LMO'99:
Langages et Modèles à objets

Hanane Naciri*

2 Mars 1999

Mots clés: Modèles à objets, Génie logiciel, Réutilisabilité des composants, Tests, Réversibilité.

1 Introduction

La conférence "LMO'99 : Langages et modèles à objets" qui s'est tenue à Villefranche-sur-mer en Janvier 99 a offert un cadre pour comparer les recherches sur les langages et modèles à objets. Les principaux domaines traités lors de ce congrès ont été la réflexibilité, le génie logiciel, les systèmes répartis, les schémas et patrons de conception, l'induction, la classification et les bases de connaissances et modèles à objets.

Ce compte rendu de la conférence LMO'99 rédigé dans le cadre du cours " Modèle à objets commun au G.L. et aux B.D " (Resp. Roger Rousseau [1]) résume l'exposé de Houari A.Sahraoui sur l'impact du couplage sur la réutilisabilité des classes, l'article de J.Jezequel et D.Deveaux sur les classes auto-testables et la conférence de Bertrand Meyer sur le développement réversible en Eiffel. Dans ce résumé, seuls les apports de la technologie objet pour le génie logiciel seront décrits.

2 Premier article : Des classes auto-testables

2.1 Fiche technique

Session	:	Génie logiciel, présidée par Bertrand Meyer[2]
Titre	:	Des classes auto-testables
Auteurs	:	Daniel DEVEAUX [3], Jean-Marc JEZEQUEL[4]
Mots clés	:	Test logiciel, Programmation contractuelle, Vérification de logiciel, Validation de logiciel

*DEA Informatique, Université de Nice Sophia Antipolis, École Supérieure en Sciences Informatiques ESSI

1. I3S et Université de Nice-Sophia Antipolis

2.2 Résumé

La technologie objet intervient dans toutes les étapes du cycle de vie du logiciel mais elle n'accorde qu'une faible importance aux tests. Les tests sont pourtant indispensables à toute forme de programmation. En particulier, la programmation par objets favorise la réutilisation des composants. Ces composants nécessitent d'être testés de façon approfondie. Une erreur non détectée dans un composant peut se répercuter dans plusieurs applications.

Cet article introduit le concept de classe auto-testable qui organise et facilite les opérations de mise au point de composants logiciels dans un contexte de développement par objets. Cette approche est basée sur les concepts de programmation par contrat (définie par B.Meyer) et sur les techniques de documentation automatique. Les tests définis dans cet article traitent uniquement de la validation du comportement d'un logiciel ou d'un composant logiciel (tests fonctionnels).

L'approche reposant sur les tests unitaires (validation de chaque procédure) et les tests d'intégration (validation des enchaînements d'appels procéduraux) ne s'applique pas aux systèmes à objet. Une méthode ne peut pas être testée individuellement. Son comportement est lié à l'état de l'objet auquel elle est appliquée. De plus, un objet ne peut être testé sans ses sous-traitants (ses fournisseurs).

Pour ces raisons, dans les systèmes à objet on il n'existe que deux niveaux de tests fonctionnels: les tests de validation de composants (activation d'un scénario d'envois de message) et le test de validation d'application (activation d'un scénario indépendant des modalités d'implémentation).

L'approche contractuelle introduite par B.Meyer facilite la mise au point et les tests de validation des composants. Elle exprime les droits et obligations de chaque partie (les contrats de clientèle et d'héritage).

En Eiffel, cette approche repose sur le mécanisme d'assertions: assertions de contrat et assertions internes. Les assertions de contrat contrôlent la cohérence du système et testent son comportement attendu. Les assertions internes vérifient qu'une étape importante du programme est bien atteinte (approche de prévision).

L'idée principale du concept de classes auto-testables est analogue à celle du concept d'assertion: mettre les scénarii de tests dans le code source des classes.

L'avantage de cette approche est que les tests et les composants représentent une seule unité. Dans ce cas l'évolution des tests suit forcément l'évolution de la source (principe d'unicité du source). Cette approche permet aussi de livrer des librairies de composants avec leurs scénarii de tests, ce qui permet au développeur d'évaluer les composants avant de les utiliser (principe de disponibilités des tests).

L'objectif des auteurs est d'obtenir un outil pédagogique simple à mettre en œuvre, adéquat avec les langages hôtes, peu coûteux et qui permet de sensibiliser les utilisateurs aux problèmes du test des systèmes à objets. Pour cela, ils ont développé une structure incluant le support de contrat (les assertions), un environnement de test (routines d'affichage, de contrôle et de couverture de code), des méthodes de tests (scénarii de tests) et un lanceur de test (exécution

[2]ISE, Santa Barbara, Californie
[3]Lab. Valoria, Equipe Aglae, UBS, , Vannes
[4]IRISA/CNRS, Projet Pampa, Univ-Rennes, Rennes

d'une séquence de tests).

