不能完整地描述开发者的工作量[26].他们指出,评估开发者工作量还需要考虑到开发者在查看 20%的代码行数时所查看代码变更的数量.开发者在查看大量的代码变更时,可能需要频繁地在不同代码变更中切换,并且不同的代码变更可能会对不同文件和模块进行修改,因此,审查大量变更要求开发者之间进行更多合作与交流.这种上下文切换和合作交流也会占用开发资源.同时,Huang 等人还指出,如果预测模型将不存在缺陷的代码变更预测为有缺陷,开发者可能就会放弃这个预测模型[26].因此,他们提出两个新指标来对即时缺陷预测进行评估:*PCI*@20%和 *IFA*.*PCI*@20%是指检查 20%代码中包含的代码变更占项目中所有代码变更的比例;*IFA* 是指在找到第 1 个有缺陷变更之前,缺陷预测模型为开发者推荐错误代码变更(预测为有缺陷而实际没有缺陷)的数量.

3 即时缺陷预测技术的关键问题

虽然即时缺陷预测技术近年来获得了大量研究者的关注,取得了较大进展,但仍然存在一些亟待解决的关键问题.本节将从科学问题、技术难点和工程实践这 3 个方面阐述目前即时缺陷预测技术中存在的问题.

3.1 科学问题

3.1.1 缺陷的可解释性

现有的即时缺陷预测技术只对变更存在缺陷的可能性进行预测,而针对所预测缺陷具体是什么,如缺陷类型和位置,目前尚没有相关研究.缺陷类型刻画了缺陷产生的原因和特点[60],缺陷位置指缺陷所在的模块、文件、函数甚至代码行,掌握缺陷类型和位置有助于辅助开发人员快速修复缺陷.尽管目前研究者已提出大量缺陷分类[60−64]和缺陷定位[65−71]技术,针对即时缺陷预测的缺陷分类和缺陷定位,目前尚没有相关研究.其原因在于:现有的缺陷分类和缺陷定位技术均依赖于缺陷具体信息,如缺陷报告或程序执行记录,而即时缺陷预测所预测出的缺陷由于其即时性,缺陷尚未暴露且没有相关缺陷报告或执行记录,因此需要深入分析缺陷引入变更,研究缺陷的可解释性,并进一步研究即时缺陷定位.

3.1.2 缺陷预测结果的实用性

现有的即时缺陷预测技术主要有两种类型的结果:一种是面向分类的预测结果,即每当有新的代码变更提交时,模型预测其为有缺陷或没有预测两个类别[17];另一种是面向优先级排序的结果,即模型对新提交的软件变更集合进行排序,排在列表前面的表示缺陷可能性更高[19,25].面向分类的预测技术常常采用分类器指标进行评估,面向排序的预测技术常常采用工作量感知的指标进行评估.然而,针对软件开发或维护过程中如何有效利用即时缺陷预测结果尚没有统一的观点,因此需要展开深入的实证研究,以调研开发者实际需要哪种类型的预测结果,如何能更有效地利用即时缺陷预测结果等问题.

3.2 技术难点

3.2.1 数据标注

如何准确地标注引入缺陷的软件变更是即时缺陷预测中的技术难点之一.近年来,研究者指出传统 SZZ 算法实现存在噪音.传统 SZZ 实现会将修改空行和注释行以及代码风格的变更、重构类变更误标记为引入缺陷变更[32,34].针对这些噪音,研究者提出了多种改进的 SZZ 实现,然而这些 SZZ 实现并不能完全消除噪音[32−34].例如,Neto 等人提出重构感知 SZZ 避免重构类变更被误标记为引入缺陷,但他们仍然指出,该 SZZ 实现对部分重构类变更仍然无法识别[34].同时,这些工作是基于 Java 代码对 SZZ 实现进行改进.不同编程语言在注释、代码风格和重构等方面存在差异.因此,研究者提出的改进 SZZ 实现在非 Java 项目上面临实际应用上的挑战.

