Une démarche pour enseigner la construction des logiciels

Jacques FLECK
IUT Strasbourg Sud, France

A une époque où l'on constate à la fois une recherche croissante de la qualité des logiciels, un raccourcissement des délais de réalisation, une concurrence effrénée pour la réduction des coûts [GRE 86] et une augmentation de la diversité des applications et des outils professionnels, on peut raisonnablement se demander si l'informaticien n'est pas confronté à une sorte de contradiction, rendant sa mission extrêmement difficile, voire impossible. Ce sentiment interpelle tout naturellement les formateurs que nous sommes : Que faut-il enseigner aux futurs informaticiens ? Quels moyens devons nous utiliser pour leur apprendre à concevoir et à réaliser des logiciels qui répondent aux exigences de plus en plus nombreuses des clients ?

En nous appuyant sur une expérience d'enseignement de l'informatique de plus de vingt ans, conduite à la fois en formation initiale en IUT (Institut Universitaire de Technologie) et en formation continue au CNAM (Conservatoire Nationale des Arts et Métiers), nous proposons une réflexion générale sur la formation aux concepts de base de l'informatique ; nous illustrons notre propos par l'enseignement de l'algorithmique et celui des systèmes experts. Le choix de ces deux approches, apparemment très différentes, est délibéré, dans la mesure où il nous permet de démontrer que les éléments méthodologiques fondamentaux sont identiques dans les deux cas et que la différenciation provient essentiellement des outils utilisés dans la mise en oeuvre.

1 "Bien définir ce que l'on veut faire ... "

S'il est vrai, comme l'écrit Niklaus Wirth dans l'avant-propos de son ouvrage [WIRTH 86], que la programmation est à la fois un art et une technique, il n'en demeure pas moins exact qu'il convient de fixer certaines règles sans pour autant bloquer les aspects "artistiques" de la créativité.

Le manque de compréhension d'un problème posé est sans doute une des principales sources d'erreurs : la réalisation débute très souvent avant que son auteur ait réellement compris l'objet du problème à traiter, quand elle ne se substitue pas purement et simplement à cette réflexion préliminaire ! Il est unanimement reconnu qu'il est très dangereux de "raisonner dans un langage de programmation", quel qu'il soit ! Le but d'un enseignement sur la construction des logiciels n'est pas d'enseigner des dispositifs et des particularités d'un langage ou d'un outil [WIRTH 86] ; il doit viser l'apprentissage d'une démarche qu'il sera possible de transposer dans des environnements divers et variés.

Le premier effort doit donc consister à expliciter le problème sous forme d'un énoncé cohérent [FLE 93] ; seule une étude préalable sérieuse et approfondie permettra de lever les ambiguïtés, de dégager les objectifs principaux [ARSAC 82] et si possible de trouver une description de calcul associant aux données les résultats correspondants [ROG 90]. Il est en effet insensé et prématuré de vouloir traduire dans un formalisme informatique, forcément contraignant, une solution incohérente ou insuffisamment spécifiée. Certes, il est parfois difficile de tout connaître par avance; une connaissance parfaite des résultats attendus ou *objectifs primaires* constitue le minimum à exiger.

La spécification d'un objectif débute par le choix d'un identifiant et la donnée d'un texte bref, en langue naturelle, permettant de décrire l'objet ou l'entité qui lui correspond [SOMM 88, FRO 90]; cette description constitue à la fois une aide au développement qui suivra et l'amorce de la documentation du logiciel. Dans la mesure du possible, il est souhaitable de préciser, dès cet instant, la nature de l'objectif par l'énoncé d'un type ; cette information servira à réaliser des contrôles de cohérence tout au long du développement.

2 "Formaliser la définition des objectifs ..."

La rédaction de commentaires en langue naturelle évite, dans un premier temps, des diversions dues aux problèmes de syntaxe et de mise en oeuvre des outils informatiques ; une part d'imprécision demeure toutefois inhérente à tout énoncé rédigé en langue naturelle ; il convient de la supprimer en complétant cette description par une expression formelle de cet énoncé.
Il existe des langages formels de haut niveau mais leur diffusion est encore trop restreinte, par rapport aux pratiques des entreprises, pour construire sur eux toute une formation. Il est néanmoins possible, dans des cycles technologiques, tels que ceux des IUT ou des CEGEP, ou dans des cycles d'initiation en général, de définir un ensemble réduit de formes permettant d'obtenir des résultats satisfaisants. Ce dispositif garantit la cohérence de l'ensemble des spécifications dans la mesure où il opère entité par entité et que les définitions sont statiques : n'oublions pas que la spécification décrit ce que le logiciel fait sans détailler comment il le fait [FRO 90] ; la limitation des formes garantit de fait l'homogénéité du développement tout en laissant une liberté suffisante dans le choix des formes.