La mise en œuvre des classes auto-testables part des spécifications fonctionnelles de classes (diagramme UML) et passe par cinq phases de développement: définition de l'interface des méthodes (validation du comportement de l'application), implémentation (développement progressif du code en introduisant les traces et les assertions), validation (enchaînement de séquences de tests et production d'une documentation incorporant les résultats des validations effectuées), intégration et enfin exploitation et maintenance (masquer ou retirer les instruments de test de l'exécutable, et en cas de maintenance corrective ou évolutive, pouvoir les réactiver pour vérifier la non régression des composants).

L'approche des classes auto-testables vérifie seulement le bon fonctionnement de la structure de l'application. Elle ne garantit pas que l'application réponde aux spécifications du cahier des charges initial. De plus, les tests de paquetages (ensemble de classes) ne sont pas encore automatisés. Cela peut se faire par un script prenant en charge le lancement récursif des tests et la production des rapports de tests. Cette approche prend en compte l'héritage (le lanceur de test fait appel aux méthodes de tests des classes ancêtres) et la généricité (définir des méthodes de tests dans une classe abstraite pour factoriser le code du test des sous-classes).

Pour mieux comprendre le concept des classes auto-testables, les auteurs ont présenté un prototype d'implémentation en Eiffel qui offre un support assez complet des contrats. Pour les autres langages (Perl, Java et C++), ils ont réalisé des prototypes en développant avant un mécanisme simple de contrats.

Cette expérimentation sur quatre langages à objets assez différents permet aux auteurs d'envisager d'étendre cette approche à d'autres langages à objets. Les auteurs se fixent aussi comme objectifs futurs une meilleure intégration des différents langages, une définition plus précise des objectifs des tests et une automatisation complète des pilotes de tests.

2.3 Opinion personnelle

Le concept de classes auto-testables s'inspire fortement de l'approche de la programmation contractuelle. Il montre l'importance du rôle du mécanisme d'assertions pour améliorer la fiabilité des applications objets et pour faciliter la phase de test d'un logiciel.

Cette étude offre seulement un cadre pédagogique simple pour sensibiliser les utilisateurs aux problèmes de test des systèmes à objets. Elle n'apporte pas de solution générale à ce problème. Elle peut être une base pour appliquer d'autres approches plus formelles.

L'idée principale de ce concept est de mettre les scénarii de tests dans le code source. Cela peut nuire à la lisibilité de ce dernier. L'information émanant du code source risque donc de se perdre au milieu d'une profusion de tests.

3 Deuxième article : Impact du couplage sur la réutilisablité des classes

3.1 Fiche technique

Session	:	Génie logiciel, présidée par Bertrand Meyer[5]
Titre	:	Impact du couplage sur la réutilisabilité des classes (une étude de cas)
Auteurs	:	Houari A.SAHRAOUI, Hakim LOUNIS[6], Yida MAO[7]
Mots clés	:	Qualité de logiciel, Réutilisabilité, Couplage, Modèles prédictifs

3.2 Résumé

La réutilisabilité des composants existants dans la technologie objet apporte beaucoup d'avantages au monde du génie logiciel. Ce besoin de réutilisabilité vient du fait que beaucoup d'éléments des systèmes logiciels se ressemblent. Il devrait donc être possible d'exploiter ces points communs et d'éviter d'avoir à recréer des composants semblables aux composants existants.

L'importance de la réutilisabilité est évidente. En particulier, la réutilisabilité a un effet indirect sur tous les facteurs de qualité du logiciel. Résoudre le problème de réutilisabilité signifie essentiellement que l'on pourra écrire moins de code, et donc consacrer plus d'efforts à l'amélioration des autres facteurs de qualité, comme la validité, la robustesse, etc.

La réutilisabilité des composants déjà existants offre un gain en temps et en efficacité. Mais des questions subsistent : est-il facile de réutiliser les composants ? comment le faire ? Les classes fortement dépendantes de leur environnement ne sont pas forcément réutilisables. En général, la capacité de réutilisation d'un composant d'une application existante est d'autant plus élevée que sa dépendance vis-à-vis des autres composants de l'application est faible.

Quelles sont alors les formes de dépendances ? Et quelles sont les caractéristiques permettant indirectement de mesurer la réutilisabilité d'un ensemble de composants (classes) ?