3.2.2 特征提取

特征提取的准确性和多样性对模型的建立有着至关重要的影响.

特征提取准确性的难点在于代码变更的复杂性,主要体现在两个方面:一是代码变更具有时间戳,在计算变更特征时需要考虑软件的动态演化,包括代码演化和开发者演化;二是代码变更提交的复杂性,软件开发和维护常采用代码版本控制系统(例如 git),在项目采用多个分支进行同时开发时,提取特征时对代码变更合并、关联和追溯分析进行综合考虑是一大技术难点.

特征提取多样性的难点在于软件制品的多源异构特性.研究表明,软件缺陷与多种软件制品相关,包括源代码、缺陷报告、测试报告、代码审查、代码静态扫描等,提取多源特征有助于更准确地刻画缺陷,从而建立更准确的预测模型.然而上述软件制品以多源异构的形式存储在不同的软件仓库中,考虑它们之间异构性、关联性是一大技术难点.

3.2.3 模型构建

现有的即时缺陷预测技术在模型构建上存在着不同的策略.研究者使用复杂的有监督机器学习模型,包括集成学习[72]和深度学习[73],有研究者主张使用简单的无监督模型[25].在有监督学习模型中,有研究者使用分类模型[17],有研究者使用回归模型[19].不同的建模技术在模型应用场景、模型输出结果上存在差异,而关于哪种建模技术是更好的即时缺陷预测模型构建技术尚没有统一的结论,因此,在模型构建技术上如何选择合适的建模技术,如何设计更先进的建模技术仍然需要进一步研究.

3.3 工程实践

尽管目前存在的相关工作将即时缺陷预测引入至工业界[18,19],然而将即时缺陷预测在工程实践中进行大规模推广仍然存在以下挑战.

(1) 缺乏准确的数据环境.在即时缺陷预测的数据标注中,识别引入缺陷的软件变更时,依赖于开发人员撰写的变更提交日志(commit log)和缺陷报告.倘若在工程实践过程中这些数据不够准确,将对模型的训练带来巨大挑战.

(2) 缺乏多维的数据特征.由于软件缺陷的复杂性,其产生机制与多源异构的软件制品相关,如源代码、缺陷报告、测试报告、代码审查报告、代码静态扫描等,在工程实践中,软件制品的多样性和准确性将直接影响模型效果.

(3) 缺乏统一的评估方法.由于目前针对即时缺陷预测技术模型评估存在着指标多样化问题,在工程实践中也将面临如何更客观地评估模型这一问题.因此,需要更多工作关注即时缺陷预测的工程实践,将工程实践与理论研究相结合,将实证研究和方法研究相结合,进一步完善即时缺陷预测技术.

4 即时缺陷预测的未来展望

针对上一节所总结的关键问题,本节围绕数据标注、特征提取、模型构建和工程实践等 4 个方面展望即时缺陷预测研究的未来趋势.

4.1 加强噪音数据的处理,提出更准确的数据标注方法

实证研究表明,即时缺陷预测研究中所使用的标注数据是存在噪音的.Bachmann 等人的研究表明,开发者可能不会为修复缺陷的代码变更记录其对应的缺陷报告 ID,从而导致部分修复缺陷的代码变更无法被识别出来[74].这将导致这些修复缺陷的代码变更所对应的引入缺陷的代码变更也无法识别出来.Da Costa 等人在他们的工作中提出了一种框架用于评估 SZZ 算法产生数据的质量,他们使用该框架发现,使用 SZZ 算法所标注的数据中存在大量噪音[33].另外,实证研究还表明,有些缺陷在引入数年后才会被发现[75].这意味着在对即时缺陷预测技术进行实际应用时,对于近期产生的代码变更,其中某些缺陷引入变更可能还没有被修复,因此无法被 SZZ 算法识别,这也会造成噪音数据.处理这些噪音数据,并且减小噪音数据对即时缺陷预测模型的影响,将进一步提升即时缺陷预测技术的实用性.

