ADA et le génie logiciel dans les I.U.T.

Daniel Feneuille
I.U.T. département informatique
avenue Gaston Berger
13625 Aix-en-Provence Cedex 1
France
42 93 90 43
fax 42 93 90 74

Résumé. Les départements informatiques des I.U.T. forment en France des analystes/programmeurs en deux années après le baccalauréat. L'enseignement de la discipline informatique représente presque 900 heures (cours, TD, TP) et doit être, sans cesse, en "veille technologique", conséquence de l'évolution des méthodes, des matériels et des logiciels. Le langage ADA, choisi depuis quelques années par certains départements, permet de résoudre de nombreux problèmes didactiques en fédérant des enseignements modulaires, en permettant l'assimilation de concepts réputés difficiles mais incontournables en 1994 comme généricité, exceptions, conception orientée objet, processus parallèles..... On montrera que ADA allie les avantages pédagogiques de Pascal avec la rigueur et le professionnalisme des langages dédiés au génie logiciel.

1. Introduction

L'enseignement de l'informatique dans les départements informatiques des **I.U.T.** (**I**nstitut **U**niversitaire de **T**echnologie) a déjà été le thème de quelques exposés dans des colloques traitant de la didactique de l'informatique; des thèmes concernant le génie logiciel sont bien évoqués dans ce colloque-ci de l'A.F.D.I. mais c'est la première fois, à notre connaissance, que l'on y traitera du langage ADA. Cependant, comme on va le voir, ces trois entités: "ADA", "génie logiciel" et "I.U.T." seront plutôt indissociables tout au long de cet article (comme elles l'étaient dans le titre proposé).

On date, approximativement, les premiers enseignements de ADA dans les I.U.T. vers 1988, alors que le langage lui-même naît officiellement en 1983 (parution de la norme). En 1994, c'est presque la moitié des 36 départements qui ont intégré le langage dans leur cursus. Jusqu'à présent, l'enseignement de ADA était, en France, limité aux formations de plus haut niveau (presque toujours en école d'ingénieurs).

Pourquoi ADA, langage réputé complexe (donc ..... difficile à enseigner[1]), est-il proposé, aujourd'hui, dans des formations plus modestes (de Bac à Bac + 2) mais toujours à vocation très professionnalisée? C'est ce que nous allons montrer en présentant, en quelques pages: le problème spécifique des I.U.T., quelques concepts forts de génie logiciel bien illustrés avec ADA, en évoquant aussi le problème des entreprises, en illustrant des structures ADA (malheureusement pas toutes) et enfin en invitant à quelques réflexions pédagogiques qui permettront, nous l'espérons, de convaincre, voire de susciter des démarches analogues.

[1] mais ceci n'est pas vrai!

2. L'enseignement de l'informatique dans les I.U.T.

La vocation des I.U.T. est de former des techniciens supérieurs capables de réaliser des travaux informatiques concrets en étant rapidement opérationnels, efficaces et méthodiques. En bref, ils doivent être des **professionnels** pouvant au choix: dans une petite entreprise être l'analyste et le programmeur réunis, ou, dans un grand service, être capables de travailler en groupes et sous la conduite d'un chef de projet sur des travaux susceptibles de demander jusqu'à des millions de lignes de code. Il y a, dans ces situations extrêmes, un énorme défi pédagogique qui dépasse, et de loin, la traditionnelle initiation à l'informatique souvent bien résolue, d'ailleurs, avec du Pascal. Une des premières nécessités didactiques est de faire acquérir à nos étudiants, et conjointement, **savoir-faire et méthodes**.

