基于图的方法还有 GPLAG[67],它也使用了同构的程序依赖图的子图的对比算法,能够在小数据集上有着不错的效率,但是随着数据集的变大,计算复杂度将会呈指数增长;Higo 等人的工作[68,71]使用了简化的方法,能够检测非相邻的代码克隆对,比一般的基于程序依赖图的方法更快.

近期使用了混合方法的工作有 Saini 等人提出的方法 Oreo[70].Oreo 是一个代码克隆检测的框架,在它提出的流程中,使用了基于指标度量的方法在预处理过程中进行了初步筛选与分组,然后进一步从函数中抽取语义,并使用了基于哈希的方法进一步筛选,能够对类型 1 和类型 2 的代码克隆进行较好的覆盖,在流程的最后加入了深度学习的方法,提升了对类型 3 代码克隆的检测能力.其中,输入数据以对(pair)的形式输入是代码克隆检测的一个技术特点,Oreo 在深度学习结构中使用的 Siamese 结构[72]很好地解决了代码克隆输入的对称性问题,即(CF1,CF2)与(CF2,CF1)在作为输入时是等效的,值得借鉴.

混合方法还有 Funaro 等人的工作[69],综合了基于文本的方法和符号的方法,能够做到对类型 3 克隆的检测;Agrawal[63]等人的工作综合了基于文本和指标度量的方法,但是只能检测 C 的代码克隆.

综上所述,基于语义的方法能够在深入的层次利用源代码的信息,从而检测代码克隆,这些方法充分利用了源代码不同于一般的自然语言文本的特性,包括了结构、顺序以及特殊的语法等信息,也能够获得一定程度的语义信息.但其还具有如下缺点.

(1) 基于图的技术需要一个程序依赖图的生成器.而对于不同的语言会产生技术隔阂.

(2) 基于图的图匹配检测技术,计算的开销非常大.

(3) 基于混合技术表征源代码语义步骤繁杂,难以部署.

4 克隆检测评估方法

不同层次表征方式有各自的特征与优缺点,因此也需要科学、系统的方法对其进行验证和评估.为了评估代码克隆对软件系统的影响,在早期研究中,主要报告检测到的代码克隆在软件系统中的比例[2,23,73–75].这样的实证研究能够为后续研究提供证据支撑,启发思维.在提出克隆检测方法的过程中,为了验证和对比模型效果,研究者需要对检测模型进行实验验证.其中,评价指标中重要的精确度和召回率需要已知克隆对这样的先验知识,然而,面临未知软件系统却难以预先获得代码克隆准确数量.评估代码克隆检测的关键是收集客观、有效的先验数据集,因此,研究者们使用数据标注方法和数据生成方法获取代码克隆先验知识以评估模型.

4.1 标注数据收集方法

一些研究者们在提出方法时会自己搜集若干代码库[30,33,76–78],利用上下文,修改历史和人工检查等标注代码克隆,并用于评估自己的检测模型,然而不同的系统有不同的规模和结构等,这样的缺陷是代码库规模不够大[54],难以形成统一的评价基础.为了解决这样的问题,Svajlenko 等人提出了 BigCloneBench[54].BigCloneBench 是一个 Java 代码集,它包含了从类型 1 到类型 4 的大量的人工标注的克隆对,包含 10 个功能,耗费了 8 个专家 216 个小时标注 600 万对代码克隆和 26 万对负样本.很多后续的工作基于这个数据集评估它们自己的方法[39,51,52],BigCloneBench 推进了代码克隆检测的研究.但是这样的数据集有两个缺陷:第一,它只有 10 个功能,这不符合真实的软件系统的情况;第二,这个数据集的构建方法是基于启发式的搜索加上人工标注,而启发式搜索在某种程度上限制了代码克隆的模式.

4.2 评估数据生成方法

在代码克隆检测中,召回率需要已知软件系统中的所有代码克隆,然而这难以在每一个新的软件系统中实现.Roy 等人借鉴软件测试中的变异测试[79],提出了一种变异插入的测试代码克隆的方法[46],其思想是在一段源码中插入一段人工代码,从而人为制造出不同类型的代码克隆对,以此作为测试集进而评估代码克隆检测工具的效果.这样的数据生成方法能够有效获得召回率,比如,Wang 等人在评估 CCAligner 针对大间隔代码克隆的召回率时使用了这种方法[51].虽然这样操作部分解决了代码克隆评价指标的问题,但是人工构造的代码克隆对不是真实数据集,难以从科学和工程上证明有效性,因此只能作为一种辅助测试方式.

