L'enseignement du Génie Logiciel par un Projet de Groupe

N Habra & E. Dubois
Institut d'Informatique
Facultés Universitaires Notre-Dame de la Paix
21, Rue GrandGagnage,
B-5000 Namur Belgique-
Tel +32 (81) 72 49 95
Email nha/edu@info.fundp.ac.be

1 Introduction : Contexte et Objectifs

L'expérience décrite dans cet article concerne l'enseignement du Génie Logiciel dans un curriculum universitaire de maîtrise en informatique. Les étudiants concernés sont soit dans la 4eme année d'un cycle complet de 5 ans conduisant au grade de Maîtrise, soit dans la deuxième année d'un cycle complémentaire (après d'autres études universitaires) conduisant au même grade.

L'enseignement du Génie Logiciel GL comprend tous les aspects —in the large— d'un développement rationnel et rigoureux de logiciels: la couverture de toutes les étapes du développement à partir de la spécification des besoins jusqu'à l'implémentation et la maintenance; le développement de logiciels opérationnels en grandeur réelle; les aspects liés au travail en groupe…

L'enseignement du GL dans notre faculté comprend principalement un cours théorique (un module) couplé au grand projet (4 modules) que nous décrivons ici. Ce projet comprend la réalisation complète d'un logiciel de gestion hospitalière par groupes de 6-7 étudiants tout au long d'un semestre.

Au cours théorique sont étudiés les principes et les concepts du GL (voir par ex. [Fai84,Ghe91]); l'accent est mis en particulier sur les approches modernes: les méthodes formelles, l'approche transformationnelle, la conception Orientée-Objet… Le but premier du laboratoire est donc de mettre en pratique ces principes et concepts.

En même temps, ce laboratoire est une occasion unique pour les étudiants d'acquérir une première expérience dans le développement d'un grand logiciel; une expérience qu'ils pourraient faire valoir pour leur entrée dans la vie professionnelle. L'orientation de notre Faculté étant l'informatique de gestion, la majorité de nos diplômés sont appelés à une carrière dans les banques, les assurances, les administrations publiques et privées,…

Nous nous retrouvons ainsi, devant une série de compromis à faire entre les techniques du présent et les principes qui soustendraient les techniques du futur. D'une part, nous souhaitons mettre en pratique des méthodes et des principes qui nous semblent fondamentaux en GL mais qui sont en décalage par rapport à la pratique professionnelle qui les jugent encore trop académiques. D'autre part, nous essayons de faire pratiquer à nos étudiants des langages et méthodes qui pourraient être utilisés directement à la sortie de la Faculté. Les choix liés à l'organisation de ce laboratoire sont déterminés par ces compromis.

2 Le Cycle de Vie Proposé

Le modèle de cycle de vie que nous proposons de suivre est basé sur une approche transformationnelle. Nous considérons comme démarche principale du développement du logiciel une succession de transformations (documentées et justifiées) conduisant d'une définition informelle des besoins à un code opérationnel. On passe ainsi par petits pas de l'abstrait vers le concret, du déclaratif vers l'opérationnel, du moins formel au plus formel.

Dans le modèle de cycle de vie, nous nous basons sur la nature des décisions à prendre dans le processus de développement pour regrouper celles-ci en différentes *phases* successives représentant chacune une unité organisationnelle comprenant des décisions homogènes. Nous avons ainsi les étapes suivantes: l'acquisition et l'analyse des besoins, la spécification, la conception et l'implémentation.

Le concept de *produit* et le concept d'*activité* sont clairement distingués du concept de la *phase*. Bien que des phases comprennent une activité principale et/ou un produit principal, certaines activités (ex. le management) et certains produits (ex. la documentation) concernent différentes phases. De même, nous considérons la validation non pas comme une phase à part réalisé à la fin de la conception du code mais comme activité à réaliser à chaque phase.

2.1 Phases et Activités

Le projet des étudiants couvre tout le cycle de vie du logiciel, ainsi les étapes principales du projet suivent les phases de ce cycle de vie.

Analyse des besoins: la toute première étape de l'analyse des besoins comprenant l'acquisition des besoins étant vue par un cours précédent, les étudiants sont dispensés de cette étape. Ainsi, comme point de départ du projet, ils reçoivent un premier document d'analyse de besoins. Pour s'approcher des cas réels, ce document inclue volontairement des incomplétudes, inconsistances et ambiguïtés; la première phase consiste donc en une étude critique de ce document pour lever ces imperfections.

Spécification des besoins: cette étape consiste à produire à partir du document informel de l'analyse une spécification formelle des besoins. Cette étape est ainsi un exercice de spécification dans lequel les étudiants apprennent à utiliser un langage formel de spécification.