3 "Spécifier un algorithme ... "

Spécifier un algorithme, c'est établir la liste des spécifications relatives à tous les objectifs primaires et secondaires du problème posé ; la spécification d'un objectif s'inspire de la méthode MEDEE [PAIR 79, DUCRIN 84], mise au point par l'école nancéenne ; elle se présente sous la forme d'un quadruplet composé :

- d'un identifiant,
- d'un énoncé en langue naturelle,
- d'un énoncé de type,
- d'une forme de calcul de la valeur de l'entité associée à l'objectif [WIRTH 86].

Nous avons retenu sept formes de calcul pour spécifier les algorithmes ; elles nous permettent d'exprimer tous les résultats attendus, qu'ils soient finaux ou intermédiaires. Elles s'appuient sur les travaux des écoles grenobloise [COU 74, COU 87] et niçoise [BIONDI 87] : il s'agit des formes traduisant l'introduction d'une donnée élémentaire (lecture), l'expression d'un calcul simple, une expression conditionnelle simple, une expression conditionnelle multiple, une suite avec comptage, une suite avec condition de fin, l'appel d'une procédure. Une vérification immédiate de cohérence est effectuée en rapprochant la forme de calcul complétée avec le type associé à l'objectif.

L'application d'une analyse descendante ou d'une analyse procédant par affinements successifs [WIRTH 71, WARN 75, ARSAC 91] constitue un guide facultatif pour découvrir tous les *objectifs secondaires* en partant des objectifs primaires ; la spécification complète des objectifs primaires met en effet en évidence les besoins en *objectifs secondaires* ou résultats intermédiaires. Ce processus est récurrent et se répète ensuite de niveau en niveau, lors de la formalisation des objectifs secondaires eux-mêmes.

Cette étape de spécification peut sembler à juste titre fastidieuse ; son efficacité est améliorée par l'utilisation d'un logiciel d'aide à la spécification tel que TIL2 [FLE 93, HUBER 88] ; TIL2 intègre un éditeur guidé par la syntaxe qui facilite l'écriture des formes de calcul ; l'introduction de contrôles systématiques en cours de saisie accroît la fiabilité et la cohérence d'un tel ensemble de spécifications ; ces contrôles portent notamment sur :

- l'état de complétude des spécifications,
- la cohérence entre le type d'un objectif et celui de l'expression de sa valeur,
- les homonymies,
- etc ...

Cette étape est terminée lorsque tous les objectifs nécessaires sont complètement spécifiés. Nous signalons l'absence de toutes contraintes quant à l'ordre d'introduction des spécifications. Un exemple de spécification d'algorithme est donné en annexe.

4 "Traduire un ensemble de spécifications en programme ..."

Il est intéressant de remarquer que cette étape ne nécessite plus aucune prise de décision ; il suffit en effet d'associer un schéma de programme à chacune des formes de calcul utilisées dans les spécifications et de les interclasser ; ce problème de traduction et d'interclassement peut être automatisé : c'est ce que nous avons prévu dans TIL2. L'usager récupère en sortie un algorithme présenté de manière classique, comme le montre l'exemple figurant en annexe.

Cet algorithme, qui est en fait un modèle de programme, peut à son tour être implanté dans un langage de programmation, soit manuellement, soit par l'intermédiaire d'un générateur de code. Cette phase est facultative dans TIL2 qui est capable d'interpréter directement les spécifications à des fins de test.

Le logiciel ne fournit en réalité que deux vues des spécifications à l'usager : l'une est sous forme d'un algorithme, l'autre est sous forme d'un programme exécutable ; *les spécifications initiales demeurent les seules informations modifiables par l'usager*. On évite ainsi les

corrections intempestives de programme qui conduisent à des résultats non conformes aux spécifications.

L'utilisation d'un logiciel comme TIL2 focalise toute l'attention de l'usager sur l'élaboration des spécifications qui demeure la phase cruciale de tout développement ; les avantages pédagogiques d'un tel logiciel sont évidents : l'usager est libre du choix des exemples traités : aucune solution n'est prédéfinie et toute solution imaginée par l'usager est personnalisée ; l'usager est guidé durant son travail mais il conserve un degré de liberté suffisant ; les contraintes imposées par le logiciel en termes de cohérence l'aident à acquérir une méthode de travail rigoureuse.

