CONDUITE D'UN PROJET ET QUALITÉ [1]

(1ère partie)

Alain VAN SANTE

Pour devenir un bon programmeur, il faut généralement passer par trois stades :

- Le premier stade consiste à s'initier aux règles de base de l'algorithmique et à maîtriser les bases d'un langage de programmation. On y réalise de petits programmes destinés à fixer ses connaissances, sans véritable objectif de performance ou de réutilisabilité.

- Le second consiste à se perfectionner dans la maîtrise des structures de données les plus courantes et à apprendre à les reconnaître et les utiliser dans la conception de programmes de moyenne importance. On se familiarise avec une démarche et des outils quelquefois un peu surdimensionnés face aux problèmes à résoudre. On prend ici conscience des problèmes de performance et de réutilisabilité.

- Le troisième n'est abordé généralement que face à la réalité du terrain, lors de la réalisation d'un projet de bonne importance, qui nécessite d'allier connaissances de base, outils, maîtrise des principaux concepts algorithmiques, auxquels viennent s'ajouter gestion, planification et communication.

Pour que ce troisième stade soit abordé dans les meilleures conditions et ne soit pas vécu comme un calvaire par nos étudiants (notamment au cours de leur stage de deuxième année), je me suis efforcé de leur présenter quelques règles et conseils élémentaires, et quelques éléments de réflexion.

J'ai pour cela essayer de faire la synthèse de mes expériences en la matière, et de celles d'illustres spécialistes, que je remercie ici pour la qualité de leurs ouvrages :

---

1 Nous reproduisons cet article, paru dans le numéro 12 de la revue TRACE, Enseignement Technologique Tertiaire (CERTA-CRDP de Dijon) avec l'aimable autorisation de l'auteur et des responsables de la publication.

G. Pierra : Les bases de la programmation et du génie logiciel - Dunod

F. Parobeck, G. Bonno : La qualité logicielle. Concepts de base et mise en oeuvre - Dunod

J.P. Martin : La qualité des logiciels - AFNOR Gestion

B. Liskov et J. Guttag : La maîtrise du développement de logiciel - Editions d'Organisation

R. Ogor et R. Rannou : Langage ADA et algorithmique - HERMES Informatique

Le document qui suit se veut donc la clé d'une porte qui mènera nos étudiants vers les sommets de la conception orientée objets, pour peu qu'ils veuillent s'en donner la peine. Je le livre donc à votre sagacité, en y ajoutant (*en italique*) quelques réflexions plus générales.

QUALITÉ D'UN LOGICIEL

Il n'est pas envisageable de concevoir un logiciel sans être conscient des problèmes de qualité.

Mais qu'est-ce que la qualité ?

La définition la plus couramment citée est la suivante :

La qualité d'un logiciel est son aptitude à satisfaire les besoins des utilisateurs.

Ceci permet de rejeter certaines idées fausses.

La qualité d'un logiciel ne se juge pas à l'absence d'erreurs détectées (ou du moins pas seulement). Elle ne se juge pas non plus à la prolifération de ses fonctionnalités.

En cela, le mieux est souvent l'ennemi du bien : il n'est pas cohérent de penser un logiciel au superlatif : c'est comme si en matière d'automobile, on ne construisait que des Rolls ou des Ferrari !

La phase la plus importante est donc celle qui consiste à traduire (ou à préciser) les besoins des utilisateurs et à spécifier clairement le problème à résoudre. Elle doit permettre d'identifier les fonctions à réaliser, les conditions et les contraintes d'utilisation afin de réaliser un logiciel **fiable et sûr**. Elle doit également permettre d'évaluer les caractéristiques souhaitées du logiciel.

Les termes suivants représentent les caractéristiques essentielles (retenues par l'AFNOR) à évaluer dans la phase de spécification :