Conception: la conception consiste à transformer la spécification (le *quoi*) en solution (le *comment*). Cette solution reste abstraite dans le sens ou elle doit être la plus indépendante possible de la machine sur laquelle elle va tourner. Le produit de cette conception est une architecture logique, c-à-d une découpe en une hiérarchie de *modules* bien définis reliés entre eux par des *relations* bien définies (ex. utilise, hérite de,…). Chaque module représente la version abstraite d'une partie du code: un algorithme représentant une procédure, un type abstrait représentant une structure de données ou une description abstraite d'un objet.

Nous divisons cette phase en deux sous-phases: une conception statique suivi d'une conception dynamique. La conception statique est faite sous l`hypothèse que le système est mono-utilisateur et que toutes les fonctions sont instantanées. La conception dynamique est un raffinement de la conception statique dans lequel cette hypothèse est levée. L'idée sous-jacente est de diviser les difficultés: ainsi la première sous-phase se concentre sur le problème d'une bonne découpe en modules sans se

soucier des problèmes de concurrence qui sont traités séparément dans la deuxième sous-phase.

Implémentation: l'implémentation consiste à transformer la solution abstraite en une solution concrète tournant sur machine réelle. Ainsi, les modules deviennent des morceaux de code: des procédures, des fichiers ou des parties de base de données, des classes, des objets…

En plus des activités décrites ci-dessus et qui consistent pour chaque phase en la production du produit principal exigé par cette phase, d'autres activités sont également demandées tout au long du cycle de vie. Ainsi, on demande aux étudiants la *documentation* non seulement du produit final mais aussi de toutes les décisions prises à chaque étape du développement ainsi que la justification de ces décisions. On demande également, à chaque groupe une *gestion* rationnelle des ressources en temps et hommes, cet aspect est discuté dans la Section 4.

2.2 Modèles et Langages

Le choix des langages utilisés pour les différentes étapes est lié à notre volonté de faire un compromis entre les langages formels répandus dans le monde académique d'une part et les langages classiques utilisés actuellement dans l'industrie d'autre part. Nous évitons d'utiliser un langage formel du style "wide-spectrum" (ex. VDM [BJ78,Jo86,] ou RAISE [Raise92]) couvrant toutes les phases du cycle de vie et supportant une approche transformationnelle complètement formalisée. Par contre, nous proposons des langages formels non traditionnels au niveau des spécifications et des langages traditionnels au niveau de l'implémentation.

Le document initial de l'analyse des besoins est rédigé dans un langage du type Merise [BP83,RT83] vu par les étudiants durant des cours antérieurs. Les spécifications formelles sont écrites en Z [Hay78]. La conception statique est décrite dans le langage RSL [Raise92], un dérivé de VDM permettant une grande liberté d'expression pour décrire des modules dans un style fonctionnel ou un style "type abstrait". La conception dynamique est décrite en Oblog [SSE87], un langage de conception Orienté-Objet, au développement duquel participe notre Faculté. En fin de compte, l'implémentation est faite dans un environnement traditionnel en Cobol couplé à un SGBD du type SQL et au gestionnaire d'interface DecForm de Dec.

Ce choix d'une grande diversité des langages est certes très consommateur en temps. Cependant, il donne aux étudiants à la fois une bonne familiarisation avec les langages formels et leur différents degrés d'abstraction et une première expérience avec un environnement de développement traditionnel. De plus, ce passage d'un paradigme à un autre permet aux étudiants d'avoir un regard critique en touchant de prés aux limites et aux mérites des langages trop académiques et des langages traditionnels.

3 Les Aspects Méthodologiques

Les aspects méthodologiques, notamment en ce qui concerne la production d'une bonne architecture logique occupent une place centrale dans ce laboratoire. En effet, nous considérons l'activité de la conception d'une telle architecture comme une des plus créatives et des moins systématiques du processus du développement.

Le premier principe poursuivi en ce domaine est de laisser aux étudiants la plus grande liberté pour développer leur propre méthode de conception. Ainsi, au cours théorique nous

décrivons surtout les caractéristiques générales d'une bonne découpe en modules (relations bien définies, abstraction, encapsulation, faible couplage,…) [Par72]. Nous parcourons ensuite quelques démarches particulières pour y arriver: démarche descendante ou ascendante, démarche orientée-objet ou orientée-fonction,… Nous sommes convaincus que toute démarche a ses mérites et ses défauts; ceux-là dépendent de plus du cas développé. Ainsi, nous laissons aux étudiants au niveau du laboratoire la liberté totale de choisir la démarche méthodologique pourvu qu'ils puissent justifier leur architecture eu égard des critères du cours. A la fin du laboratoire, en comparant les différentes architectures produites, nous constatons que les bonnes architectures sont souvent des architectures mixtes comprenant une découpe orientée-fonction au niveau des modules du haut de la hiérarchie et une découpe orientée-objet au niveau des autres modules. La démarche de production de ces architectures semble aussi être une démarche mixte: analyse descendante des fonctions jusqu'à un certain niveau de découpe suivi d'une analyse ascendante. Ce constat nous confirme dans le choix de ne pas obliger les étudiants à coller à une seule méthodologie rigide.