5 代码克隆检测关键问题与解决思路

虽然代码克隆检测的研究已经有超过 20 年的历史,获得了非常多的成果,但是现有的研究与工业界的期望还有一定的距离,这是作为研究者需要正视的问题.本节首先讨论代码克隆检测研究中的关键问题,然后讨论针对这些问题的解决思路.

5.1 关键问题

虽然代码克隆检测以往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、实用性、技术难点和工程实践 3 个方面阐述目前代码克隆研究中存在的问题.

5.1.1 代码克隆检测研究的科学问题

代码克隆检测研究的科学问题分为 3 个方面.

(1) 代码克隆产生的归因分析与检测技术的有机结合.即分析所检测到的代码克隆产生的具体原因,以及研究如何进一步提高代码克隆检测的准确性相关研究,从历史角度研究了代码演进过程,从克隆族谱的角度研究了代码克隆的形成过程[6],以及形成后的更替与传播[78],然而现有归因分析仅仅对代码克隆的影响进行了定性研究[16].进一步地,研究者需要针对代码克隆形成原因,设计相关克隆检测技术,提高克隆代码检测技术对克隆代码的预防能力和时效性.

(2) 代码克隆检测中数据标注的准确性.准确的有标注的代码克隆数据对于代码克隆检测的研究至关重要,是检测模型训练、评估及技术对比的基准.虽然,类型 1 和类型 2 代码克隆因为定义具体而易于准确标注,然而,类型 3 和类型 4 代码克隆却难以准确标注,特别是没有具体定义的类型 4,一般是指功能相似的异构代码(即结构不相同的代码)[4,33],如何对异构代码进行标注是代码克隆数据标注中的一大难点.现有的 BigCloneBench 对类型 4 代码克隆的标注采用人工审查代码的方法[54],而如果从功能和需求出发,结合代码功能或需求描述文档,将有助于提高代码克隆标注的准确性.例如,在针对文件级克隆进行标注时,可参考需求文档中的功能描述;在针对方法级或片段级克隆进行标注时,可参考代码注释中的功能描述.

(3) 代码克隆检测的可拓展性.如今在代码大数据[80]的环境下,软件系统的规模日益变大,这也给代码克隆检测带来了新的挑战.近期的研究表明,一些早期的代码克隆检测技术[38,42]已经难以胜任千万行级别甚至上亿行级别的代码检测任务[41,51],因此,未来的代码克隆检测技术研究不仅要考虑精确度和召回率等检测指标,还需提升对大规模代码检测的能力.此外,代码增加与迭代速度也日益变快,比如开源仓库 Github 中每天都会有成千上万的贡献者贡献代码,在这样的情况下,如果每次都进行全量的代码克隆检测将花费较高的代价[68,81,82].因此,在未来的研究中,代码克隆检测的可拓展性,即能否将检测范围拓展到大规模代码仓库中,能否在高频率增量代码的场景下达到开销与性能的平衡,是研究者需要考虑的问题.

5.1.2 克隆研究的实用性

现有代码克隆研究主要在于以克隆对或者克隆类的形式给出结果,不同的研究给出了不同的粒度属性.克隆对或者克隆类仅仅只有源代码的指向,没有更进一步的分析与展示;在研究工作中选择的粒度也仅仅根据现有数据的方便程度而非工程中的实用程度.针对软件开发或者维护过程中如何有效利用代码克隆检测的结果尚未有统一的观点,因此,需要展开深入的实证研究,结合开发过程,以调研开发者实际需要哪种类型的代码克隆结果,如何能够更加有效地利用代码克隆检测结果等问题.

5.1.3 技术难点

如何更准确地标注代码克隆对是代码克隆检测中的技术难点之一.一方面,虽然有变异插入的标注方法可以大量生成代码克隆对,但是这种方法难以保证在真实的代码库中奏效;另一方面,虽然有研究者花费人力标注了大量的数据集,但是这样的数据集有其本身的局限性,在这个基础上设计的模型,特别是机器学习模型,一旦用到一个新的软件系统中,难以表现出良好的泛化性能.因此,研究者们在针对新进项目获取标注数据时面临着实际应用上的挑战.

源代码的表征方法也是一个技术难点,从文本到词汇、语法、语义,对于源代码信息的利用程度会逐渐增

高,但是带来的副作用是对于语言的独立性逐渐下降,对于如解释器等的额外工具的需求也会逐渐增加,大大提高了算法开发与实际应用的门槛.所以考量多个维度,在适当的层次抽取源代码,考虑它们之间的易用性、关联性也是一大技术难点.