Un programme pédagogique national (P.P.N.), révisé à intervalles réguliers, donne aux enseignants des I.U.T. de solides lignes de conduite: découpage de la matière et concepts, recommandations, etc ..... Mais fort heureusement le P.P.N. ne conclut pas sur LE langage à enseigner, car un langage de programmation n'est pas une fin en soi. En revanche, un bon langage devrait permettre de faire assimiler, de façon concrète et élégante des concepts fondamentaux qui résisteront au moins à une décennie de professionnalisme. Les lignes novatrices de la dernière révision du P.P.N. concernent surtout les concepts de **génie informatique et d'atelier logiciel**, de réseaux, et enfin **d'approche objet**. Ces révisions du P.P.N. ont pour effet de provoquer certains enseignants et les incitent à remettre en cause un enseignement bien rodé au profit d'un autre, même si celui-ci sent encore un peu la "peinture fraîche pédagogique"[2]. C'est ce qui nous est arrivé avec ADA.

L'enseignement (rien que pour l'informatique) occupe, à l'I.U.T. un volume d'heures très conséquent (et qui en ferait rêver plus d'un!): environ 900 heures (15 heures par semaine à raison de 34 + 24 semaines en deux ans, soit exactement 870 heures). Cette discipline y est donc, par la force des choses, découpée en thèmes puis en *modules*: algorithmique, tris, récursivité, structures de données, systèmes d'exploitation, fichiers, arithmétique, parallélisme.... pour ne citer que ceux qui nous intéressent[3]. Des enseignants "se spécialisent" dans quelques *modules* et y organisent les concepts à illustrer en s'appuyant sur un langage (pas toujours le même mais surtout PASCAL, C et COBOL). Les interfaces didactiques entre les *modules* sont souvent gérées "au mieux" et il n'est pas rare d'y trouver des redondances voire des contradictions! Qu'il fut agréable, enfin, de découvrir un langage véhiculant (et parfois de façon très riche!) beaucoup de concepts fondamentaux. ADA permet la couverture d'un large spectre de problèmes sans se disperser et en allant au fond. Reconnaissons tout de même que le nombre d'heures disponibles indiqué plus haut est consommé en partie en T.P. ce qui facilite bien les choses. ADA, fédérateur de modules d'enseignement permet ainsi de couvrir et d'encadrer un bon tiers du volume horaire d'informatique. La nouvelle norme (ADA9X, disponible sous peu: fin 1994) permettra encore d'augmenter cette couverture (l'héritage et la liaison dynamique par exemple fondamentaux aujourd'hui pour la P.O.O.; ou aussi la synchronisation par les données, ou encore l'interfaçage SQL).

---

[2] "remettre sur le métier sans cesse son ouvrage"!

[3] notons encore Réseaux, Analyse et Conception des Systèmes d'Information, Base de Données, I.A. .....

Il en résulte la disparition, naturelle, des langages redondants. En tout premier lieu la suppression de PASCAL s'impose puisque ADA c'est du PASCAL+[4].

Les griefs, forts, que l'on peut faire actuellement à PASCAL, sont connus[5]:

impossibilité de séparer interfaces et implémentations, (alors qu'ADA propose spécifications et réalisations en compilation séparée avec les paquetages)
difficulté à concevoir et mettre en oeuvre des traitements d'erreurs satisfaisants, (alors qu'ADA propose les exceptions)
sensation de toujours recommencer le même code à l'infini, (alors qu'ADA propose la généricité)
absence de surcharge, (naturelle en ADA)
pas de masquage d'information, (alors qu'ADA propose le private)
typage trop faible empêchant des validations "naturelles", (typage très fort en ADA)
"bogues" découverts trop tard dans la mise au point, (souvent découverts à la compilation en ADA)
pas d'entrée-sortie de type énumératif, (sans problème avec les entrées-sorties génériques de ADA)
pas de notion de processus, (tâches et rendez-vous ADA)
aucune normalisation, (en ADA la norme a précédé la parution des compilateurs du langage)
pas de portabilité, (en ADA on passe indifféremment du PC au gros système sous UNIX)
type article limité, (en ADA article avec discriminant)
pas de type tableau avec le type indice non contraint, (array (.. is <>) en ADA)
peu d'attributs de types (fondamentaux en ADA).

