Enseigner la rétro-ingénierie, en s'interrogeant sur l'évolution du logiciel : retour d'expériences *

Mireille Blay-Fornarino[1], Sébastien Mosser[1], and Xavier Blanc[2]

[1] Université Nice côte d'Azur
Laboratoire I3S
prenom.nom@unice.fr
[2] Université de Bordeaux,
LABRI
Xavier.Blanc@labri.fr

Résumé

Nous avons expérimenté cette année au niveau M2 un enseignement sur la rétro-ingénierie d'applications logicielles. Dans cet article, nous partageons les objectifs de cet enseignement, explicitons la démarche mise en place et concluons par une rétrospective.

1 Introduction

La maintenance du logiciel est connue pour être la phase la plus coûteuse de tout projet logiciel [3]. Quelles que soient les raisons de cette maintenance adaptative, perfective ou preventive [8], elle requiert de « comprendre » le système à maintenir. Cette étape peut être facilitée en utilisant une approche de rétro-ingenierie (*reverse-engineering*) qui vise à analyser les logiciels pour identifier leurs composants et interrelations et créer des représentations du logiciel sous une forme différente ou à des niveaux d'abstraction plus élevés [2, Chapitre 5]. Cette étape fondamentale repose non seulement sur l'étude des codes, mais également des artefacts adjacents comme la documentation, l'histoire du logiciel, l'expertise des développeurs, etc [4]. Les approches dans ce domaine ont beaucoup progressé ces dernières années avec le développement des projets open source, la démocratisation de plates-formes de développement ouvertes telles que Github et le développement d'approches empiriques et l'utilisation d'outils de fouilles de données (cf. le site web du cours [1]).

Dans ce contexte de mutation des méthodes de développement et de production du logiciel, de remise en question des règles de gestion de l'évolution du logiciel [7], de nouvelles approches de rétro-ingénierie [10] il a été décidé de proposer aux étudiants de M2 de la filière « Architecture Logicielle » un module qui aborde cette vaste problématique. Pour faire face à la complexité du domaine et à la nécessité d'une approche pratique, nous avons donc fait le choix d'un enseignement de la rétro-ingénierie dirigé par des études de cas et plus spécifiquement des questionnements autour de l'évolution du logiciel. Ce faisant, nous amenions les étudiants à s'intéresser à une question liée à la maintenance des codes et dans le même temps, pour répondre à cette question, à utiliser une démarche de rétro-ingénierie.

Dans cet article, nous partageons cette expérience d'enseignement [1]. Par ce retour d'expérience, et les échanges que nous souhaitons occasionner nous avons pour ambition d'améliorer notre pédagogie et de peut-être l'inscrire dans un mouvement plus général en terme d'expérimentations en génie logiciel.

Cet article présente une première partie où nous explicitons les objectifs, la démarche et la mise en œuvre de ce module d'enseignement, puis dans un deuxième temps nous faisons une rétrospective sur cette expérience.

*Merci à Philippe Collet, Naouel Moha, Simon Urli, Yves Roudier pour leur aide dans la mise en œuvre de ce module.

1. Tous les artefacts associés à cet enseignement sont disponibles sur le site du cours [1].

2 Un enseignement de rétro-Ingénierie dirigé par des questionnements

La démarche pour mettre au point ce module a procédé en deux temps. Nous avons d'abord mené une réflexion sur le processus de maintenance de code, sur son impact dans la vie du développeur et ce qu'un tel apprentissage devrait et pourrait apporter à un jeune développeur. Cette première étape a été étayée par un sondage présenté brièvement ci-après et de nombreuses discussions avec des collaborateurs. Dans un deuxième temps, sur la base de cette réflexion et d'expériences passées, nous avons fait le choix d'un focus sur la rétro-ingénierie et d'une approche par étude de cas, présentée en section 2.3.

2.1 Contexte d'enseignement

Ce module d'enseignement prend place au niveau M2 et s'adresse à des étudiants de master et dernière année d'école d'ingénieur, en Sciences Informatiques. Ces étudiants proviennent de formations différentes, mais nous avons mis comme pré-requis une très bonne connaissance de la programmation et des notions liées aux architectures logicielles comme les patrons de conception et UML. Cette année, 23 étudiants ont suivi ce module. Le module a duré 7 semaines avec un créneau de 4h chaque semaine et une évaluation en huitième semaine.