Le schéma de la Figure 1 résume l'approche suivie par cette équipe. Ils ont choisi pour cette étude le couplage comme métrique pour mesurer la réutilisabilité des composants. Ils ont posé l'hypothèse suivante : le couplage entre une classe et son environnement a un impact sur la capacité de réutiliser cette classe et affecte donc d'une certaine manière la réutilisabilité. Cette hypothèse suppose qu'il y a une relation de cause à effet entre la réutilisabilité et le couplage (qui est directement mesurable). Pour tester leur hypothèse, les auteurs ont choisi deux ensembles de métriques existants : les métriques de couplage au niveau de la conception et les métriques de couplage au niveau de l'implémentation (code source). Puis, ils ont construit des modèles prédictifs à partir d'un échantillon de classes existantes. Ces modèles permettent de prédire

[5]ISE, Santa Barbara, Californie
[6]CRIM Montréal(QC), Canada
[7]ACD Systems Ltd,Victoria(BC), Cananda

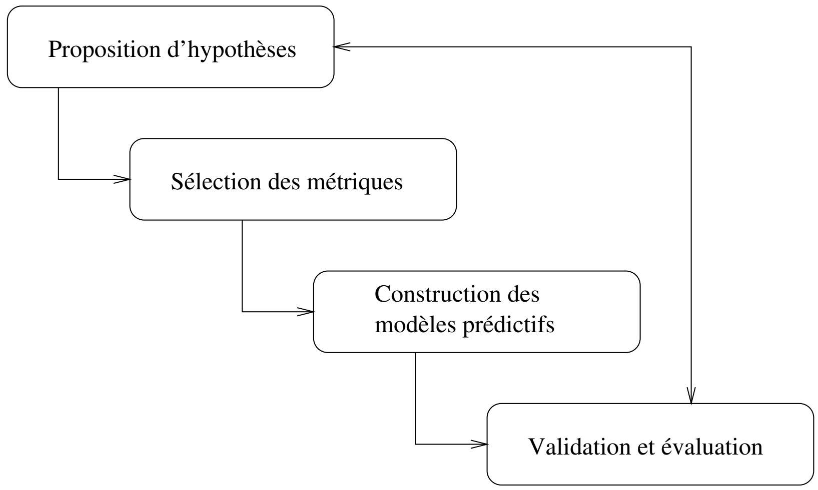

FIG. 1 – *L'approche suivie pour étudier l'impact du couplage sur la réutilisabilité*

la capacité de réutiliser une classe à partir de métriques de couplage. Ils ont ensuite validé empiriquement chacun de ces modèles en mesurant leur complétude et leur exactitude.

La réutilisabilité consiste d'abord à trouver le composant qui correspond le mieux aux besoins (services offerts et attendus par la classe) pour pouvoir l'adapter à moindre coût à ceux-ci. Mais il est difficile de mesurer directement la réutilisabilité d'un ensemble de composants. Pour mesurer la réutilisabilité de manière empirique, différents aspects peuvent être considérés :

1. La quantité de travail nécessaire pour réutiliser un composant d'une version d'un système à une autre version du même système.

2. La quantité de travail nécessaire pour réutiliser un composant d'un système à un autre, au sein du même domaine

Dans cet article, c'est l'étude du deuxième aspect qui a été choisie comme mesure empirique. Exemples de mesures pour la réutilisabilité :

– Réutilisable à 100%

– Réutilisable, avec moins de 25% de transformation

– Réutilisable, avec plus de 25% de transformation

– Non réutilisable

L'hypothèse a été vérifiée sur les deux niveaux de métriques de couplage:

– **Métriques de couplage au niveau de la conception**. Mesure du couplage au niveau statique: la nature, le mode et le sens de dépendance d'une classe envers d'autres classes.

– **Métriques de couplage au niveau de l'implantation**. Messure du couplage d'une manière plus précise: la nature des liens entre les classes, la complexité de l'information partagée, le mode de transmission de cette information et l'usage qui en est fait.

Voici le résumé des étapes qu'ils ont adoptées pour construire leurs modèles prédictifs à partir d'un échantillonnage de classes existantes:

- Collecte des données (l'échantillon choisi est un système multi-agents ouvert appelé LALO)
- Techniques de construction (utilisation de l'algorithme d'apprentissage symbolique automatique C4.5)
- Évaluation des modèles (Exactitude, Complétude, Pertinence globale)

En conclusion, les auteurs se fixent comme **objectifs futurs** :

- Généralisation de l'approche à l'ensemble des caractéristiques de qualité du produit logiciel.
- Introduction du contexte dans la définition et l'utilisation des modèles prédictifs.
- Combinaison de plusieurs modèles (maintenabilité, réutilisabilité, ...).
- Optimisation des modèles.
- Utilisation des modèles dans la définition des stratégies de reconstruction (refactoring : ex. transformer l'agrégation en généralisation).