Comment alors ne pas quitter PASCAL, et même dès l'initiation, quand on découvre son successeur naturel! Peut-on aussi faire disparaître de nos enseignements COBOL et C? La réponse est, nous semble-t-il assez simple pour le premier nommé. Dans la profession, COBOL résiste à cause de son âge (on verra plus loin le paragraphe n°4 "ADA et les entreprises" à propos du projet Stanfins-R: "système de gestion de comptabilité et de finance du DOD") et il n'est pas rare de trouver encore et toujours des embauches COBOL pour nos étudiants. Mais, si c'est la seule raison pour en pérenniser l'enseignement le jeu n'en vaut pas la chandelle car les concepts véhiculés sont archaïques et sclérosants et nous avons vérifié que son apprentissage, en autodidacte, est rapide et sans problème pour des étudiants cultivés d'informatique[6]. La résistance ne viendrait-elle pas du côté de certains enseignants (sous couvert d'expérience) aussi conservateurs que certains programmeurs de la profession?[7] Bien que le langage C soit un piètre langage pour l'acquisition de saines pratiques méthodologiques il nous parait être le langage incontournable à connaître (ne fût-ce qu'à cause de sa connexité avec UNIX tout aussi incontournable). Aussi, s'il convient de compléter la formation des D.U.T. avec C (couplé à UNIX s'entend), il est intéressant de tenter tout de suite l'enseignement de C++ sans un palier intermédiaire (c'est-à-dire le C). Il

[4] et même du PASCAL ++ et ... encore plus!
[5] certes, le Pascal-objet de Borland a réparé certaines lacunes signalées ici.
[6] la bonne informatique bien sûr!
[7] l'alibi "expérience" n'est que la réponse d'hier au problème d'aujourd'hui!

n'en reste pas moins que c'est une mission périlleuse en phase d'initiation (c'est-à-dire dans l'enseignement des rudiments de base[8]). L'acquisition, grâce à ADA en amont, des mécanismes fondamentaux d'encapsulation, d'abstraction, de généricité, de composition modulaire et d'héritage (même si c'est limité en ADA83!) permet une assimilation sans douleur des concepts objets jugés traditionnellement difficiles comme: classes, polymorphisme, ligature dynamique.... Nous l'avons vérifié.

3. Le génie logiciel et ADA

Le thème de génie logiciel[9] est présent dans tout le programme pédagogique (P.P.N). Il est courant, dans certaines formations, de restreindre l'enseignement de l'informatique à C++, à cause des indispensables objets (l'un des composants de la construction logicielle). Mais cette approche, nous l'avons déjà noté, oblige à démarrer les apprentissages premiers avec trop de mauvaises habitudes conditionnant la vie entière de l'apprenti programmeur. Avec ADA (ancienne version) on s'intéresse à la conception orientée objet; l'absence d'héritage authentique ne permet pas de faire de l'objet-vrai! Dans cet esprit nous avons extrait des nombreux concepts à inculquer (tout au moins dans les premiers mois) quelques points forts en insistant, avant tout, sur la **qualité**. Des facteurs connus de la qualité du logiciel nous en avons privilégié trois: réutilisabilité, extensibilité et portabilité avec, au moins pour les deux premiers, leur corollaire: la production de **modules composables**.

Sans redévelopper ici ces concepts[10], mais pour concrétiser notre démarche, posons **NOS** définitions des trois facteurs qui ont retenu notre préoccupation:

La réutilisabilité: c'est l'aptitude d'un logiciel à être repris, en partie ou même en totalité, pour développer de nouvelles applications (sans le traditionnel "copier-coller puis remplacement" et aussi et surtout sans la nécessité de revalider).

L'extensibilité: c'est la facilité d'adaptation d'un logiciel aux changements de spécifications (impliquant modifications rapides et .... fiables!) permettant une maintenance adaptative sûre.

La portabilité: c'est la possibilité pour un produit logiciel de ne pas dépendre d'un environnement matériel particulier, notamment pour des applications numériques (mais recompilation obligée cependant!).