- **CONFIDENTIALITÉ** : aptitude d'un logiciel à être protégé contre tout accès par des personnes non autorisées
  - Contrôle et historique des accès
  - Protection du code et des données
- **COUPLABILITÉ** : aptitude à être couplé à un autre logiciel
  - Standardisation des données et des interfaces
  - Compatibilité avec des standards du marché
- **EFFICACITÉ** : aptitude à minimiser l'utilisation des ressources disponibles
  - Consommation minimale de mémoire
  - Vitesse et capacité optimales des périphériques
  - Consommation minimale de temps machine
- **MANIABILITÉ** : aptitude à être convivial et facile d'emploi pour l'utilisateur auquel il est destiné
  - Facilité de dialogue homme-machine
  - Facilité de mise en oeuvre et d'exploitation
  - Facilité d'apprentissage
  - Documentation et aide en ligne
- **ROBUSTESSE** : aptitude à conserver un comportement conforme aux besoins exprimés en présence d'événements non souhaités ou non prévus.
  - Exactitude des résultats obtenus
  - Tolérance aux incidents matériels
  - Tolérance aux fautes de l'utilisateur
- **MAINTENABILITÉ** : aptitude à faciliter la localisation et la correction des erreurs résiduelles
  - Lisibilité et auto-documentation
  - Modularité
  - Observabilité
  - Simplicité
  - Traçabilité

- **ADAPTABILITÉ** : aptitude à faciliter la suppression ou l'évolution des fonctionnalités existantes ou l'adjonction de nouvelles fonctionnalités
  - Expansibilité
  - Lisibilité
  - Modularité
  - Observabilité
  - Simplicité
  - Traçabilité
- **PORTABILITÉ** : aptitude à minimiser les répercussions d'un changement d'environnement matériel et logiciel
  - Absence de lien étroit avec l'environnement matériel ou logiciel

Toutes ces caractéristiques doivent donner lieu à une évaluation, en fonction des besoins de l'utilisateur.

La caractéristique suivante doit être une exigence optimale pour tout logiciel :

- **FIABILITÉ** : aptitude d'un logiciel à accomplir l'ensemble des fonctions spécifiées, dans un environnement opérationnel donné, pour une durée d'utilisation donnée.

Une autre caractéristique peut également être retenue, bien que ne concernant que les informaticiens :

- **RÉUTILISABILITÉ** : aptitude à pouvoir être réemployé en tout ou partie dans un autre cycle de production.

Elle est certainement de loin la plus difficile à obtenir, d'autant qu'elle impose généralement une perte de temps sur les premiers développements (on fixe à environ 40 à 60 % cette perte de temps), qu'elle pose des problèmes de gestion des modules réutilisables et de suivi des versions successives.

En tout état de cause, cette qualité n'en est donc peut être pas une pour les projets moyens. Car un module réutilisable n'a d'intérêt que s'il est réutilisé !

CONCEPTION D'UN PROJET

Le cycle de développement d'un projet regroupe différentes catégories d'activité :

- **les activités de production**
- **les activités de contrôle (technique et qualité)**
- **les activités de gestion**

Nous nous intéresserons bien sûr ici essentiellement à l'activité de production, mais nous allons rappeler rapidement le rôle des autres activités :

CONTRÔLE

CONTRÔLE TECHNIQUE

Le contrôle technique regroupe tous les tests et vérifications des programmes (tests unitaires, tests d'intégration et tests de qualification, boîte noire et boîte blanche), mais aussi les contrôles des documents produits (relecture, analyse critique, etc.).

Les tests représentent une partie importante dans le développement et ne doivent pas être improvisés. Leur préparation doit démarrer dès la phase de spécification, en tenant compte de toutes les contraintes du cahier des charges. Leur exécution doit suivre le cycle de vie (tests unitaires et tests d'intégration) et doit permettre de valider le produit final (tests de qualification).

La définition et la réalisation des tests représentent des tâches complexes pas toujours reconnues à leur juste valeur. De ce fait, elles restent le plus souvent artisanales, quand elles sont réalisées.

Les techniques suivantes sont les plus souvent employées et sont généralement suffisantes pour des développements moyens.

RÉALISATION DE JEUX D'ESSAI

Il s'agit de construire une série de valeurs représentatives de l'ensemble des cas à traiter par le logiciel ou la partie de logiciel à tester. On part pour cela des spécifications du problème (boîte noire) et de la structure de l'algorithme à tester (boîte blanche), qui est linéarisée.

L'exécution permet alors de contrôler la validité du module testé, ou, au besoin, de déterminer les causes de non-fonctionnement. On utilise pour cela des techniques d'analyse dynamique plus ou moins sophistiquées :

- Utilisation d'un utilitaire de mise au point

Un utilitaire de mise au point permet la pose de points d'arrêt (par exemple sur une ligne donnée ou sur la modification d'une variable donnée), la visualisation et la modification des variables (ou de la mémoire) au cours de l'exécution du programme, l'exécution pas à pas, etc.