3.3 Opinion personnelle

Les modèles à objets offrent un développement à base de composants. Ces composants ne sont pas toujours immédiatement et facilement réutilisables. L'approche suivie par cette équipe mesure empiriquement la réutilisabilité des composants. Elle se base sur l'hypothèse suivante: le couplage entre une classe et son environnement a un impact sur la capacité de sa réutilisation.

Les auteurs ont décrit une expérience de vérification de cette hypothèse à l'aide d'une technique d'échantillonage. Cette étude permet seulement d'automatiser la détection de composants potentiellement réutilisables mais pas de généraliser ces résultats à un niveau plus large.

Les auteurs mesurent la réutilisabilité indépendamment des autres facteurs de qualité des logiciels. Du point de vue du génie logiciel, il aurait été intéressant de comparer le critère de réutilisabilité avec d'autres facteurs de qualité.

4 Conférence invitée

4.1 Fiche technique

Session	:	Génie logiciel, présidée par Bertrand Meyer[8]
Titre	:	Analyse, conception, implémentation: développement réversible en Eiffel
Auteur	:	Bertrand Meyer
Mots clés	:	Génie logiciel, cycle de vie des logiciels, réversibilité, développement incrémental, Eiffel

4.2 Résumé

B.Meyer a tout d'abord présenté lors de sa conférence les grandes unités du logiciel:

- Analyse, conception, implémentation.
- Documentation: le logiciel et sa documentation doivent représenter un seul produit. Il faut lutter contre la distinction entre les deux (l'évolution de la documentation doit suivre l'évolution du logiciel).
- La technologie objet au service des applications scientifiques, temps réel, gestion ...

Pour B.Meyer, la technologie objet offre des services à beaucoup de domaines : scientifique, temps réel, gestion ... Il donne comme exemple les bibliothèques Eiffel de composants numériques et leur utilisation dans le domaine numérique. L'objectif des langages à objets, en particulier Eiffel, est de permettre un développement à base de composants. Ils permettent la construction de grandes bibliothèques de milliers de composants réutilisables pour programmer rapidement toutes sortes d'application, avec un haut niveau de qualité.

L'objectif des modèles à objets est de s'écarter des cycles de vie classiques des logiciels afin d'éviter leurs inconvénients. Par exemple, le cycle de vie en cascade d'un logiciel pose les problèmes suivants :

- Apparition tardive du code.
- Absence de prise en compte des changements.
- Division synchrone du travail.

Le but du génie logiciel est d'offrir des modèles de développement en continu "SEAMLESS" (Voir Figure 2). Ces modèles offrent un ensemble de techniques et de méthodes utilisables à