La production de modules composables (qui ne se réduit pas à la décomposition en modules d'un problème, technique chère à la programmation structurée) doit privilégier leurs combinaisons, leur lisibilité, leur continuité et leur robustesse. Ces modules, à l'instar de ce qui se propose pour le matériel, doivent être connus, identifiables, définis, diffusables.... pour être utilisés. En corollaire ces modules (nommés unités de compilation en ADA) sont compilables séparément et gérés par un environnement qui signalera les dépendances hiérarchiques des unités (notion de bibliothèque en ADA).

[8] par exemple, la maîtrise du réflexe: spécifier d'abord, réaliser ensuite.
[9] tout le monde a-t-il de ce concept la même définition?
[10] la place manquerait

Paquetage, surcharge, encapsulation et généricité (voir plus loin le paragraphe n°5 "Un aperçu de ....") présents dans ADA, résolvent en partie, mais pas trop mal, l'illustration des deux premières exigences. Quant à la portabilité des algorithmes numériques il est bien agréable de proposer au compilateur un contrat portant sur la précision minimale des nombres envisagés et le voir respecté sur toute machine[11].

Pour en terminer, et pour qu'il n'y aît pas d'ambiguïté avec le concept d'objet, notons qu'il est actuellement bien clair qu'avec ADA83 on ne peut pas parler de langage-objet mais d'approche objet (grâce, encore, aux paquetages réalisant des types abstraits); mais ce n'est déjà pas si mal. Et de toute façon .... bientôt ADA94 (alias ADA9X) qui, lui, est complètement objet.

4. ADA et les entreprises

ADA a été conçu à la suite d'un appel d'offre du DOD (département de la défense des U.S.A). Il s'agissait vers 1980 de proposer UN langage pour remplacer les quelques 500 langages[12] informatiques utilisés par cet organisme[13]. Actuellement c'est quelque deux millions de lignes par jour qui sont produites en ADA pour les applications de l'armée US[14]. Il est très rare qu'un autre langage, aujourd'hui, ait droit de cité au DOD et les dérogations sont soumises à une commission peu laxiste. Ailleurs, dans le monde et en France en particulier, ADA est bien utilisé en avionique et en informatique embarquée en général ainsi que pour le contrôle de trafic (aérien et ferroviaire). En fait, il est apprécié partout où le code à développer est conséquent (plus de 300.000 lignes) et où la fiabilité est cruciale. Mais il n'en reste pas moins qu'actuellement peu d'entreprises utilisent ADA pour leur développement notamment en informatique de gestion (COBOL, dans les années 1960, était aussi une émanation du DOD!). A cet égard le déroulement du projet "Stanfins-R" mérite d'être rappelé en quelques lignes ci-dessous et dépasse largement l'anecdote.

Le projet Stanfins consistait à reconcevoir puis réécrire le système de comptabilité de l'armée de terre des U.S.A. (système utilisé dans le monde entier). La société qui obtint le marché en 1986 commença pendant 6 mois en COBOL (langage que tout le monde maîtrisait et qui paraissait essentiel pour une telle application). Mais ADA était obligatoire à cette époque et il fallait une dérogation pour utiliser COBOL. Malgré des arguments solides[15] cette dérogation ne vint jamais[16]. L'application fut donc écrite en ADA dans les délais prévus[17] malgré le manque d'expérience des programmeurs (peu formés à ADA et au

[11] ou alors le voir refusé à la compilation (mais c'est aussi cela la portabilité!) et c'est possible avec ADA.
[12] et même parait-il 700 langages ou dialectes!
[13] le langage "vert" (futur ADA) devait être généraliste mais aussi hyper spécialisé (temps réel notamment)
[14] d'après le superviseur du développement ADA du DOD: W. Whitaker (cf "Dr Dobb's journal" Août 93)
[15] synonyme de "fallacieux" pour un ADA-tollah.
[16] l'essentiel de l'argumentation du rejet tient dans ces lignes extraites du rapport:" ADA est autant une technologie qui discipline le processus de développement du logiciel qu'un langage de programmation ..... les exigences poussées en fiabilité ... font apparaître le besoin d'utiliser la technologie ADA en tant que technologie orientée vers l'ingénierie".
[17] les détracteurs avaient parié qu'il n'en serait rien!

génie logiciel[18]) avec une productivité double de celle habituellement connue pour des situations analogues[19] (économie estimée à 24 millions de dollars).