现有代码克隆检测技术在模型构建上存在着不同的策略:在源代码的表征方法方面,基于文本、词汇、语法、语义等方式有多种选择;预处理可以进行不同的抉择,而只做简单的消除或者复杂变形;模型中相似度的计算则受到前面阶段选择的局限;最后结果的呈现也有克隆对克隆类,甚至考虑重新设计可视化界面等多种选择.但是正因为选择太多,所以需要权衡各种策略的利弊,这也给模型的设计增添了困难.因此,针对场景,选择合适的模型是代码克隆检测设计的一大难点.最后,工业界对于类型 4 有着较大的需求,但是现有的工具难以满足这一需求,因此也是需要研究突破的方向.

5.2 解决思路

针对上一节所总结的关键问题,本节围绕数据标注、表征方法、模型构建和工程实践 4 个方面,阐述问题可能得以解决的思路和研究的未来发展趋势.

5.2.1 综合考虑多源软件制品,提出更准确的数据标注方法

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)、代码审查系统(如 Gerrit)等.这些不同系统中存储着不同的软件制品,代码克隆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、需求报告、代码审查、代码静态扫描等,现在还缺乏这方面的研究.综合考虑多源异构的软件制品,有助于更立体地获得真实软件系统中标注的代码克隆数据.同时也应该注意,从多个源头的软件制品获取代码克隆也会引入噪音,比如从需求映射到代码中的噪音问题[83]、代码静态扫描中的误报问题[84]、代码注释与代码的不匹配问题等[85],需要谨慎甄别,以保证标注数据的准确性不受干扰.

5.2.2 综合考虑实际需求,选择或提出更加合适的代码表征方法

在自然语言处理领域,表征学习是一个被关注的热点,同时在软件工程领域,如何将源代码进行适当的表征也是大量工作的基础,从而引起了研究者们的关注[86]:将源代码进行适当的表征,就可以增强对代码克隆的检测能力.与此同时,也要结合已有的表征方法,对优缺点进行权衡,比如基于语义表征代码中基于图的表征方式,早期利用图表征源代码进行代码克隆检测的研究使用了传统的图匹配算法[36,64,68],算法的可移植性和可拓展性都比较差,但是如果考虑图嵌入[87]技术,用嵌入向量的相似比较完成代码克隆检测,既利用了以图为基础的代码中的语义信息,又能够兼顾算法的可用性.虽然基于图的表征方式中图的定义与构建都需要许多研究工作来探索[67,86],但依然是一个可以尝试的方向.因此,使用针对源代码的方法,提取多维特征来增强源代码的表征能力,将是代码克隆研究的一个发展方向.

5.2.3 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.相比传统的机器学习技术,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[39,52].将目前机器学习领域大量研究与应用的技术,如递归神经网络应用到代码克隆检测技术中,将有可能进一步提升代码克隆研究的性能.在现有的工作中,深度学习已经展现了一定的能力,例如 CCLearner 以基于符号的方式学习代码克隆[52],CDLH 利用了源代码的抽象语法树结构[39],Oreo 提出的深度学习框架解决了代码克隆输入的对称问题,并能在初步检测的基础上进一步提高检测性能[70],这些方法都在对类型 3 和类型 4 的代码克隆检测中取得了一定成效,但是由于标注数据集本身限制的问题,例如被广泛使用的 BigCloneBench,只标注了 10 个功能的代码克隆[54],不足以满足现实需求.要更好地研究深度学习模型,一方面,需要大量准确的标注数据,虽然研究者提出的变异测试方法[46]可以生成大量的训练数据,但是真实、准确的标注代码克隆也是不可或缺的.另一方面,代码表征的复杂性需要被充分探索,相关研究使用到了词汇[52]和语法[39]的方式用于研究,本文归纳的基于文本、词汇、语法、语义的表征方式都可以作为输入应用于深度学习的检测中,如何有效地将这些表征技术应用到代码克隆检测中需要未来工作进一步分析与研究.