4.2 综合考虑多源软件制品,提取多维特征

现代大型软件项目都会使用多个系统对软件开发和维护中产生的数据进行存储和管理,例如代码版本控制系统(如 git)、缺陷跟踪系统(如 bugzilla)、代码审查系统(如 gerrit)等.这些不同的系统中存储着不同的软件制品,现有的大部分即时缺陷预测研究都只考虑了从代码版本控制系统中提取特征.而研究表明,软件缺陷的产生是一个复杂过程,并与多种软件制品相关,包括源代码、缺陷报告、测试报告、代码审查、代码静态扫描等,综合考虑多源异构的软件制品,有助于更立体地刻画缺陷产生机制,建立更准确的缺陷预测模型.例如,有研究

人员已经发现,结合代码审查系统,从代码审查的角度提取新的特征,有助于增强现有即时缺陷预测模型的预测效果[28].因此,综合考虑多源软件制品,提取多维特征来增强特征表征能力,将是即时缺陷预测技术的一个发展方向.

4.3 研究更先进的建模技术,取得建模技术上的突破

近年来,深度学习成为机器学习的热点研究领域,被大量应用到如图像处理、语音识别等研究中[76−78].与传统的机器学习技术相比,研究者发现深度学习在这些领域的应用可以取得更好的性能.Yang 等人首次在即时缺陷预测中应用深度学习技术[73],他们使用深度学习方法对初始的变更特征进行整合,从而生成更加复杂的特征,然后基于这些生成的特征构建分类器对有缺陷变更进行预测.他们发现,使用深度学习技术显著提升了即时缺陷预测模型的性能.但是他们仅使用了深度学习中的深度信念网络[79]技术用于整合特征.而将目前机器学习领域大量研究与应用的技术,如卷积神经网络(convolutional neural network,简称 CNN)[80]应用到即时缺陷预测技术中,将有可能进一步提升即时缺陷预测技术的性能.如何有效地将这些技术应用到即时缺陷预测技术中,需要未来工作进一步分析与研究.

4.4 解决实践中的工程问题,推动即时缺陷预测的广泛应用

在工程实践中,实现自动化、大规模的即时缺陷预测,在数据环境准备、特征提取和模型构建等方面仍然面临许多技术难点.克服现有难点主要从以下几方面展开:(1) 积极研究大规模代码变更数据自动化标注方法,尽快构建面向即时缺陷预测的大数据环境及其演进方法;(2) 解决在工程实践中多源异构软件制品的关联问题,提出代码变更的多维特征提取方法;(3) 结合更先进的机器学习技术,提出适应于工程实践中面向大规模代码变更数据的建模方法;(4) 建立即时缺陷预测的标准评估体系,推动即时缺陷预测向更统一更准确的方向发展;(5) 为了适应云计算、移动互联网及人工智能的快速发展,有针对性地研究面向不同软件类型的即时缺陷预测技术.

5 总 结

近年来,即时缺陷预测技术由于其即时性、细粒度和可追溯的优势,成为了软件缺陷预测领域的研究热点.本文围绕即时缺陷预测的数据标注、特征提取、模型构建及模型评估等方面,梳理并总结了当前研究进展.基于当前进展分析,本文总结了当前即时缺陷预测面临的关键问题及未来发展趋势.主要工作总结如下.

(1) 针对数据标注,本文详细归纳了不同数据标注方法的提出背景及其优缺点;针对特征提取,本文分类并详细介绍了不同维度的变更特征;针对模型构建,本文归类了现有模型构建技术,包括有监督和无监督建模技术,其中,有监督建模技术又分为同项目建模和跨项目建模;针对模型评估,本文总结了现有不同类型的验证方法和评估指标.

(2) 本文从科学问题、技术难点和工程实践这 3 个角度总结了当前即时缺陷预测面临的关键问题.