La place manque, dans ce papier, pour apporter contradiction aux "justifications" les plus couramment évoquées par la profession pour ne pas franchir le pas ADA, énumérons les cependant mais succinctement:

> des raisons économiques: la formation à ADA et aux concepts qui vont avec coûterait cher; sans oublier le prix de la transition des programmes.
> des raisons psychologiques: réticence à quitter le "terrain" que l'on connaît bien pour d'autres "aventures" mal maîtrisées.
> des raisons politico/idéologiques: le DOD c'est l'armée US (imparable!).
> des raisons mythologiques: ADA c'est compliqué, contraignant, trop universel (imparable bis!)

En revanche, aux raisons qui devraient inciter les entreprises à utiliser ADA, évoquées tout au long de l'article ou sous-jacentes, on peut ajouter la normalisation (ISO, ANSI, AFNOR) garante d'une grande portabilité et d'une bonne continuité des applications (préoccupation majeure de l'industrie).

Enfin, pour clore cette partie et concernant le devenir professionnel de nos étudiants en ADA (en clair: les embauches) la situation nous a préoccupés. Nous avons même, un temps, désespéré de les voir travailler avec ce langage en entreprise. Mais les choses évoluent. Il faut dire que le "marché ADA" était réservé, traditionnellement, aux formations ingénieurs et il paraissait impossible (voire incongru) que de modestes techniciens supérieurs puissent maîtriser les concepts forts qui font l'informatique d'aujourd'hui. Et pourtant!

5. Un aperçu de ADA et quelques points forts (typage et composition de modules)

L'exposé succinct auquel nous nous livrons dans ce paragraphe ne prétend pas faire une présentation exhaustive du langage (c'est évident), ni même un survol, mais évoque quelques points intéressants.

Tout d'abord ADA propose les traditionnelles "structures algorithmiques":

bloc	`begin .... end;`
conditionnelle	`if ( ) then ... else .... end if;`
	`case ... is when ... => .... when ... else ... end case;`
répétitive	`loop .... exit when ...; ... end loop;` (avec ou sans `while` ou `for`)

etc .....[20]

Le premier temps fort est le **typage** (et même le **fort** typage). En premier lieu toute donnée doit appartenir à un type et être déclarée et il existe les traditionnels types prédéfinis:

---

[18] toujours les séquelles du COBOL!
[19] près de la moitié du code fut généré automatiquement (soit 1.200.000 lignes!)
[20] ceci pour rassurer; c'est d'abord du Pascal+ sans plus!

entier: `INTEGER (voire LONG_INTEGER et SHORT_INTEGER)`
réel: `FLOAT, DURATION`
booléen: `BOOLEAN`
caractère: `CHARACTER`
chaîne: `STRING`

mais le programmeur peut construire ses propres types utilisateurs avec des constructeurs syntaxiques:

tableau	`array .... of ...`
article	`record is .... end record;`
énumératif	`... is (.....);`
pointeur	`access`
numérique	`digits` ou `delta`

ainsi que certains sous types (contrainte de type scalaire) bien commodes pour contrôler des validités:

```
subtype T_TEMPERATURE is INTEGER range -35 .. +56;
```

ou dériver un type préalablement défini (créant ainsi un nouveau type disjoint du type père):

```
type T_NOUVELLE_TEMP is new INTEGER range -35 .. +56;
```

ou construire un type dérivé sans référence à un type de base:

```
type T_EXTRA_TEMP is range -35 .. +56;
```

Quelques remarques sur ces trois dernières déclarations:

Les données déclarées de type T_TEMPERATURE sont INTEGER et donc sont compatibles avec ce type; seule la contrainte est garantie (c'est déjà intéressant mais limité). Les données de type T_NOUVELLE_TEMP ne sont compatibles qu'entre elles (pas avec INTEGER ni T_TEMPERATURE); d'éventuels mélanges sont détectés dès la compilation évitant des erreurs conceptuelles (plus intéressant!). Quant à la dernière présentation, elle garantit en plus une portabilité complète d'une machine à l'autre (le compilateur lui-même choisissant le type père implicite: ici SHORT_INTEGER ou INTEGER).