5.2.4 知识图谱的技术也是解决类型 4 克隆检测的手段之一

对于类型 4 的代码克隆来说,功能相同但是结构不同是其主要的特点,这个特点背后的逻辑是它们都拥有相同的业务性或者系统性的知识,如果能够建立起一个软件系统的知识图谱,就能够帮助跨过符合类型 4 定义的代码克隆之间天然的语义鸿沟,自上而下地检测编码方式不同但是实现了相同功能或业务的不同粒度的源代码.比如说,有研究将 Java 的 API 知识引入了代码检索领域[88],其主要思想是利用额外的 API 文档作为知识库提高检索的准确率,类似地,借助构建的或已有的外部知识,可以增强对源代码的表征能力,提高对类型 4 的代码克隆检测能力.解决实践中的工程问题,推动代码克隆的广泛应用.

在目前的技术中,代码克隆检测技术能够对类型 1 到类型 3 的代码克隆的检测有不错的效果,但却并不能完全应用到工业界之中.首先,这些工具有自己的局限性,不能完全满足第 5.2 节中提到的特性.其次,代码克隆的相关问题在工业界的应用场景中需要更多的实践才能凸显其价值,比如 Dang 等人[20]评估了 XIAO[34]这一工具在微软不断迭代取得的成功:XIAO 的主要成就在于集成到了 Visual Studio 并且在微软开发团队的日常运作中被广泛使用.从中我们可以得到两个方面的启示:(1) 技术方面,在开发技术原型时要结合初步的调查,考虑到在实际中它们的使用情况(即使这样的设想有时候不如预期),第二则是要考虑交互问题,第三是要考虑模型的效率和可靠性.(2) 社会方面,首先,对于目标的用户要有内部知情人来帮助交互,从而能够协调两边的信息;第二要能够真诚地与生产团队交互;第三是要有良好的反馈机制,帮助产品的迭代与优化;第四是要有高效的协作处理过程;最后则是要能够积极、主动地去寻找、拓展产品的领域.除此之外,一个科研原型的落地有两种模式:pull 模式和 push 模式,pull 模式就是根据已有的问题来进行科研并提出解决方案;而 push 模式则是根据科研问题和初步调查,结合科研直觉开发出原型工具,并尝试将原型进行“推销”,在与早期使用者的交互中获得更多反馈并加以改进.

6 总 结

代码克隆对软件系统的开发、维护产生一定程度的影响,其中负面影响包括降低软件稳定性、造成代码库的冗余和软件缺陷的传播.学术界对代码克隆的研究有超过 20 年的历史,但是尚有许多问题,特别是代码克隆的检测问题需要研究.本文基于当前研究进展,重新梳理归纳了代码克隆检测研究,也总结讨论了当前代码克隆研究面临的关键问题及未来发展趋势.主要工作总结如下:(1) 从源代码表征方式这种不同于以往研究该问题的角度阐述并归类了现有的克隆检测方法;(2) 总结了模型评估中使用的实验验证方法与性能评估指标;(3) 从科学性、实用性和技术难点 3 个方面归纳总结了代码克隆研究的关键问题,围绕数据标注、表征方法、模型构建和工程实践 4 个方面,阐述了问题的可能解决思路和研究的未来发展趋势.

References:

[1] Kamiya T, Kusumoto S, Inoue K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. on Software Engineering, 2002,28(7):654–670.

[2] Roy CK, Cordy JR. A survey on software clone detection research. Queen’s School of Computing TR, 2007,541(115):64–68.

[3] Rieger M, Ducasse S, Lanza M. Insights into system-wide code duplication. In: Proc. of the 11th Working Conf. on Reverse Engineering. IEEE, 2004. 100–109.

[4] Sheneamer A, Kalita J. A survey of software clone detection techniques. Int’l Journal of Computer Applications, 2016,137(10): 1–21.

[5] Chen W-K, Li B, Gupta R. Code compaction of matching single-entry multiple-exit regions. In: Proc. of the Int’l Static Analysis Symp. Springer-Verlag, 2003. 401–417.

[6] Kim M, Sazawal V, Notkin D, Murphy G. An empirical study of code clone genealogies. ACM SIGSOFT Software Engineering Notes, 2005,30:187–196.

[7] Aversano L, Cerulo L, Di Penta M. How clones are maintained: An empirical study. In: Proc. of the European Conf. on Software Maintenance and Reengineering. IEEE, 2007. 81–90.

[8] Koschke R. Survey of research on software clones. In: Proc. of the Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2007.

[9] Mondal M, Rahman MS, Saha RK, Roy CK, Krinke J, Schneider KA. An empirical study of the impacts of clones in software maintenance. In: Proc. of the 19th IEEE Int'l Conf. on Program Comprehension. IEEE, 2011. 242–245.

