Appel à Défis pour le Génie de la Programmation et du Logiciel

Vers plus de fiabilité sur les résultats de recherche en génie logiciel

N. Anquetil[1], A. Etien[1], J.-R. Falleri[2], and C. Urtado[3]

[1]Équipe RMoD, Cristal, Inria, Université de Lille, CNRS, Lille
[2]Équipe Progress, LaBRI, CNRS et ENSEIRB, Bordeaux
[3]EuroMov Digital Health in Motion, Univ. Montpellier, IMT Mines Ales, Ales

D'après l'IEEE, le génie logiciel est l'"Application of a systematic, disciplined, quantifiable approach to development, operation and maintenance of a software". Malgré tout, pour mener à bien un développement logiciel, il faut bien souvent effectuer des choix cornéliens. Par exemple quel langage de programmation faut-il choisir ? Comment gérer les branches de son dépôt ? Est-ce que l'on doit écrire le code avant ou après les tests ? De notre point de vue, la communauté de recherche en génie logiciel est aux avant-postes pour répondre à ces questions que se posent fréquemment les développeurs.

A l'origine, plusieurs axiomes utilisés dans la communauté génie logiciel reposaient sur des "mythes" comme le montre les livre de Laurent Bossavit "The Leprechauns of Software Engineering" [5] et de Robert L. Glass "Facts and Fallacies of Software Engineering"[9]. Pour prendre un exemple du livre de Laurent Bossavit, l'adage selon lequel plus un défaut est découvert tard dans le procédé de développement, plus il est cher à corriger [4], semble ne pas reposer sur des données fiables, mais a pourtant été propagé par de nombreux articles de recherche en génie logiciel. (exemple récent : [14]), alors même qu'il a été potentiellement réfuté [13].

Pour pallier ce problème, la mouvance "génie logiciel empirique" s'est donc mise en place dans la communauté de recherche en génie logiciel. L'idée derrière cette dénomination est de renforcer la façon dont sont validées les hypothèses et les résultats obtenus dans les différents domaines du génie logiciel. Ce renforcement passe par l'utilisation de méthodes empiriques qui ont fait leur preuves dans d'autres domaines, comme la médecine ou l'économie, par exemple les études de cas ou les expériences contrôlées, en les adaptant au contexte du génie logiciel, et pourquoi pas en les améliorant. Ce sujet intéresse déjà de nombreuses personnes dans le communauté française, avec le création d'un groupe de travail sur le sujet au sein du GDR-GPL en 2013, et plusieurs thèses soutenues sur le sujet chaque année.

Néanmoins l'obtention de résultats empiriques fiables en génie logiciel n'en est encore qu'à ses balbutiements, tant les facteurs qui peuvent très fortement influencer un résultat sont nombreux (par exemple l'expérience des développeurs ou le domaine applicatif du logiciel considéré). Un exemple particulièrement intéressant de controverse est très certainement la série d'articles et de rebuttal de Baishakhi et al. avec Berger et al [15, 2, 6, 3]. Le sujet initial concerne une étude qui analyse l'association entre les langages de programmation et le nombre

de bugs. La controverse qui s'en suit est très révélatrice car elle montre premièrement que l'obtention de résultats fiables est très complexe, et elle montre dans un second temps que notre communauté de recherche a énormément gagné en maturité sur cet aspect, car c'est la première fois qu'un article qui ne fait qu'essayer de reproduire des résultats passés fait couler autant d'encre.

L'objectif de ce défi est de s'inscrire dans cette mouvance et d'installer au cœur de la communauté française l'envie d'utiliser ces méthodes, tout en essayant de proposer des améliorations de fond sur les bonnes pratiques à suivre. C'est donc un défi très transverse dans son essence car il peut intéresser autant des chercheurs dans la communauté des méthodes formelles que des chercheurs dans le domaine la variabilité logicielle. Dans la suite du document, nous exposons les pistes qui nous semblent actuellement les plus importantes à développer.

Premièrement, utilisons nous les bonnes méthodologies pour évaluer nos résultats ? Dans son tutorial [16], Mary Shaw liste cinq méthodes permettant de valider des résultats de Génie Logiciel : l'analyse, l'évaluation, l'expérience, l'exemple et la persuasion. Concernant l'analyse, l'évaluation et les expériences, nous utilisons bien trop souvent des méthodes statistiques d'analyses très classiques reposant sur un calcul de p-value. Or, ces méthodes font l'objet de nombreuses controverses [10]. Il est donc grand temps de se poser la question de ce qu'on peut faire pour améliorer la confiance que l'on peut avoir en nos résultats, comme par exemple l'utilisation de l'inférence bayésienne [8], ou l'utilisation de méthodes croisées qui mettent en œuvre de l'évaluation qualitative et quantitative [7]. Pourquoi aussi ne pas aussi généraliser l'utilisation de l'enregistrement préalable (séparer la publication du protocole expérimental de l'analyse des résultats) pour éviter les phénomènes de type HARKing [12] ? Une autre question que l'on peut se poser : est-ce que l'exemple ou la persuasion peuvent aussi donner des validations probantes ?