Le **deuxième point fort** qu'il nous paraît important de mettre en exergue est la composition de modules réutilisables. Ces modules permettent de réaliser une abstraction du monde réel. Le cadre structurel permettant de réaliser en ADA ces entités est le paquetage, générique si possible (les concepts de type privé et de surcharge complétant agréablement la mise en œuvre). Le paquetage ADA rassemble, dans une même unité de compilation, **un type** (abstraction d'une des réalités du problème à résoudre), défini à cet endroit par sa structure de données, avec **ses fonctionnalités** (sous-programmes). Cette encapsulation d'un type avec ses outils[21] facilite, aussi, et grandement les mises au point et la localisation d'erreurs. Un paquetage est structuré en deux parties (spécifications et corps) compilables séparément. Le contrat et sa réalisation sont alors disjoints. Le contrat peut être supposé résolu pour les analyses à venir sans attendre la réalisation.

[21] concept de type abstrait (à défaut d'objet)

Nous avions prévu à cet endroit deux à trois pages d'exemples de réalisations de paquetage ADA. Les contraintes matérielles de publication des actes de ce congrès nous oblige à proposer ces pages au lecteur intéressé sous la forme de compléments qu'il est invité à nous demander. Cependant nous présentons ci-dessous la forme générale de la partie spécification d'un paquetage pour fixer les idées.

```
generic    -- ici des paramètres (de type en général)
          -- permettent la réutilisabilité et factorisent le code source

package P_ICI_SON_NOM is
     type T_ABSTRAIT is ..... ;   -- sa déclaration effective
                                  --    ou    private    pour    le    rendre
inaccessible

     function ........
     procedure .....
     -- ci-dessus les sous programmes dédiés au type T_ABSTRAIT

     private  -- si T_ABSTRAIT a été déclaré plus haut comme tel
              --   visible   pour   tous   mais   inaccessible   quand   même
directement
              -- il faut utiliser obligatoirement les méthodes
end P_ICI_SON_NOM;
```

6. Les avantages pédagogiques de ADA et quelques problèmes

Outre la couverture des modules d'enseignement (ADA fédérateur.....) citée plus haut dans le paragraphe n°2 et la possibilité d'illustrer les concepts actuels (type abstrait, généricité: essentiel pour la réutilisabilité, conception descendante et ascendante ....) notons encore quelques avantages:

La portabilité du langage permet aux étudiants de développer des applications en ADA sur des ordinateurs type PC personnels, puis de les transposer sur n'importe quel système de notre institut.

La normalisation impose aux compilateurs (quels qu'ils soient) des verdicts identiques notamment avec des références (chapitre, paragraphe, sous-paragraphe et verset pour les messages d'erreurs) au manuel de normalisation (document unique pour toute machine et utilisé par tous les étudiants!).

Les "contrats" ou contenus des parties "spécifications" des paquetages sont proposés aux étudiants comme travaux à réaliser et sont non modifiables (car compilés séparément par les enseignants). Les étudiants doivent réaliser, à leur guise et même en équipe et en parallèle, les corps associés mais ne peuvent en aucune façon en dénaturer ni le fond ni la forme.

Dans le même esprit, les enseignants peuvent proposer aux étudiants le corps, lui aussi compilé séparément, du paquetage dont le contenu est, par cet artifice, disponible mais invisible. Contenu que les étudiants "recopient" par renommage des composants. Ils peuvent utiliser ces composants (sortes de "corrigés" inaccessibles) puis les réaliser et les soumettre aux validations, à leur gré, un à un, par étapes successives en conservant ainsi à chaque fois

l'intégrité des réalisations. C'est en cela, nous l'avons dit, que ADA permet d'aller au fond sans se disperser.