(3) 本文围绕数据标注、特征提取、模型构建和工程实践这 4 个方面展望了未来即时缺陷预测技术的发展趋势.

References:

[1] Zubrow D. IEEE Standard Classification for Software Anomalies. IEEE Std, 2009. 1−23.

[2] Newman M. Software errors cost us economy 59.5 billion annually: NIST assesses technical needs of industry to improve software-testing. 2002. http://www.abeacha.com/NIST_press_release_bugs_cost.htm

[3] Marks L, Zou Y, Hassan AE. Studying the fix-time for bugs in large open source projects. In: Proc. of the 7th Int'l Conf. on Predictive Models in Software Engineering. New York: ACM Press, 2011. No.11.

[4] LaToza TD, Venolia G, DeLine R. Maintaining mental models: A study of developer work habits. In: Proc. of the 28th Int'l Conf. on Software Engineering. New York: ACM Press, 2006. 492−501.

[5] Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. The impact of automated parameter optimization on defect prediction models. IEEE Trans. on Software Engineering, 2018. [doi: 10.1109/TSE.2018.2794977]

[6] Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Trans. on Software Engineering, 2019,45(2):111−147.

[7] Mende T, Koschke R. Revisiting the evaluation of defect predictionmodels. In: Proc. of the 5th Int'l Conf. on Predictor Models in Software Engineering. New York: ACM Press, 2009. No.7.

[8] Xia X, Shihab E, Kamei Y, Lo D, Wang X. Predicting crashing releases of mobile applications. In: Proc. of the 10th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. New York: ACM Press, 2016. No.29.

[9] Song Q, Jia Z, Shepperd M, Ying S, Liu J. A general software defect-proneness prediction framework. IEEE Trans. on Software Engineering, 2011,37(3):356−370.

[10] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans. on Software Engineering, 2008,34(4):485−496.

[11] Hassan AE. Predicting faults using the complexity of code changes. In: Proc. of the 31st Int'l Conf. on Software Engineering. Washington: IEEE, 2009. 78−88.

[12] Menzies T, Butcher A, Cok D, Marcus A, Layman L, Shull F, Turhan B, Zimmermann T. Local versus global lessons for defect prediction and effort estimation. IEEE Trans. on Software Engineering, 2013,39(6):822−834.

[13] Xia X, Lo D, Nagappan N, Wang X. Hydra: Massively compositional model for cross-project defect prediction. IEEE Trans. on Software Engineering, 2016,42(10):977−998.

[14] Nam J, Fu W, Kim S, Menzies T, Tan L. Heterogeneous defect prediction. IEEE Trans. on Software Engineering, 2018,44(9): 874−896.

[15] Zhang F, Hassan AE, McIntosh S, Zou Y. The use of summation to aggregate software metrics hinders the performance of defect prediction models. IEEE Trans. Software Engineering, 2017,43(5):476−491.

[16] Koru AG, Zhang D, El Emam K, Liu H. An investigation into the functional form of the size-defect relationship for software modules. IEEE Trans. on Software Engineering, 2009,35(2):293−304.

[17] Kim S, Jr. Whitehead EJ, Zhang Y. Classifying software changes: Clean or buggy? IEEE Trans. on Software Engineering, 2008, 34(2):181−196.

[18] Shihab E, Hassan AE, Adams B, Jiang ZM. An industrial study on the risk of software changes. In: Proc. of the 20th Int'l Symp. on the Foundations of Software Engineering. New York: ACM Press, 2012. No.62.

[19] Kamei Y, Shihab E, Adams B, Hassan AE, Mockus A, Sinha A, Ubayashi N. A large-scale empirical study of just-in-time quality assurance. IEEE Trans. on Software Engineering, 2013,39(6):757−773.

[20] Jiang T, Tan L, Kim S. Personalized defect prediction. In: Proc. of the 28th Int'l Conf. on Automated Software Engineering. Washington: IEEE, 2013. 279−289.

