UMLChecker : un outil vérifiant la conformité entre une spécification et du code dans un enseignement de programmation orientée objet

Arnaud Lanoix[1,2] and Emmanuel Desmontils[1,3][0000−0002−6150−278X]

[1] LS2N, Nantes Université, France
[2] IUT de Nantes, France
[3] INSPÉ de Nantes, France
arnaud.lanoix-brauer@univ-nantes.fr ; emmanuel.desmontils@univ-nantes.fr

Résumé Nous présentons UMLChecker, un outil permettant de vérifier la conformité entre des spécifications UML et du code objet. Il est important que les étudiants en informatique soient capables de produire le code correspondant à un diagramme de classes. L'outil automatise le processus de vérification de la correction d'un programme dans un langage à objets selon un diagramme UML. Il a été utilisé en première année de BUT d'informatique. Basé sur les capacités d'introspection du langage cible et sur la disponibilité d'outil de tests, il permet d'intégrer la conception et la programmation UML dans le processus d'apprentissage. Il favorise l'auto-évaluation et l'évaluation continue des étudiants.

Keywords: Programmation orientée objet · UML · Test de conformité

Dans le cadre du programme national du BUT [4] informatique [3], plusieurs modules abordent les concepts, méthodes et langages de la Programmation Orientée Objet (POO), largement reconnue. En première année, 2e semestre, une ressource se concentre sur deux aspects complémentaires : i) la conceptualisation d'un problème à travers une conception orientée objet avec l'utilisation de diagrammes de classes UML, ii) le développement de programmes conformes à la POO. L'un des apprentissages critiques du programme national est l'AC11.01, qui consiste à implémenter des conceptions simples. Ceci nécessite un lien fort entre la conception UML et la programmation en POO. En d'autres termes, à la fin de la première année, un étudiant doit être capable de lire et comprendre la spécification d'un problème présentée sous forme de diagramme de classes UML, puis produire le code correspondant sans recourir à des outils de génie logiciel dédiés [3,7]. Outre la nécessité d'une traduction rigoureuse, cela exige une compréhension approfondie des subtilités d'un diagramme de classes pour les refléter fidèlement dans le code. Par exemple, l'étudiant doit pouvoir rendre avec précision la visibilité des attributs, les cardinalités, la navigabilité, les noms de rôles, etc. Ces compétence sont complexes à enseigner [5,9,11]. Elles faisaient jusqu'ici l'objet de situations didactiques demandant une forte implication de

4. Bachelor Universitaire de Technologie

l'enseignant. En effet, vérifier les travaux réalisés par 25 à 30 étudiants d'un groupe TD pendant la séance est extrêment lourd à gérer, car s'assurer qu'un étudiant a correctement traduit un diagramme UML est compliqué et prend du temps [12,13]. Nous avons donc cherché à améliorer les retours étudiants en automatisant une partie du processus de vérification. Sans en arriver à une situation adidactique (au sens de Broussau [2]), nous cherchons à permettre à l'enseignant de se concentrer sur l'accompagnement des étudiants ayant des difficultés plus importantes.

Pour s'assurer qu'un étudiant a bien compris une spécification UML et a correctement développé le code correspondant, notre approche consiste à évaluer la conformité de ce code par rapport à la spécification en nous appuyant sur le développement guidé par les tests. Cette évaluation, également connue sous le nom de test de conformité, a déjà fait l'objet de recherches approfondies [1,8,10]. Cependant, sa mise en œuvre implique généralement des processus complexes qui peuvent dépasser les capacités d'un étudiant en informatique en première année de BUT.