Enfin parmi tous les modules d'enseignement s'appuyant sur ADA, celui de "**l'initiation aux activités parallèles**" est celui qui, avec un minimum de mots réservés supplémentaires (task, select, accept, entry, terminate, abort), permet de réaliser des applications spectaculaires et inaccessibles auparavant (tâches et rendez-vous). Ce module termine en apothéose les concepts ADA et redonne un coup de fouet motivant aux étudiants. Il permettra enfin une assimilation aisée des autres difficultés du programme enseignées plus tard notamment en système d'exploitation (concept de sémaphores et d'exclusion mutuelle par exemple).

Bien sûr tout n'est pas idyllique dans notre remise en cause pédagogique. Nous avons signalé plus haut le fait que peu d'entreprises utilisent ADA et cela se ressent dans notre recherche de stages de fin de scolarité (10 semaines). Il semble incongru, nous l'avons noté, que des techniciens supérieurs soient capables de pratiquer ADA réservé aux formations à BAC + 5; et pourtant ça marche!

Une contrainte qu'il ne faut pas occulter non plus est que cette reconversion, pour être réussie, nécessite un travail d'équipe conséquent (cette contrainte nous a été hélas déjà signalée comme rédhibitoire par certains collègues qui envisageaient des expériences analogues). Mais est-ce bien la faute à ADA?

Ne cachons pas enfin que ce langage est tout de même un peu "spécial" à enseigner. En effet il n'y a pas grand chose à rejeter ni à occulter au premier niveau de la présentation du langage: peut-on faire l'impasse sur des concepts aussi intéressants que surcharge, généricité, exception, spécifications et réalisations séparées.... Il faut en parler tout de suite et en même temps dès les premières heures de l'apprentissage. Aussi le traditionnel découpage de la matière à enseigner, sécurisant guide didactique, est presque inexistant: intéressant et bon défi pédagogique!

7. Conclusion

Ce papier est plus le reflet d'un témoignage qu'une habituelle contribution scientifique, nous en sommes conscients, mais son ambition était de convaincre, à défaut de faire regretter celles et ceux qui ne franchiront jamais le pas ADA pour leur enseignement, langage qui, rappelons le, est bien plus qu'un langage de programmation car aussi un remarquable vecteur de formation au génie logiciel.

Si certains départements d'I.U.T. informatique ont choisi ADA comme composante importante de la formation qu'ils assurent, d'autres hésitent encore et expérimentent (par exemple: quelques heures d'initiation à l'algorithmique parallèle via les tâches). Mais l'idée fait son chemin et le phénomène est en marche. Ceux des départements qui ont opté pour ADA appliquent à la pédagogie de l'informatique l'aphorisme que proposait J.P. Rosen à propos de la programmation: "Ceux qui se sont mis à ADA passent leur temps à énumérer leurs récriminations mais pas un ne retournerait à son ancien langage".

8. Bibliographie (succincte mais touchant au génie logiciel et à ADA et son enseignement)

[BAR 88] BARNES J., Programmer en ADA, 1988, InterEditions.
"L'un des premiers (au sens propre et figuré) livres sur le sujet"

[BOO 88] BOOCH G., Ingénierie du logiciel avec ADA, 1988, InterEditions.
"LE livre à lire dès qu'on découvre ADA pour concevoir orienté-objet"

[BOO 90] BOOCH G., Conception orientée objets et applications, 1992, Addison Wesley 92.
"LA suite logique du précédent!"

[LEB 94] LEBIB R., Incidences de la mise en œuvre des concepts du génie logiciel à travers l'utilisation du langage ADA sur la productivité en informatique de gestion, 1994, Thèse de doctorat Paris IX Dauphine.
"200 pages pour convaincre de passer de COBOL à ADA"

[GAU 91] GAUTHIER M., ADA un apprentissage, 1991, Dunod informatique.
"Livre destiné aux pédagogues qui ont déjà quelques notions du langage"

[OGO 90] OGOR R. et RANNOU R., Langage ADA et algorithmique, 1990 Hermès.
"Livre pour apprendre, issu d'un cours bien rôdé à l'ENST Bretagne"