- Intégration d'instructions de mise au point dans le programme

Cette intégration peut avoir pour but de remédier à l'absence d'utilitaire de mise au point : on visualise donc le contenu de variables à des endroits stratégiques du programme, en y intégrant des pauses à l'aide des instructions du langage.

Elle peut permettre d'obtenir une trace écrite de l'exécution d'un programme, en intégrant des ordres d'écriture dans un fichier texte permettant de suivre le chemin parcouru dans le code lors de l'exécution du programme.

ANALYSE STATIQUE

Si l'exécution des jeux d'essai est le "juge de paix" qui permet de déterminer la présence de défauts non détectés jusqu'à présent, il est souhaitable que leur nombre soit le moins élevé possible. Pour cela il est nécessaire de jalonner le développement de contrôles permettant de détecter les défauts au plus tôt afin de minimiser leur coût de correction. Différents outils peuvent être employés :

- Respect de normes d'écritures des programmes

Il s'agit de suivre des règles strictes pour l'écriture et la documentation des programmes : règles de construction du nom des variables et des modules (actions ou fonctions), passage des paramètres, places et pertinence des commentaires, structures autorisées, instructions interdites, etc.

- Construction de l'arbre de structure

Il s'agit de créer un arbre qui représente la structure de l'algorithme sur deux axes : l'axe du temps (de la gauche vers la droite) et celui de la complexité (du haut vers le bas). Cet arbre permet de visualiser rapidement la structure de l'algorithme, en se limitant aux actions importantes. *Des méthodes de programmation comme JSP (Jackson System Programming), LCP (Logique de construction de programmes), PSP (Programmation sans panne) peuvent aider à la construction de ces arbres.*

- Construction des graphes d'appel

Il s'agit de créer un graphe dans lequel chaque nœud représente une fonction ou une procédure du programme, et chaque lien une relation d'appel. On peut également associer à ces liens la liste des paramètres passés lors de l'appel, et organiser le graphe par niveau : au niveau le plus haut, on trouvera le programme principal, puis, à chaque niveau, les procédures ou fonctions appelées au niveau supérieur. Une procédure ou fonction apparaît toujours à un niveau inférieur à celles qui l'appellent (sauf dans le cas de fonctions ou de procédures récursives indirectes croisées, qui apparaissent au même niveau).

Ces différentes méthodes sont le plus souvent associées à des produits logiciels, comme l'**éditeur de références croisées** (qui permet de détecter les variables ou les portions de code non utilisées), ou l'**analyseur de programme** qui fournit le graphe d'appel. Ces outils (et bien d'autres) ne sont cependant utiles que dans le cadre d'une organisation sérieuse des procédures de test.

LE CONTRÔLE QUALITÉ

Le contrôle qualité consiste à juger de la prise en compte des objectifs fixés (documentation, formation des utilisateurs, mise en place). Il intervient dans toutes les phases du cycle de vie d'un programme et doit devenir une "philosophie de pensée".

Il doit faire l'objet d'une véritable politique de l'entreprise, qui doit se fixer des objectifs à travers le respect des normes en vigueur (normes ISO, normes AFNOR).

LA GESTION D'UN PROJET

La complexité croissante des logiciels nécessite des équipes et des temps de travail importants. Il est donc vital d'assurer le suivi et la cohérence des différentes tâches réalisées, ainsi que les liaisons entre elles.

C'est le rôle de la **gestion de projet** qui doit permettre de planifier toutes les ressources et de suivre l'évolution du projet pour affiner (ou réajuster) les estimations.

Une bonne gestion de projet repose sur la tenue d'une documentation permettant de suivre l'évolution du projet (planning prévisionnel, planning effectif, révisions, diagramme de GANTT, réseau PERT).

Puisqu'il est impossible d'éliminer les erreurs, tout doit être fait pour permettre de les repérer le plus vite possible, puis de les corriger. Toutes les corrections doivent être répertoriées afin qu'il soit possible de suivre l'évolution du produit : c'est le rôle de la **gestion des modifications**.

