L'analyse de logiciels, phylogénie et histoire

Adnane Ghannem, Salima Hassaine

Sous la supervision de Yann-Gaël Guéhéneuc et Sylvie Hamel
DIRO, Université de Montréal
{ghannema|hassaisa|guehene|hamelsyl}@iro.umontreal.ca

1 Contexte et problème

D'une part, les programmes industriels actuels nécessitent le développement de logiciels de plus en plus complexes et de très bonne qualité du point de vue de la maintenabilité et de la réutilisabilité. Dans l'industrie, le coût de la maintenance est estimé à 50% et 75% de son coût total [5]. On estime également que plus de la moitié de cette maintenance est consacrée à la compréhension du programme lui-même [2]. D'autre part, le développement des programmes libres peut être réalisé par des milliers de développeurs répartis sur toute la planète. La communauté, qui s'intéresse à la compréhension de programmes, a donc un défi de taille : comment aider les développeurs à saisir toute la complexité de ces programmes pour réaliser leurs changements sans introduire de bogues ou aller à l'encontre des choix de conception ?

2 Solution

L'étude de l'historique et de la phylogénie de programmes sont deux volets de recherche intéressantes permettant d'offrir aux développeurs une vision précise mais simplifiée du programme dans le temps afin d'en faciliter la compréhension, la réutilisabilité dans les règles de l'art et de diminuer les erreurs lors de son évolution [6, 1]. Il a été démontré, dans un travail précédent [3], qu'il est possible de représenter de façon unique un programme sous la forme d'une chaîne de caractères représentant les classes du programme et les relations entre celles-ci. Ainsi, pour étudier l'historique et la phylogénie des programmes, nous proposons d'utiliser des algorithmes développés dans la communauté en bioinformatique et de les adapter/appliquer au problème de l'étude de l'évolution de grands programmes, en deux volets :

Étude de l'historique de programmes qui consiste en l'étude des différentes versions d'un même programme dans le but d'en identifier et d'en comprendre les composantes essentielles. Dans ce volet, l'extraction de patrons récurrents dans l'histoire d'un programme est une direction de recherche intéressante pour répondre, par exemple, aux questions : (1) Quels sont les éléments stables/instables ? (2) Quels éléments changent fréquemment ensembles ? (3) Pourquoi le changement a-t-il été fait ? Notre méthodologie consiste donc d'abord à généraliser l'algorithme vectoriel développé précédemment [3] pour l'alignement global de séquences à un algorithme vectoriel pouvant faire de l'alignement local (ce qui permettra de répondre à la question (1). Ensuite, étant donné les résultats de différences (respectivement de similarités) entre deux versions d'un même programme, on pourra facilement identifier tous les fichiers qui changent (respectivement, ne changent pas), répondant ainsi à la question (2). Finalement, la question (3), il faut pour pouvoir y répondre connaître les patrons récurrents d'évolution (*e.g.,*

patrons de changements) dans l'historique d'un programme. Ces patrons pourront alors former un langage qui inclura l'intention des développeurs et les conséquences de leurs présences.

Étude phylogénique des programmes qui consiste en la création d'algorithmes pour la construction d'un réseau phylogénique des versions d'un programme. Les chaînes de caractères de différentes versions d'un programme vont contenir des différences dues à la recombinaison de certaines parties des chaînes, c'est-à-dire à des changements dans l'architecture et l'implantation des programmes. Premièrement, il faut alors adapter et/ou appliquer des algorithmes efficaces produits dans le domaine de la bio-informatique pour créer ces réseaux en se basant premièrement sur un principe de parcimonie (*i.e.,* si on a une composante A dans une version i d'un programme qui est changée en une composante B dans la version $i+1$, ce changement comptera pour 1 seul changement même si, possiblement, la séquence de changement était plutôt A changé en C puis en B). Cette adaptation, nous permet de d'étudier la similitude entre différents grands programmes, dans le but de dégager les similarités et interconnexions, par exemple, d'en le but d'identifier les parties communes entre programmes pour éventuellement déceler du code dupliqué ou expliquer la filiation entre programmes. Deuxièmement, il faut définir les équivalents informatiques des exons (parties d'un ADN codant une caractéristique de l'organisme) et introns (parties tampons entre deux exons) pour, par exemple, permettre d'identifier les parties stables et instables.

3 Conclusion

L'idée d'utiliser et de généraliser des algorithmes de bioinformatique dans le contexte de génie logiciel est, selon nos connaissances, complètement nouvelle et très prometteuse. Nous voyons également l'impact que nos deux volets de recherche aura sur la compréhension de programmes, la collaboration sur des projets de développement à code source libre ainsi que sur l'identification de code commun entre versions et programmes différents, pour ainsi réduire les coûts de maintenance.

Références

[1] Salah Bouktif, Yann-Gael Gueheneuc, and Giuliano Antoniol. Extracting change-patterns from cvs repositories. In *WCRE '06 : Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 221–230, Washington, DC, USA, 2006. IEEE Computer Society.

[2] T. A. Corbi. Program understanding : challenge for the 1990's. *IBM Syst. J.*, 28(2) :294–306, 1989.

[3] Olivier Kaczor, Yann-Gael Gueheneuc, and Sylvie Hamel. Efficient identification of design patterns with bit-vector algorithm. *csmr*, 0 :175–184, 2006.

[4] Joseph B. Kruskal and Mark Liberman. *The symmetric time-warping problem : From continuous to discrete*. In David Sankoff and Joseph B. Kruskal, editors, in Time Warps, String Edits, and Macromolecules : The Theory and Practice of Sequence Comparison, Addison-Wesley, 1983.

[5] Ian Sommerville. *Software Engineering*. Addison-Wesley, sixth edition, 2000.

[6] Thomas Zimmermann, Peter WeiSSgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, pages 563–572. IEEE Computer Society, May 2004.

Summary

Maintenance of large software systems is a costly activity because their chaotic evolution impedes their comprehension and modification. A study of their evolution could decrease the costs. Our work aims at developing evolution analysis techniques (history and phylogeny), by adapting bioinformatics algorithms.