2.2 Choix d'enseignements liés aux résultats du sondage

Le sondage a été diffusé de juillet à octobre 2016. Les résultats du sondage sont accessibles à l'adresse suivante : `https://goo.gl/9FPQHp`. Les répondants au nombre de 217 sont à 82% des développeurs. Nous ne présentons ici que les points qui ont influencé les choix pédagogiques.
Les questions relatives aux outils ont permis de mettre en exergue la quasi absence d'utilisation d'outils pour mener à bien les tâches de maintenance ; seuls des outils du type de SONAR ont été cités. Il nous est donc apparu essentiel d'introduire dans nos enseignements une vision différente de la rétro-ingénierie qui pose les limites des métriques, de leur visualisation et aborde l'existence d'autres formes d'analyse. Par ce sondage, nous avons retrouvé les résultats connus qui montrent une part importante de la maintenance dédiée à l'ajout de fonctionnalités et la correction de bugs. Par contre on note qu'aujourd'hui le support de nouvelles plateformes est bien moins souvent évoqué que par le passé. Nous avons donc plus spécifiquement visé dans nos enseignements les 2 premiers points.
Les deux critères choisis le plus souvent pour expliquer la difficulté de la maintenance sont la taille et la complexité des codes et l'architecture mal adaptée. Il est donc apparu comme important de nous focaliser sur des applications qui présentent ces difficultés.

2.3 Un apprentissage dirigé par des études de cas

La rétro-ingénierie ne peut pas se faire sans objectifs. Trouver un objectif intéressant voire ludique tout en restant réalisable dans un contexte d'enseignement contraint dans le temps est un défi connu. Nous avons fait le choix cette année de laisser les étudiants choisir eux-même leur sujet d'étude, nous explicitons à présent les artefacts utilisés pour conduire cet apprentissage.

2.3.1 Des cours magistraux sous forme de problèmes

Un premier cours a présenté un point de vue général sur la maintenance de code, avec un focus sur la partie rétro-ingénierie. Dans ce cours sont introduits les grands principes de la maintenance de

logiciels [5] ainsi que des approches exploratoires comme présentées dans le livre « Your Code as a Crime Scene » [10] de Adam Tornhill. Ensuite nous avons fait le choix d'interventions sur des sujets particuliers qui mettaient l'accent sur des aspects spécifiques. Ainsi un cours a porté sur l'identification d'anti-patrons et de corrections, en utilisant des outils d'analyse de code par Naouel Moha. Un second cours par Sébastien Mosser a présenté les principes de SPOON [9] pour l'analyse et la transformation de code java en utilisant une approche par méta-modélisation. Puis avec Xavier Blanc les étudiants ont été conduits à s'interroger sur l'« histoire de leur code » et comment les outils pouvaient les aider à mieux comprendre la maintenance du code au quotidien. Enfin le cours de Yves Roudier s'est lui davantage focalisé sur la rétro-ingénierie pour la recherche des failles de sécurités. Ces quelques cours les ont confronté à différentes questions posées par la rétro-ingénierie et aux outils et artefacts différents associés *e.g.* le code, la gestion de versions, la visualisation, les recueils de failles, etc.

2.3.2 Des études de cas dirigées par des interrogations sur les bonnes pratiques du GL

Les lois sur l'évolution du logiciel [7] ont servi de bases pour leur suggérer des pistes de réflexions. Comment « vérifier » la véracité des lois ? Quel est l'impact des changements des processus de développement ? Nous avons proposé aux étudiants de s'interroger à leur tour sur ce type de question, d'élaborer une question, de déterminer les métriques, outils, moyens qui leur permettraient d'y répondre, de poser leurs hypothèses et les limites de leur étude puis d'investiguer. Il pouvait ne pas y avoir de réponse, l'important était la démarche suivie.

Ils ont choisi entre autre de s'intéresser à la loi de Conway [2] dans un contexte Open-Source, à la qualité des codes développées en suivant des approches test en premier (TDD) ou test en dernier (TL), à la corrélation entre des demandes d'utilisateurs et l'évolution d'un jeu (Terasology), à la recherche de dépendances entre composants exprimés dans des langages différents ou utilisant des approches évènementielles.