5 "Construire une base de connaissances d'un système expert ..."

Même si les systèmes experts n'ont pas connu l'essor espéré, nul ne peut aujourd'hui les ignorer ; ils font partie intégrante des programmes de formation des IUT et des CEGEP [PAQ 91].

Une base de connaissances est un ensemble de règles qui, à l'instar d'un algorithme, exprime des traitements sur des entités ; on y retrouve les notions d'objectif primaire et d'objectif secondaire ainsi que la plupart des attributs qui s'y réfèrent : un identifiant, une description en langue naturelle, un type. Seule l'expression de la valeur de l'entité doit être modifiée ; une forme de calcul unique, permettant d'associer à chaque entité une liste de couples {valeur de référence , liste de conjonctures} est suffisante. Une conjoncture étant une conjonction de faits, une expression de la forme $X = \{V, <C_1, C_2, ... C_n>\}$ s'interprète comme suit : "la valeur V est associée à l'entité X si l'une des conjonctures $C_1, C_2, ... C_n$ est vérifiée".

La démarche de spécification d'une telle base s'apparente complètement à celle de la spécification d'un algorithme.

Une fois encore, le recours à un logiciel d'aide à la spécification améliorera la qualité des bases de connaissances construites du point de vue de leur cohérence et de leur complétude ; c'est pour cette raison que nous avons conçu le logiciel SAPIENS2 [FLE 87, THI 88, SOH 89] dont l'architecture est similaire à celle de TIL2.

6 Conclusion

Notre recherche d'éléments méthodologiques fondamentaux communs, servant à construire des logiciels, a mis en évidence deux parties distinctes, qui sont toujours présentes, même lorsqu'on réalise des programmes simples : les spécifications d'une part, l'implantation de ces spécifications d'autre part. La démarche qui débouche sur l'élaboration des spécifications est relativement standardisée ; elle s'appuie sur deux aspects : l'identification et la description de tous les objectifs, la formalisation de ces objectifs par le typage et les formes de calcul. Durant cette phase, l'usager doit concentrer toute son attention sur la compréhension et la

mise en forme des objectifs attendus ; il n'a pas à réfléchir aux moyens à mettre en oeuvre pour les atteindre. L'automatisation de certaines tâches intervenant pendant cette étape essentielle permet d'améliorer substantiellement la qualité des résultats tout en apportant une aide significative au niveau de la formation de l'usager.

Notre étude a montré que l'implantation pouvait se déduire de manière quasi automatique des
spécifications. Le cycle de développement se résume ainsi en trois étapes :

a. Elaboration des spécifications,
b. Implantation manuelle ou automatique des spécifications,
c. Tests,
.... au besoin retour en a).

Nous n'insisterons pas davantage sur les apports de ces logiciels ; signalons toutefois la contribution essentielle des contrôles de cohérence effectués en permanence sur les spécifications : ces vérifications, *faites à priori*, évitent la propagation d'erreurs grossières et apportent de ce fait des gains de temps appréciables, contrairement aux tests qui se font uniquement à postériori. Cela signifie qu'on ne s'engage dans l'implantation qu'après avoir validé les spécifications : en ce sens, cette démarche impose un changement important de mentalité car trop de développeurs se contentent encore de tester le produit final !

Nous tenons également à mentionner l'aspect incrémental et générique de cette démarche : il est en effet aisé de compléter ou de remplacer les formes de calcul ; c'est ainsi que nous avons procédé pour implanter dans TIL2 des fonctionnalités supplémentaires permettant à l'usager de définir ses propres types de données et de se constituer progressivement un environnement de plus haut niveau. La démarche que nous avons initialisée dans des contextes élémentaires peut se généraliser dans des environnements plus complexes, voire dans d'autres secteurs de l'enseignement de l'informatique.