[21] Shivaji S, Whitehead EJ, Akella R, Kim S. Reducing features to improve code change-based bug prediction. IEEE Trans. on Software Engineering, 2013,39(4):552−569.

[22] Fukushima T, Kamei Y, McIntosh S, Yamashita K, Ubayashi N. An empirical study of just-in-time defect prediction using cross-project models. In: Proc. of the 11th Working Conf. on Mining Software Repositories. New York: ACM Press, 2014. 172−181.

[23] Tan M, Tan L, Dara S, Mayeux C. Online defect prediction for imbalanced data. In: Proc. of the 37th Int'l Conf. on Software Engineering. Washington: IEEE, 2015. 99−108.

[24] Kamei Y, Fukushima T, McIntosh S, Yamashita K, Ubayashi N, Hassan AE. Studying just-in-time defect predictionusing cross-project models. Empirical Software Engineering, 2016,21(5):2072−2106.

[25] Yang Y, Zhou Y, Liu J, Zhao Y, Lu H, Xu L, Xu B, Leung H. Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. In: Proc. of the 24th Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2016. 157−168.

[26] Huang Q, Xia X, Lo D. Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction. In: Proc. of the 33rd Int'l Conf. on Software Maintenance and Evolution. Washington: IEEE, 2017. 159−170.

[27] Fu W, Menzies T. Revisiting unsupervised learning for defect prediction. In: Proc. of the 25th Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2017. 72−83.

[28] McIntosh S, Kamei Y. Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. IEEE Trans. on Software Engineering, 2018,44(5):412−428.

[29] Mockus A, Weiss DM. Predicting risk of software changes. Bell Labs Technical Journal, 2000,5(2):169−180.

[30] Śliwerski J, Zimmermann T, Zeller A. When do changes induce fixes? In: Proc. of the 2nd Working Conf. on Mining Software Repositories. New York: ACM Press, 2005. 24−28.

[31] Kamei Y, Shihab E. Defect prediction: Accomplishments and future challenges. In: Proc. of the 23rd Int'l Conf. on Software Analysis, Evolution, and Reengineering. Washington: IEEE, 2016. 33−45.

[32] Kim SH, Zimmermann T, Pan K, Jr. Whitehead EJ. Automatic identification of bug-introducing changes. In: Proc. of the 21st Int'l Conf. on Automated Software Engineering. Washington: IEEE, 2006. 81−90.

[33] Da Costa DA, McIntosh S, Shang W, Kulesza U, Coelho R, Hassan AE. A framework for evaluating the results of the SZZ approach for identifying bug-introducing changes. IEEE Trans. on Software Engineering, 2017,43(7):641−657.

[34] Neto EC, Da Costa DA, Kulesza U. The impact of refactoring changes on the SZZ algorithm: An empirical study. In: Proc. of the 25th Int'l Conf. on Software Analysis, Evolution and Reengineering. Washington: IEEE, 2018. 380−390.

[35] Zimmermann T, Kim S, Zeller A, Jr. Whitehead EJ. Mining version archives for co-changed lines. In: Proc.of the 3rd Working Conf. of Mining Software Repositories. New York: ACM Press, 2006. 72−75.

[36] Fowler M, Beck K, Brant J, Opdyke W, Roberts D. Refactoring: Improving the Design of Existing Code. Boston: Addison-Wesley Professional, 1999.

[37] Silva D, Valente MT. RefDiff: Detecting refactorings in version histories. In: Proc. of the 14th Int'l Conf. on Mining Software Repositories. Washington: IEEE, 2017. 269−279.

[38] Basili VR, Perricone BT. Software errors and complexity: An empirical investigation. Communications of the ACM, 1984,27(1): 42−52.

[39] Hatton L. Reexamining the fault density component size connection. IEEE Software, 1997,14(2):89−97.