Les groupes d'étudiants étaient formés de 2 à 4 étudiants. Dans le cas de 4 étudiants, ils devaient partager le problème en deux pour que chaque sous-groupe explore une branche spécifique du même problème, par exemple, une étude des projets en TDD et les autres en TL.

Afin de donner un objectif final à cette étude nous leur avons demandé de travailler à la rédaction d'un chapitre de livre par étude de cas. La rédaction s'est fait en utilisant gitbooks dans la lignée de la démarche suivie par les étudiants de l'université de Delft dans le cours sur les architectures logicielles [3].

Chaque chapitre devait respecter le schéma suivant : *(i)* Problématique sous la forme d'une « question bien posée » et incluant un état de l'art étayant l'opportunité de la question, *(ii)* la démarche proposée pour répondre à la question (métriques, outils, codes choisis et pourquoi, méthode), *(iii)* les résultats brutes (chiffres, visualisations), *(iv)* l'analyse des résultats, *(v)* la conclusion (analyse critique, perspectives). Ce choix suit entre autre les principes énoncés dans [5] avec l'introduction de métriques, leur analyse éventuellement par visualisation puis par raffinement en fonction des points identifiés.

2.3.3 Fertilisations croisées

La largeur du domaine étudié, l'auto-apprentissage qui sous-tend à l'ensemble de ce module, en même temps que les provenances diverses des étudiants étaient une opportunité pour une fertilisation croisée entre les différents groupes d'étudiants. Chaque groupe d'étudiants a proposé des métriques et des outils associés à son étude. Certains pouvaient être réutilisées et les choix des uns et leurs explications pouvaient servir de références à d'autres étudiants. Par exemple, il y eut des discussions sur les notions de complexité des codes ou de qualité. Nous avons donc organisé deux séances où les

2. *les organisations qui définissent des systèmes ... sont contraintes de les produire sous des designs qui sont des copies de la structure de communication de leur organisation* [3].

3. https://www.gitbook.com/book/delftswa/desosa2016/details

étudiants devaient brièvement énoncer l'état d'avancement de leur étude et les outils utilisés, tandis que leurs camarades et un ensemble d'enseignants les interrogeaient sur leur approche. Cela a donné lieu à des évaluations croisées. Les intervenants dans ce module ayant des provenances et compétences différentes, nous avons également fait le choix d'évaluations à plusieurs des exposés et des chapitres, ce qui a permis des retours plus riches.

3 Rétrospective

Nous n'avons pas à ce jour de retours étudiants chiffrés sur lesquels nous appuyer.

3.1 Des projets dirigés ou pas par les étudiants

La formulation d'un problème sous la forme d'une question s'est révélée bien plus difficile que nous ne le pensions a priori. Établir une démarche pour répondre à la question l'a été tout autant. Même si des projets choisis servaient de base d'études, nous attendions des étudiants de réfléchir à la démarche proposée et d'expliciter en quoi celle-ci pourrait être généralisée. Peu de groupes ont réussi cet exercice. L'accompagnement de 7 projets différents a été particulièrement difficile car l'esprit était de les amener à trouver eux-même les éléments et non pas de leur donner une solution. Le fait que nous n'ayons pas fixé les sujets à l'avance rendait l'exercice particulièrement difficile. Les points positifs ont été la proposition de nouveaux outils et la largeur des points de vue abordés, de même que des propositions de sujets intéressants et très différents qui les intéressaient. Nous les avons aidés à construire leur propre projet et cela nous est apparu comme très positif pour certains groupes. D'autre part ils ont utilisé des approches et outils différents dont certains ont été apportés par les étudiants comme Kibana [4] pour la visualisation et la fouille des données, les analyseurs de langage naturel pour l'étude des requêtes utilisateur, etc. Les points négatifs sont liés à la formulation de questions approximatives qui ont donné lieu à des résultats très moyens. Malgré un encadrement qui se voulait adapté à chaque sous-groupe, il faut bien reconnaitre que pour certains le décalage entre des approches plus traditionnelles et cette demande de créativité et d'auto gestion a été difficile à dépasser. L'an prochain nous envisageons de faire des propositions précises de sujets qui pourront servir d'exemples, tout en encourageant les étudiants à faire leur propre proposition, au moins pour les groupes d'étudiants les plus créatifs.