Références bibliographiques

[ARSAC 82] "Premières leçons de programmation" : J.Arsac
CEDIC / Fernand Nathan (1982)

[ARSAC 91] "Préceptes pour programmer" : J.Arsac
Dunod (1991)

[BIONDI 87] "Introduction à la programmation"
Tome 1 : Algorithmique et langages : J.Biondi et G.Clavel
Masson (Editions de 1984 et de 1987)

[COU 74] "Introduction à l'algorithmique et aux structures de données"
J. Courtin et J. Voiron
IUT B de Grenoble - Département Informatique (1974)

[COU 87] "Initiation à l'algorithmique et aux structures de données"

Tome 1 : "Programmation structurée et structures de données élémentaires" :
J. Courtin et I. Kowarski - *Dunod Informatique (1987)*

[DUCRIN 84] "Programmation" Tomes 1 et 2 : A. Ducrin
Dunod Informatique (1984)

[FLE 87] "SAPIENS, un logiciel pour l'enseignement des systèmes experts" : J. Fleck
Séminaire des Départements d'Informatique d'IUT - Villetanneuse (1987)

[FLE 93] "Une méthode et des logiciels pour l'apprentissage de la programmation et des systèmes experts" : J. Fleck
Thèse de doctorat - Université René Descartes Paris V (1993)

[FRO 90] "Types de données et algorithmes" : C. Froidevaux,
M.C. Gaudel, M. Soria - *McGRAW-HILL (1990)*

[GRE 86] "Informatique - Programmation" Tome 1 : "La programmation structurée" :
Grégoire - *C.N.A.M. Cycle A - Masson (1986)*

[HUBER 88] "TIL - Un logiciel d'aide à la conception d'algorithmes" : M. et F. Huber, J. Fleck - *Rapport de projet CNAM - Strasbourg (1988)*

[PAIR 79] "La construction des programmes" : C. Pair - *Revue RAIRO Informatique n°13 (1979)*

[PAQ 91] "Systèmes à base de connaissances" : G. Paquette, L. Roy
Editions Beauchemin Ltée - Québec (1991)

[ROG 90] "Acquisition of Programming Knowledge and Skills" :
J. Rogalski et R. Samurçay - Psychology of Programming - *Academic Press (1990)*

[SOH 89] "PRESAPIENS, un logiciel d'aide à la cration de bases de connaissances" : I. Sohn, P. Dupouy, J. Fleck - *Rapport de projet CNAM (1989)*

[SOMM 88] "Le génie logiciel et ses applications" : I.Sommerville
Traduction par B.Dion - InterEditions (1988)

[THI 88] "SAPIENS, un logiciel pour l'apprentissage des systèmes experts" : G. Thibout, J. Fleck - *Rapport de projet CNAM (1988)*

[WARN 75] "Les procédures de traitement et leurs données" - Précis de logique informatique J. D. Warnier - *Les Editions d'Organisation (1975)*

[WIRTH 71] "Program development by stepwise refinement" : N. Wirth
Communications de l'ACM, Vol. 11 n° 4 (1971)

[WIRTH 86] "Introduction à la programmation systématique" : N. Wirth
Traduction par O. Lecarme - Masson (1986)

Annexe : Exemple de spécification d'un algorithme

Calculer la valeur de e^x avec une précision e en utilisant le développement limité

$$e^x = 1 + x/1 + x^2/2! + x^3/3! + ... + x^r/r!$$

La solution nécessite la spécification de trois suites

1. EXP
. réel
. Suite des résultats intermédiaires du développement
. forme itérative par arrêt sur condition (forme n°6)
propriété pour le calcul des indices : @T > e
expression du $i^{ème}$ terme : @EXP + @T (forme n°2)
expression du 1^{er} terme : 1 (forme n°2)

2. T
. réel
. Suite des termes du développement
. forme itérative par arrêt sur condition (forme n°6)
propriété pour le calcul des indices : "comme EXP"
*expression du $i^{ème}$ terme : @S * x / r (forme n°2)*
expression du 1er terme : 1 (forme n°2)

3. r
. entier
. Suite des rangs des termes du développement
. forme itérative par arrêt sur condition (forme n°6)
propriété pour le calcul des indices : "comme EXP"
expression du $i^{ème}$ terme : @r + 1 (forme n°2)
expression du 1er terme : 0 (forme n°2)

Par ailleurs, x et e sont des réels déterminés par lecture.

L'algorithme du calcul de e^x

```
Exponentielle :
    e = donnée
    x = donnée
    #EXP = 1
    résultat = #EXP
    #T= 1
    #r = 0
    %EXP, %T, %r = Tant que @T > e
    Répéter      EXP = @EXP + @T
                 résultat = EXP
                 r = @r + 1
                 T = @T * x / r
    Fin répéter
```