Un des défis inhérent au génie logiciel est que de nombreuses variables confondantes existent au sein d'un projet de développement logiciel. Typiquement, le domaine applicatif, l'équipe de développement, le procédé de développement, le management, sont autant d'éléments qui peuvent influencer les résultats qu'on peut obtenir. Par exemple, la mise en œuvre d'une preuve de code peut avoir un effet énorme sur un logiciel de pilotage automatique d'aéronef, et un effet marginal sur un jeux-vidéo mobile. Cela rend les expériences contrôlées très complexes à monter dans le domaine du génie logiciel car idéalement elles nécessitent souvent de faire venir un nombre très important de développeurs et de faire des mesures sur un nombre très grand de systèmes logiciel, ce qui est impossible en pratique. Il est donc temps de trouver des techniques plus légères qui nous permettent de valider nos résultats, comme par exemple la technique en provenance de la recherche en économie de "Regression on Discontinuities", utilisé par Théo Zimmermann pour valider des résultats sur des changements d'outils et méthodes dans l'écosystème Coq [17]. Il ne faut pas aussi négliger les validations moins généralisables comme une étude de cas bien menée qui peut mettre rapidement en lumière les résultats d'une méthode ou d'un outil dans le contexte où il est censé être utilisé. Il faut aussi favoriser les tentatives de reproduction qui permettent de rendre plus solides nos connaissances [11].

A notre sens, en tant que communauté nous devons fournir un effort important pour mieux extraire les propriétés importantes d'un développement logiciel (domaine applicatif, profil des développeurs, etc.) pour faciliter la description du contexte d'application d'une méthode ou d'un outil dans une validation donnée, et de permettre au lecteur d'estimer lui même si les résultats peuvent avoir du sens dans son contexte.

Deuxièmement, une question nous parait primordiale : nos résultats sont ils assez repro-

ductible ? Dans l'essence même de la méthode empirique, la reproductibilité des résultats est primordiale. Il sera vain d'essayer de donner confiance à un résultat qu'on ne pourra pas reproduire. Il est donc important que les chercheurs de notre communauté visent à faciliter toujours plus cet aspect de leurs travaux. C'est un point qui peut-être très difficile quand on ne suit pas les bonnes pratiques (comme la publication du code ou des données sous licence libre). Il est important aussi de réfléchir à la pérennité des actions mises en œuvre (sera-t-il encore possible de lancer un programme dans 10 ans ?). Dans notre opinion, la plateforme Software Heritage [1] est typiquement une des meilleures solutions actuelle pour pérenniser le code. Il serait donc important de sensibiliser la communauté à son utilisation. Malheureusement il reste encore d'important problèmes. Par exemple que faire quand la machine pour laquelle le code a été écrit n'existe plus ? Dans certains cas il devient impossible de compiler et a fortiori faire tourner les prototypes de recherche. Que pouvons nous faire pour éviter ce genre de problèmes ?

En filigrane, un objectif très important dans ce défi est de continuer à sensibiliser la communauté française sur les méthodes empiriques pour le génie logiciel, de recenser et partager quelles sont les meilleures pratiques, de leur permettre d'identifier un panel de "spécialistes" qu'ils peuvent solliciter facilement quand leur vient le besoin de procéder à une validation empirique des résultats ou des hypothèses qui leurs sont importants.

Références

[1] Jean-François Abramatic, Roberto Di Cosmo, and Stefano Zacchiroli. Building the universal archive of source code. *Commun. ACM*, 61(10) :29–31, 2018.

[2] Emery D. Berger, Celeste Hollenbeck, Petr Maj, Olga Vitek, and Jan Vitek. On the Impact of Programming Languages on Code Quality : A Reproduction Study. *ACM Trans. Program. Lang. Syst.*, 41(4) :21 :1–21 :24, October 2019.

[3] Emery D. Berger, Petr Maj, Olga Vitek, and Jan Vitek. FSE/CACM Rebuttal, 2019.

[4] Barry W Boehm and Kevin J Sullivan. Software economics. *this volume*, 1981.

[5] Laurent Bossavit. *The Leprechauns of Software Engineering*. Lulu. com, 2015.

[6] Prem Devanbu and Vladimir Filkov. On a Reproduction Study Chock-Full of Problems, 2019.

[7] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting Empirical Methods for Software Engineering Research. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer London, London, 2008.

[8] Carlo Alberto Furia, Robert Feldt, and Richard Torkar. Bayesian data analysis in empirical software engineering research. *IEEE Transactions on Software Engineering*, 2019.

[9] Robert L Glass. *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional, 2002.

[10] John P. A. Ioannidis. What Have We (Not) Learnt from Millions of Scientific Papers with P Values ? *The American Statistician*, 73(sup1) :20–25, 2019.

[11] William G Kaelin Jr. Publish houses of brick, not mansions of straw. *Nature News*, 545(7655) :387, 2017.

[12] Norbert L. Kerr. HARKing : Hypothesizing After the Results are Known. *Personality and Social Psychology Review*, 2(3) :196–217, 1998.

[13] Tim Menzies, William Nichols, Forrest Shull, and Lucas Layman. Are Delayed Issues Harder to Resolve ? Revisiting Cost-to-fix of Defects Throughout the Lifecycle. *Empirical Softw. Engg.*, 22(4) :1903–1935, August 2017.

[14] F. Qin, Z. Zheng, Y. Qiao, and K. S. Trivedi. Studying Aging-Related Bug Prediction Using Cross-Project Models. *IEEE Transactions on Reliability*, 68(3) :1134–1153, September 2019.

[15] Baishakhi Ray, Daryl Posnett, Premkumar Devanbu, and Vladimir Filkov. A Large-scale Study of Programming Languages and Code Quality in GitHub. *Commun. ACM*, 60(10) :91–100, September 2017.

[16] Mary Shaw. Writing Good Software Engineering Research Papers : Minitutorial. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 726–736, USA, 2003. IEEE Computer Society. event-place : Portland, Oregon.

[17] Théo Zimmermann and Annalí Casanueva Artís. Impact of switching bug trackers : a case study on a medium-sized open source project. In *ICSME 2019 - International Conference on Software Maintenance and Evolution*, Cleveland, United States, September 2019.