3.2 Des approches étayées ou pas par des articles scientifiques

Pour certains groupes, les étudiants ont eux-mêmes proposés des articles pour étayer leur propos, certains dans un cadre académique d'autres industriel. Chaque article a été validé par un enseignant avant d'être utilisé comme référence. Cependant dans le même temps, certains groupes ont fait fi de la contrainte ou n'ont pas su utiliser les articles pour positionner leur travail. Une synthèse d'un article de recherche leur sera demandée l'an prochain qui devrait leur permettre de mieux appréhender l'intérêt de ces lectures.

3.3 Des cours très influençants

Nous avions fait le choix de peu de cours présentant des problématiques spécifiques après un cours général d'introduction. L'approche a suscité l'intérêt des étudiants pour les quelques cours proposés avec de nombreuses questions aux intervenants, et même des prises de contacts directes parfois un peu débordantes. En cherchant à étayer leurs propres études par les contenus des cours, ils y trouvaient dans l'ensemble plus d'intérêt. Le premier cours a eu un impact très fort et a fortement orienté les choix des

4. https://www.elastic.co/products/kibana

étudiants. En effet, il portait sur la problématique générale et présentait des approches diverses incluant des « fouilles » dirigées non seulement par les codes mais également par l'histoire. Ce dernier point a créé un engouement des étudiants qui se sont majoritairement engagés dans des projets exploitant ce type d'informations. De fait on se pose la question aujourd'hui de les laisser seuls face à des questions, se construire leur propre perspective et ensuite seulement leur donner des éléments de recul. Le danger est la confusion que risque d'occasionner cette approche un peu violente pédagogiquement parlant.

4 Conclusion

L'enseignement de la rétro-ingénierie est une tâche complexe qui recouvre de nombreux aspects. Sur la base des résultats d'un sondage et de l'expérience d'une équipe d'enseignants/chercheurs nous avons choisi une approche dirigée par des études de cas portant essentiellement sur la vérification de loi sur l'évolution des codes. Nous avons eu un résultat particulièrement intéressant sur la comparaison de projets menés en TDD et TL. Ce n'est pas les résultats en eux même que nous défendons car ils mériteraient d'être étayés par des discussions avec les développeurs des projets analysés mais la démarche suivie. Nous avons identifié plusieurs leviers qui nous semblent pouvoir améliorer l'approche l'an prochain. Nous espérons que ces futurs développeurs sauront tirer parti de cet apprentissage pour mieux comprendre l'importance d'un génie logiciel « responsable » [6].

Références

[1] M. Blay-Fornarino. Focus sur la rétro-ingénierie, {http://mireilleblayfornarino.i3s.unice.fr/doku.php?id=teaching:reverse:2016}.

[2] Pierre Bourque and Richard E Fairley, editors. {*SWEBOK*} *: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, CA, version 3. edition, 2014.

[3] Frederick P. Brooks, Jr. *The Mythical Man-month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[4] Elliot Chikofsky and James Cross II. Reverse Engineering and Design Recovery : A Taxonomy. *IEEE Software*, 7(1) :13–17, 1990.

[5] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object Oriented Reengineering Patterns*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[6] Martin Fowler. An appropriate use of metrics {https://martinfowler.com/articles/useOfMetrics.html}, 19 february 2013.

[7] Israel Herraiz, Daniel Rodriguez, Gregorio Robles, and Jesus M Gonzalez-Barahona. The Evolution of the Laws of Software Evolution : A Discussion Based on a Systematic Literature Review. *ACM Computing Surveys*, 46(2) :28 :1—-28 :28, 2013.

[8] Iso. International Standard - ISO/IEC 14764 IEEE Std 14764-2006. *Software Engineering ? Software Life Cycle Processes ? Maintenance, ISO/IEC 14764 :2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, pages 1–46, 2006.

[9] Renaud Pawlak, Martin Monperrus, Nicolas Petitprez, Carlos Noguera, and Lionel Seinturier. Spoon : A library for implementing analyses and transformations of java source code. *Software : Practice and Experience*, 46 :1155–1179, 2015.

[10] A Tornhill. *Your Code as a Crime Scene*. Pragmatic programmers. Pragmatic Bookshelf, 2015.