[40] Mockus A, Votta LG. Identifying reasons for software changes using historic databases. In: Proc. of the 16th Int'l Conf. on Software Maintenance. Washington: IEEE, 2000. 120−130.

[41] Schneidewind NF, Hoffmann HM. An experiment in software error data collection and analysis. IEEE Trans. on Software Engineering, 1979,5(3):276−286.

[42] Khoshgoftaar TM, Allen EB. Ordering fault-prone software modules. Software Quality Journal, 2003,11(1):19−37.

[43] Gyimothy T, Ferenc R, Siket I. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. on Software Engineering, 2005,31(10):897−910.

[44] Sammut C, Webb GI. Encyclopedia of Machine Learning and Data Mining. 2nd ed., Boston: Springer-Verlag, 2017. 314−315.

[45] Moser R, Pedrycz W, Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proc. of the 30th Int'l Conf. on Software Engineering. New York: ACM Press, 2008. 181−190.

[46] Fan Y, Xia X, Lo D, Li S. Early prediction of merged code changes to prioritize reviewing tasks. Empirical Software Engineering, 2018,23(6):3346−3393.

[47] Nagappan N, Ball T. Use of relative code churn measures to predict system defect density. In: Proc. of the 27th Int'l Conf. on Software Engineering. New York: ACM Press, 2005. 284−292.

[48] D'Ambros M, Lanza M, Robbes R. An extensive comparison of bug prediction approaches. In: Proc. of the 7th Working Conf. on Mining Software Repositories. Washington: IEEE, 2010. 31−41.

[49] Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: Proc. of the 35th Int'l Conf. on Software Engineering. Washington: IEEE, 2013. 392−401.

[50] Eyolfson J, Tan L, Lam P. Do time of day and developer experience affect commit bugginess? In: Proc. of the 8th Working Conf. on Mining Software Repositories. New York: ACM Press, 2011. 153−162.

[51] Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. IEEE Trans. on Software Engineering, 2007,32(1):2−13.

[52] Card DN, Agresti WW. Measuring software design complexity. Journal of Systems and Software, 1988,8(3):185−197.

[53] Zhang Y, Jin R, Zhou ZH. Understanding bag-of-words model: A statistical framework. Int'l Journal of Machine Learning and Cybernetics, 2010,1(1-4):43−52.

[54] Jones J. Abstract syntax tree implementation idioms. In: Proc. of the 10th Conf. on Pattern Languages of Programs. Hillside, 2003. https://www.hillside.net/plop/plop2003/papers.html

[55] Matsumoto S, Kamei Y, Monden A, Matsumoto KI, Nakamura M. An analysis of developer metrics for fault prediction. In: Proc. of the 6th Int'l Conf. on Predictive Models in Software Engineering. New York: ACM Press, 2010. 18.

[56] Bird C, Nagappan N, Murphy B, Gall H, Devanbu P. Don't touch my code! Examining the effects of ownership on software quality. In: Proc. of the 19th Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2011. 4−14.

[57] Thongtanunam P, McIntosh S, Hassan AE, Iida H. Investigating code review practices in defective files: An empirical study of the QT system. In: Proc. of the 12th Int'l Conf. on Mining Software Repositories. Washington: IEEE, 2015. 168−179.

[58] Mitchell TM. Machine Learning. Burr Ridge: McGraw Hill, 1997.

[59] Mende T, Koschke R. Effort-aware defect prediction models. In: Proc. of the 14th European Conf. on Software Maintenanceand Reengineering. Washington: IEEE, 2010. 107−116.

[60] Lyu MR. Handbook of Software Reliability Engineering. IEEE Computer Society Press, 1996. 359−399.

[61] Thung F, Lo D, Jiang L. Automatic defect categorization. In: Proc of the 19th Working Conf. on Reverse Engineering. Washington: IEEE, 2012. 205−214.

[62] Thung F, Le XB, Lo D. Active semi-supervised defect categorization. In: Proc. of the 23rd Int'l Conf. on Program Comprehension. Washington: IEEE, 2015. 60−70.

