plusieurs programmes. Contribue à la rédaction d'un document, nommé *cahier des charges*, qui définit ses besoins sous forme de *spécifications* (quelles données, quels résultats, quelles propriétés...) et qui précise les conditions d'utilisation que devront respecter les futurs utilisateurs.
l'**analyste :** • l'apprenant, • l'analyste.	Prend connaissance du cahier des charges fourni par le demandeur et rédige : • un *document d'analyse* expliquant la solution mise en œuvre et les choix retenus lorsqu'il y a plusieurs solutions, • les *algorithmes* mettant en œuvre la solution choisie, • des *jeux d'essais de vérification* qui montrent que les solutions proposées répondent correctement au problème posé.
le **programmeur** : • l'apprenant, • le programmeur.	Transcrit les algorithmes fournis par l'analyste en *programmes* exécutables[1]. Vérifie ces programmes avec les jeux d'essais fournis par l'analyste.
le **testeur** : • l'apprenant, • l'équipe de test, • l'enseignant.	Rédige des *jeux d'essais de validation* à partir des spécifications fournies par le demandeur et vérifie que les programmes correspondent à la demande.
l'**utilisateur** : • l'apprenant, • l'usager, • l'enseignant.	Dialogue avec le programme en lui fournissant les données nécessaires. Il est, en principe, compétent dans le domaine d'utilisation du programme mais il peut faire des erreurs dans les données qu'il fournit ou dans l'usage qu'il fait du programme. C'est pourquoi ses erreurs potentielles doivent être contrôlées le mieux possible (contrôle de réponse).

1 ou plus exactement interprétables ou compilables.

Remarque : nous reviendrons sur les différents jeux d'essais évoqués dans ce tableau, en particulier pour expliquer pourquoi il en existe plusieurs types et pour en justifier l'existence.

Selon la taille des équipes, il est possible pour une même personne de jouer plusieurs rôles. On pourra ainsi, dans une petite structure, avoir un analyste programmeur ou bien un programmeur testeur. Les conditions optimales de fonctionnement sont toutefois réunies lorsque chaque rôle est joué par des personnes différentes.

Au cours de son travail d'analyse et de programmation, l'apprenant est amené à jouer les quatre derniers rôles présentés dans le tableau. Il est primordial qu'il en prenne conscience. Si plusieurs apprenants ont la possibilité de travailler ensemble, ils auront tout intérêt à « faire tourner » les rôles, par exemple en faisant en sorte que chacun donne son algorithme et son programme à tester par un autre.

1.4 - Validation et vérification à l'aide de jeux d'essais

La confiance dans les produits logiciels que l'on développe est une propriété essentielle pour leur acceptation. Celle-ci repose sur les deux questions suivantes : « le produit est-il valide ? » « le produit est-il correct ? »

La *validité* (décrétée après un processus de validation) est une préoccupation du demandeur du produit. Le produit sera valide s'il répond à l'attente du demandeur exprimée en même temps que l'expression des besoins.

La *correction* (décrétée après un processus de *vérification*) est une préoccupation des informaticiens. Le produit sera correct s'il est bien fait et s'il répond sans erreur aux spécifications du problème énoncées dans le cahier des charges.

La **validation** est un processus expérimental qui consiste à définir des jeux de données pertinents, élaborés à partir des spécifications et à les tester avec le programme terminé, vu comme une « boîte noire ». C'est en général le travail d'une personne (testeur) qui n'a pas participé au développement et qui, de ce fait, ne peut pas être influencé par la solution mise en œuvre dans le programme. Les couples (données / résultats) qui servent à ces tests sont les **jeux d'essais de validation**.

La **vérification** est effectuée sur la solution. Elle peut théoriquement prendre deux formes : preuve ou processus expérimental. Faire la *preuve* qu'un algorithme ou qu'un programme est « juste » est théoriquement possible en construisant une démonstration composée d'un enchaînement de propositions logiques et en définissant de façon précise ce que sont les *propriétés* que doit respecter l'algorithme ou le programme. C'est un domaine à part entière de

l'informatique. La preuve d'un algorithme très court peut être très longue et il est hors de question de rédiger des preuves dans le cadre de notre travail.

L'autre méthode est une démarche expérimentale. Elle met en rapport des séries de valeurs fournies à l'algorithme ou au programme et des séries de résultats qu'il devrait produire. Ce sont les **jeux d'essais de vérification**.

Il y a donc bien **deux** séries de valeurs :

- les jeux d'essais de validation, qui relèvent du problème et qui s'appliquent au programme terminé,
- les jeux d'essais de vérification, qui relèvent de la solution et qui s'appliquent à tout ou partie des algorithmes ou des programmes constituant cette solution.