Le deuxième principe méthodologique que nous proposons est de découper l'activité de conception en conception statique suivi d'une conception dynamique, les difficultés de l'élaboration d'une bonne architecture sont ainsi décomposées. Cette idée semble porter ses fruits dans le cas de gestion hospitalière que nous proposons aux étudiants de résoudre. En effet dans ce cas, comme dans beaucoup de cas de gestion classique, les aspects dynamiques sont limités à l'interface utilisateur et à certains accès concurrents à la BD (ex. dans notre cas la réservation concurrente des chambres pour des malades), le reste des fonctionnalités demeure assez statique. Ainsi, la conception statique basée sur les hypothèses simplificatrices (mono-utilisateur, fonctions instantanées) produit une assez bonne découpe qui servira de base pour être raffinée en tenant compte de la dynamique. Il est clair que pour des systèmes hautement concurrents (ex. systèmes en temps-réel), les aspects dynamiques sont plus déterminants et doivent donc intervenir plus tôt dans la démarche de conception.

4 Les Aspects de Communication et de Management

Depuis plus de 10 ans que nous encadrons ce laboratoire, les aspects de la gestion du groupe et de la communication inter-personnelle s'avèrent de plus en plus importants; ils sont souvent déterminant dans la réussite ou l'échec du projet. Ainsi, nous accordons de plus en plus d'importance à ces aspects et nous attirons l'attention des étudiants sur leur importance.

Le projet est réalisé en groupes de six personnes choisis au hasard. Il n'y a pas de structure imposée concernant le partage de travail au sein du groupe. On suggère de désigner un coordinateur et un secrétaire pour chaque phase du projet, et de pratiquer une tournante pour l'attribution de ces deux rôles. On suggère également de déterminer un planning de travail avec description détaillée des ressources (personnes, temps…) allouées à chaque tâche; ce planning peut être bien évidement révisé en fonction des imprévus. On demande également d'avoir des compte-rendus des réunions du groupe dans lesquels sont notés les décisions (techniques et/ou managériales) prises à chaque réunion.

Ces aspects managériaux sont suivis actuellement par un staff distinct du staff technique. Ce suivi permet d'aider les étudiants à gérer le projet d'une façon plus efficace, à bien organiser leur réunion et à mieux résoudre les divergences éventuelles; il permet surtout d'attirer l'attention des étudiants sur l'importance d'une bonne gestion (ex. un partage clair et réaliste

des responsabilités), sur la difficulté inhérente aux projets informatiques de faire des plannings (ex. déterminer a priori le temps nécessaire pour telle ou telle étape) et sur tous les aspects de communication entre personnes différentes regroupées pour la réalisation d'une tâche bien déterminée. Les échos que nous avons eu de nos anciens étudiants, actuellement confrontés dans leur vie professionnelle au travail du groupe (en tant qu'exécutants ou en tant que managers), nous confirme dans notre choix d'accorder autant d'importance aux aspects managériaux qu'aux aspects techniques.

Bibliographie

BJ78 D. Björner et C.B. Jones, *The Vienna Development Method: the Meta Language,* LNCS Vol. 6, Springer-Verlag, 1978.

BP83 F. Bodart et Y. Pigneur, *Conception Assistée des Applications Informatiques: Etude d'Opportunité et Analyse Conceptuelle,* Masson, Paris, 1983.

Fai84 R. Fairley, *Software Engineering Concepts,* Mc Graw-Hill, 1984.

Ghe91 C. Ghezzi, *Fundamentals of Software Engineering,* Prentice-Hall, 1991.

Hay87 I. Hayes, *Specification Case Studies,* Prentice-Hall, 1987.

Raise92 Raise Language Group, *The Raise Specification Language,* BCS Practionners Series, Prentice-Hall, 1992.

Jo86 C.B. Jones, *Systematic Software Development Using VDM,* Prentice-Hall, 1986.

Par72 D.L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the ACM,* 15(12), December, 1972.

RT83 A. Rochfeld et H. Tardieu, "MERSIE: an Information System Design and Development Methodology", *Information & Management,* 6(3), June, 1983.

SSE87 A. Sernadas, C. Sernadas et H.D. Ehrich, "Object_Oriented Specification of Databases: an Algebraic Approach", in *Proceedings of the 13th Int. Conf. on Very Large Data Bases,* P. Hammersley (Ed), Brighton, UK, September, 1987.