[63] Huang L, Ng V, Persing I, Chen M, Li Z, Geng R, Tian J. AutoODC: Automated generation of orthogonal defect classifications. Automated Software Engineering, 2015,22(1):3−46.

[64] Hernández-González J, Rodriguez D, Inza I, Harrison R, Lozano JA. Learning to classify software defects from crowds: A novel approach. Applied Soft Computing, 2018,62:579−591.

[65] Lukins SK, Kraft NA, Etzkorn LH. Bug localization using latent dirichlet allocation. Information and Software Technology, 2010, 52(9):972−990.

[66] Zhou J, Zhang H, Lo D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In: Proc. of the 34th Int'l Conf. on Software Engineering. Washington: IEEE, 2012. 14−24.

[67] Xie X, Chen TY, Kuo FC, Xu B. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Trans. on Software Engineering and Methodology, 2013,22(4):Article No.31.

[68] Kim D, Tao Y, Kim S, Zeller A. Where should we fix this bug? A two-phase recommendation model. IEEE Trans. on Software Engineering, 2013,39(11):1597−1610.

[69] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Bug localization with combination of deep learning and information retrieval. In: Proc. ofthe 25th Int'l Conf. on Program Comprehension. Washington: IEEE, 2017. 218−229.

[70] Youm KC, Ahn J, Lee E. Improved bug localization based on code change histories and bug reports. Information and Software Technology, 2017,82:177−192.

[71] Hoang TV, Oentaryo RJ, Le TD, Lo D. Network-clustered multi-modal bug localization. IEEE Trans. on Software Engineering, 2018. [doi: 10.1109/TSE.2018.2810892]

[72] Yang X, Lo D, Xia X, Sun J. Tlel: A two-layer ensemble learning approach for just-in-time defect prediction. Information and Software Technology, 2017,87:206−220.

[73] Yang X, Lo D, Xia X, Zhang Y, Sun J. Deep learning for just-in-time defect prediction. In: Proc. of the 15th Int'l Conf. on Software Quality, Reliability and Security. Washington: IEEE, 2015. 17−26.

[74] Bachmann A, Bird C, Rahman F, Devanbu P, Bernstein A. The missing links: Bugs and bug-fix commits. In: Proc. of the 18th Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2010. 97−106.

[75] Graves TL, Karr AF, Marron JS, Siy H. Predicting fault incidence using software change history. IEEE Trans. on Software Engineering, 2000,26(7):653−661.

[76] Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science, 2006,313(5786):504−507.

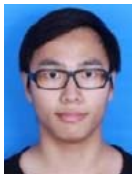
[77] Hinton GE. Learning multiple layers of representation. Trends in Cognitive Sciences, 2007,11(10):428−434.

[78] Deng L, Li J, Huang JT, Yao K, Yu D, Seide F, Seltzer M, Zweig G, He X, Williams J. Recent advances in deep learning for speech research at Microsoft. In: Proc. of the 38th Int'l Conf. on Acoustics, Speech and Signal Processing. Washington: IEEE, 2013. 8604−8608.

[79] Hinton GE. Deep belief networks. Scholarpedia, 2009,4(5):Article No.5947.

[80] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. In: Proc. of the Advancesin Neural Information Processing Systems. 2012. 1097−1105.

蔡亮(1976－),男,江西九江人,博士,副教授,CCF 高级会员,主要研究领域为计算机应用.



范元瑞(1994－),男,博士生,CCF 学生会员,主要研究领域为软件仓库挖掘,经验软件工程.

鄢萌(1989－),男,博士,助理研究员,CCF 专业会员,主要研究领域为智能软件工程,软件仓库挖掘,软件维护与演化.

夏鑫(1986－),男,博士,讲师,博士生导师,CCF 专业会员,主要研究领域为软件仓库挖掘,经验软件工程.