Pour que la validation et la vérification soient probantes, **il ne faut pas tester avec des séries de valeurs prises au hasard mais choisir ces séries à la suite d'un raisonnement** prenant en compte :

- l'expression des besoins et les spécifications pour la validation,
- les spécifications et la structure des algorithmes pour la vérification.

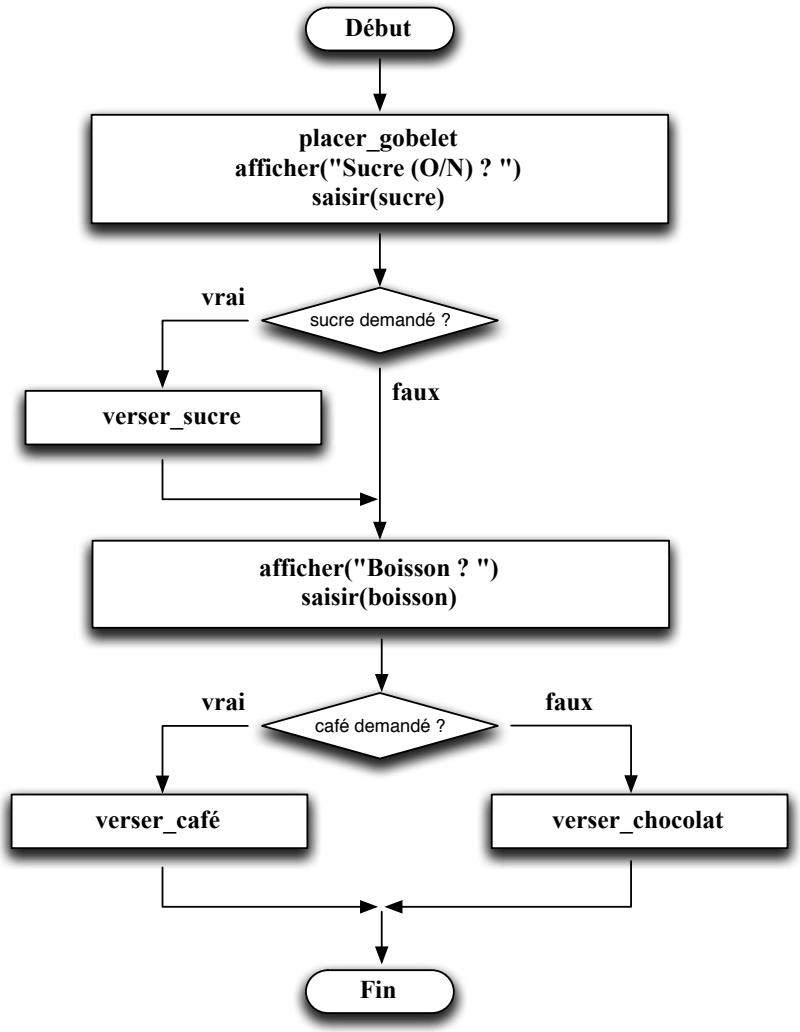
Prenons un exemple, simple, pour illustrer les différences entre ces deux « types » de jeux d'essais, celui d'un programme qui simule le comportement d'un distributeur gratuit de boissons chaudes. Cet appareil propose du café ou du chocolat, ces deux boissons pouvant être sucrées ou non sucrées. Nous ne prenons pas en compte les pannes.

Vu de l'extérieur, ce distributeur se présente sous la forme de trois boutons avec lesquels l'utilisateur indique son choix. Certaines combinaisons de boutons sont interdites (il n'est pas possible de sélectionner café ET chocolat). Sont acceptées les demandes suivantes : café non sucré, café sucré, chocolat non sucré, chocolat sucré.

Un **jeu d'essais de validation** contiendra donc des valeurs permettant de tester toutes ces combinaisons, sans avoir besoin de connaître la solution :

boisson ?	**sucre ?**	**boisson servie**
café	oui	café sucré
café	non	café non sucré
chocolat	oui	chocolat sucré
chocolat	non	chocolat non sucré

L'analyste qui a étudié ce problème va probablement produire un algorithme ressemblant à celui de la page suivante :



Ce graphe peut se tester de deux façons :

• Soit l'on se focalise sur les conditions et l'on envisage toutes les combinaisons de valeurs pour les demandes en sucre et en boisson. On calcule alors le nombre de chemins (il y en a quatre), éventuellement partiellement redondants et on propose une série de tests couvrant tous ces chemins (voir le schéma présenté dans la partie A, page suivante).
• Soit l'on se focalise sur les actions et l'on veut toutes les tester en essayant de ne pas le faire plusieurs fois. On calcule alors le nombre de chemins linéairement indépendants (il y en a deux) et on propose une série de tests couvrant ces chemins (voir le schéma présenté dans la partie B, deux pages plus loin).

A) chemins parcourant toutes les combinaisons de valeurs :