LE CYCLE DE DÉVELOPPEMENT

Toutes ces activités doivent être organisées dans le temps. On distingue deux types d'organisation :

LE CYCLE EN CASCADE

Toutes les étapes concernant le projet sont réalisées successivement :

Spécification du logiciel

C'est l'ensemble des activités consistant à définir de manière précise, complète et cohérente ce dont l'utilisateur a besoin. Elle permet d'obtenir un dossier de spécifications contenant :

- l'identification des fonctions à réaliser
  - données en entrée et en sortie
  - traitements effectués sur ces données
  - conditions de mise en œuvre et d'arrêt

  - traitement des exceptions
- les conditions d'exploitation et d'utilisation
  - matériels utilisés
  - logiciels utilisés
  - outils de développement disponibles
  - type de traitement (*batch* ou conversationnel)
- les performances souhaitées
- les critères de qualification du produit
- la criticité du logiciel (résistance aux pannes, MTBF, MTTR [2])

Conception préliminaire

C'est l'ensemble des activités conduisant à l'élaboration de l'architecture du logiciel. Elle permet de produire un dossier de conception préliminaire contenant :

- la description des fonctions à réaliser
- la définition des interfaces, des données et des composants du logiciel
- une justification éventuelle des choix réalisés
- un découpage du travail à réaliser

Conception détaillée

C'est l'ensemble des activités consistant à détailler les résultats de la conception préliminaire (algorithmes et structures de données) jusqu'à un niveau suffisant permettant le codage.

Il permet de produire le dossier de conception détaillée contenant :

- la description des fonctions réalisées
- les interfaces, contrôles et enchaînements
- la description des données et des structures de données

Codage

C'est l'activité qui consiste à traduire le résultat de la conception détaillée en programmes à l'aide d'un langage de programmation.

2 MTBF : mean time between failures (temps moyen de bon fonctionnement) ; MTTR : mean time to repair (durée moyenne de réparation).

Elle permet d'obtenir les programmes sources **autodocumentés**.

Test unitaires

C'est l'opération ayant pour but de vérifier le bon fonctionnement de chaque composant pris individuellement.

Intégration

C'est l'opération ayant pour but d'assembler progressivement tous les composants du logiciel.

Validation

C'est l'opération ayant pour but de s'assurer du bon fonctionnement du logiciel et de sa conformité avec les spécifications.

Une fois cette validation effectuée, le logiciel rentre dans sa phase terminale : la phase d'exploitation.

Exploitation

Cette phase regroupe la mise en œuvre opérationnelle du logiciel et son utilisation.

Elle comprend également la maintenance corrective (liée aux erreurs résiduelles) et la maintenance adaptative ou évolutive (liée à l'évolution des besoins).

Cette organisation est la plus classique et s'adapte parfaitement à des projets de moyenne ou grande importance.

Mais, dans certains cas, il peut être intéressant de disposer d'une version restreinte permettant de vérifier certaines hypothèses, de préciser certains objectifs ou plus simplement de faire plaisir à un client impatient.

Dans ce cas on adoptera plutôt un CYCLE INCREMENTAL.

CYCLE INCRÉMENTAL

Un cycle incrémental peut se décrire comme une succession de cycles en cascade fonctionnant parallèlement.

Chaque phase qui se termine fournit les résultats qui permettent :

- le démarrage de la phase suivante dans le même cycle,
- le démarrage de la même phase dans le cycle suivant.

Cette méthode augmente les temps de développement, mais permet de disposer rapidement de préversions simplifiées qui peuvent jouer le rôle de prototype. *Elle apparaît dans la littérature actuelle sous différentes formes, comme le modèle en spirale de Boehm ou les cycles en V ou en W de Golberg (cf. La Méthode Merise Tome 3 : Gamme opératoire de A. Rochfeld et J. Moréjon aux Editions d'Organisation).*

PRODUCTION

Lorsqu'un projet devient important (plusieurs dizaines de milliers de lignes de sources), une démarche modulaire s'impose naturellement.

Son objectif est de décomposer le problème en sous-problèmes donnant lieu à la création de modules indépendants interagissant entre eux de manière simple et clairement définie, et satisfaisant les spécifications une fois intégrés.

Cette décomposition doit permettre :

- le travail de plusieurs équipes avec un minimum d'interaction entre elles,
- l'amélioration de la maintenabilité du logiciel réalisé,
- la minimisation des efforts

Malheureusement si tout le monde est d'accord pour affirmer l'intérêt d'une telle décomposition, personne ne peut véritablement répondre à la question : Comment faut-il décomposer ?

Nous allons donc essayer de fournir quelques conseils pour aboutir à une décomposition raisonnable.

Suite dans le prochain numéro.

Alain VAN SANTE
Lycée Gaston Berger - Lille