[8] ISE, Santa Barbara, Californie

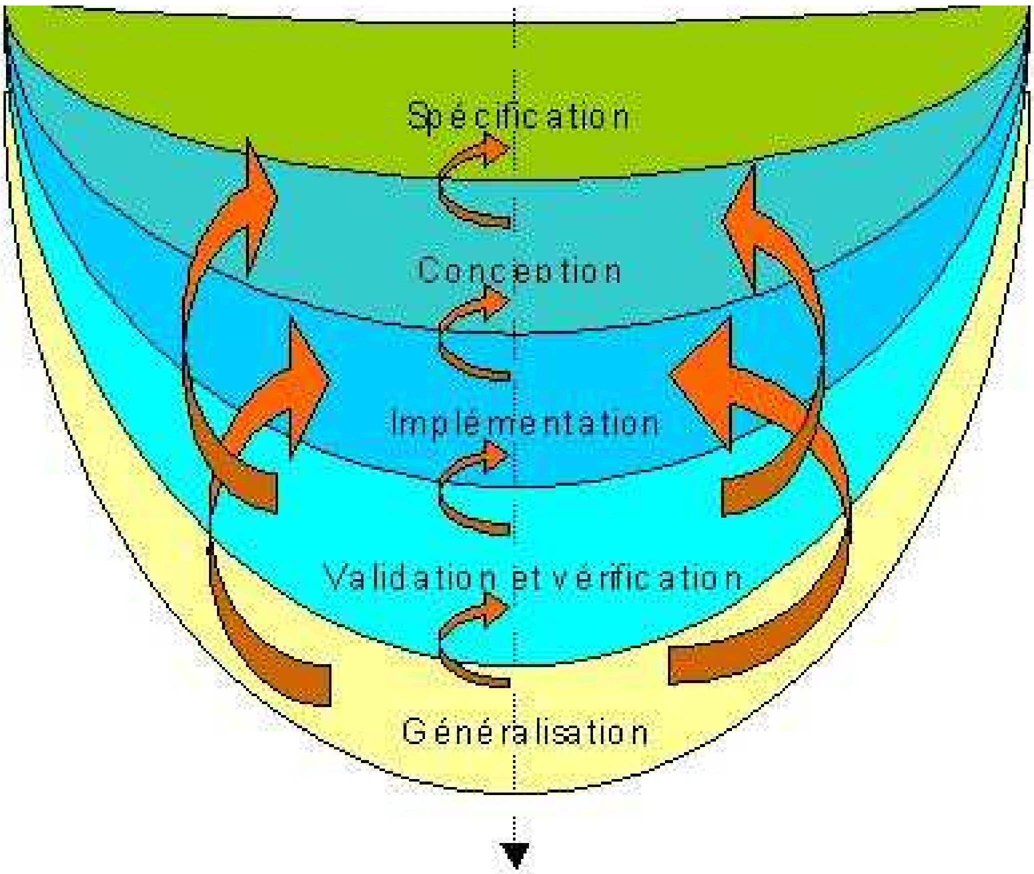

FIG. 2 – *Modèle de développement en continu SEAMLESS.*

n'importe quelle étape du cycle de vie d'un logiciel. Cela suppose qu'on utilise une notation cohérente de l'analyse à la maintenance, en passant par la conception et l'implémentation.

B.Meyer a insisté sur l'intérêt de chercher des points communs entre la modélisation et l'implémentation au lieu d'insister sur les différences entre les langages et les méthodes de conception.

B.Meyer a traité la réversibilité avec les retours potentiels et a remis en cause le schéma théorique. Il a insisté sur le rôle d'avoir une notation unique pour éviter beaucoup d'inconvénients lors d'un changement ou d'un ajout pendant la phase finale du développement. Il a aussi abordé l'utilisation d'un formalisme de description de classes. Pour assurer une meilleure réversibilité, il faut être en même temps précis et abstrait au niveau de la phase d'analyse. Il faut trouver un compromis pour ces deux critères.

Eiffel offre un développement réversible. Il permet le va et vient entre le modèle de conception et le code. Il sépare la modélisation du codage grâce à ses environnements interactifs: Eiffel Case pour la modélisation et Eiffel Bench pour le codage. Ce dernier repose sur une compilation incrémentale proche de l'interprétation (l'utilisateur change un bout de code et il ne veut pas attendre longtemps, d'où le besoin de la compilation incrémentale). Eiffel répond aussi aux besoins des utilisateurs au niveau de la génération de code (ex. générer du C++ à partir d'Eiffel).

Finalisation:

– Génération du code final.

– Optimisation.

– Elimination du code mort.

– Simplification des appels (liaisons statiques).

– Expansion en ligne.

4.3 Opinion personnelle

B.Meyer milite pour une notation unique du schéma allant de la conception à l'implémentation. L'exemple de l'association met en évidence la difficulté de traduire cette relation au niveau de l'implémentation. Actuellement, les langages à objets ne fournissent pas la représentation directe de ce type de mécanisme, qui reste à mon avis trés difficile à implémenter.

Mis à part ce problème, la plupart des langages à objets permettent un développement réversible. Cela offre un environnement interactifs entre les différentes phases de développement du logiciel. La réversibilité propose alors une alternative aux cycles de vie traditionnels.

5 Conclusion

La conférence LMO'99 était une occasion de suivre de près l'évolution des langages et des modèles à objets mais aussi de rencontrer des chercheurs de ce domaine.

L'approche objet apporte des solutions potentielles à tous les problèmes du génie logiciel pour obtenir à moindre coût des systèmes évolutifs, réutilisables, efficaces et bien documentés.

Ce résumé n'a traité que de la réutilisabilité, la fiabilité et la réversibilité comme facteurs de qualité. Il reste à étudier l'impact de la technologie objet sur les autres critères.

Références

[1] Jacques Malenfant, Roger Rousseau. LMO90: Actes du Colloque Langages et Modèles à Objets. *Villefranche-sur-mer, France*, 27-29 janvier 1999.