Afin de faciliter l'évaluation de la conformité structurelle du code des étudiants par rapport à une spécification UML, nous avons développé UMLChecker. Cet outil permet à un enseignant de générer automatiquement un ensemble de cas de tests JUnit [4,6] spécifiques à partir de la spécification UML. Une caractéristique clé de notre outil est qu'il n'ajoute pas de complexité aux étapes de développement, étant donné que les étudiants sont déjà habitués à valider leurs développements par des tests unitaires. Les retours fournis par UMLChecker permettent aux étudiants de corriger facilement de nombreuses erreurs basiques. L'enseignant est donc moins sollicité pour des questions élémentaires. Cela renforce l'autonomie des étudiants, les encourageant à progresser plus rapidement. La démarche TDD, en général, et l'utilisation d'UMLChecker, en particulier, accélèrent également le processus d'évaluation, facilitant ainsi un contrôle continu tout au long de la ressource.

La mise en œuvre a montré que cet outil est efficace. ll a été utilisé en BUT1 informatique à Nantes en 2021-2022 et en 2022-2023, de manière ad hoc, sur deux promotions d'environ 100 étudiants découpées en 4 groupes TD et 8 groupes TP. Actuellement, aucune réelle évaluation systématique des apports n'a été réalisées, par exemple, en réalisant des séances "avec/sans" et en comparant le travail réalisé par les étudiants. Notre objectif actuel est d'effectuer une évaluation plus formelle d'UMLChecker afin de perfectionner l'outil. Nous envisageons de soumettre des groupes d'étudiants à des séances de travaux pratiques, avec ou sans cas de tests de conformité, et d'évaluer leurs réalisations. Nous envisageaons aussi d'utiliser ce dispositif afin d'analyser les erreurs faites par les étudiants en fonction du diagramme proposé en utilisant des dispositifs de trace (avec xAPI [5]). Cela nous permettra aussi de voir comment les étudiants utilisent les retours et en quoi les messages proposés leur permettent de progresser.

5. https ://xapi.com/

Références

1. K. Beck. *Test Driven Development : By Example.* Addison-Wesley Professional, 2002.
2. Guy Brousseau. Théorie des situation didactiques. *Recherches en Didactiques des Mathématiques. Paris : La pensée Sauvage*, 1998.
3. Licence professionnelle "bachelor universitaire de technologie" - informatique - programme national 2022, 2022. MAJ 2023 - https ://www.enseignementsup-recherche.gouv.fr/fr.
4. S. Gulati and R. Sharma. *Java Unit Testing with JUnit 5 : Test Driven Development with JUnit 5.* Apress, 2017.
5. R. B. Jackson and J. W. Satzinger. Teaching the Complete Object-oriented Development Cycle, Including OOA and OOD, with UML and the UP. *Inf. Syst. Educ. J*, 1(28) :1–20, 2003.
6. *JUnit 5*, 2023. https ://junit.org/junit5/.
7. P. Leiding and H. Salmela. *IS2020 : A Competency Model for Undergraduate Programs in Information Systems.* Association for Computing Machinery (ACM), Association for Information Systems (AIS), Education SIG of the Association for Information Systems and Computing Academic Professionals (ED-SIG), December 2020.
8. L. Madeyski. *Test-Driven Development.* Springer, 2009.
9. S. Moisan and J.-P. Rigault. Teaching Object-Oriented Modeling and UML to Various Audiences. In *Models in Software Engineering*, volume 6002, pages 40–54. Springer Berlin Heidelberg, 2010.
10. J. Printz and J.-F. Pradat-Peyre. *Pratique des tests logiciels.* Dunod, 2021.
11. R. Rivera-Lopez, E. Rivera-Lopez, and A. Rodriguez-Leon. Another approach for the teaching of the foundations of programming using UML and Java. In *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering.* World Scientific and Engineering Academy and Society, 2009.
12. D. Silva, I. Nunes, and R. Terra. Investigating code quality tools in the context of software engineering education. *Comp Applic In Engineering*, 25(2) :230–241, March 2017.
13. J. Van Eyck, N. Boucké, A. Helleboogh, and T. Holvoet. Using code analysis tools for architectural conformance checking. In *Proceedings of the 6th International Workshop on SHAring and Reusing Architectural Knowledge*, pages 53–54. ACM, May 2011.