[10] Rattan D, Bhatia R, Singh M. Software clone detection: A systematic review. Information and Software Technology, 2013,55(7): 1165–1199.

[11] Patenaude J-F, Merlo E, Dagenais M, Laguë B. Extending software quality assessment techniques to Java systems. In: Proc. of the 7th Int'l Workshop on Program Comprehension. IEEE, 1999. 49–56.

[12] Hotta K, Sano Y, Higo Y, Kusumoto S. Is duplicate code more frequently modified than non-duplicate code in software evolution: An empirical study on open source software. In: Proc. of the Joint ERCIM Workshop on Software Evolution (EVOL) and Int'l Workshop on Principles of Software Evolution (IWPSE). ACM, 2010. 73–82.

[13] Lozano A, Wermelinger M, Nuseibeh B. Evaluating the harmfulness of cloning: A change based experiment. In: Proc. of the 4th Int'l Workshop on Mining Software Repositories. IEEE Computer Society, 2007. 18.

[14] Krinke J. A study of consistent and inconsistent changes to code clones. In: Proc. of the 14th Working Conf. on Reverse Engineering (WCRE 2007). IEEE, 2007. 170–178.

[15] Kapser CJ, Godfrey MW. "Cloning considered harmful" considered harmful: Patterns of cloning in software. Empirical Software Engineering, 2008,13(6):645.

[16] Mondal M, Rahman MS, Roy CK, Schneider KA. Is cloned code really stable. Empirical Software Engineering, 2018,23(2): 693–770.

[17] Juergens E, Deissenboeck F, Hummel B, Wagner S. Do code clones matter. In: Proc. of the 31st IEEE Int'l Conf. on Software Engineering. IEEE, 2009. 485–495.

[18] Mondal M, Roy CK, Schneider KA. Dispersion of changes in cloned and non-cloned code. In: Proc. of the 6th Int'l Workshop on Software Clones. IEEE Press, 2012. 29–35.

[19] Lozano A, Wermelinger M. Tracking clones' imprint. In: Proc. of the 4th Int'l Workshop on Software Clones. ACM, 2010. 65–72.

[20] Dang Y, Zhang D, Ge S, Huang R, Chu C, Xie T. Transferring code-clone detection and analysis to practice. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering in Software Engineering in Practice Track (ICSE-SEIP). 2017. 53–62.

[21] 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.

[22] Goto A, Yoshida N, Ioka M, Choi E, Inoue K. How to extract differences from similar programs: A cohesion metric approach. In: Proc. of the 7th Int'l Workshop on Software Clones. IEEE Press, 2013. 23–29.

[23] Roy CK, Zibran MF, Koschke R. The vision of software clone management: Past, present, and future. In: Proc. of the Software Evolution Week—IEEE Conf. on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE). 2014. 18–33.

[24] Fontana FA, Zanoni M, Ranchetti A, Ranchetti D. Software clone detection and refactoring. Int'l Scholarly Research Notices, 2013.

[25] Yuan Y, Guo Y. CMCD: Count matrix based code clone detection. In: Proc. of the 18th Asia Pacific Software Engineering Conf. (APSEC). IEEE, 2011. 250–257.

[26] Prechelt L, Malpohl G, Philippsen M. Finding plagiarisms among a set of programs with JPLAG. Journal of Universal Computer Science, 2002,8(11):1016.

[27] Baker BS. On finding duplication and near-duplication in large software systems. In: Proc. of the 2nd Working Conf. on Reverse Engineering. IEEE, 1995. 86–95.

[28] Tsantalis N, Mazinanian D, Krishnan GP. Assessing the refactorability of software clones. IEEE Trans. on Software Engineering, 2015,41(11):1055−1090.

[29] Tsantalis N, Mazinanian D, Rostami R. Clone refactoring with Lambda expressions. In: Proc. of the 39th Int'l Conf. on Software Engineering. IEEE Press, 2017. 60−70.

[30] Meng N, Hua L, Kim M, McKinley KS. Does automated refactoring obviate systematic editing. In: Proc. of the 37th Int'l Conf. on Software Engineering, Vol. 1. IEEE Press, 2015. 392–402.

[31] Walenstein A, Lakhotia A. The software similarity problem in malware analysis. In: Proc. of the Schloss Dagstuhl- Leibniz-Zentrum für Informatik. 2007.

[32] Kontogiannis K, Galler M, DeMori R. Detecting code similarity using patterns. In: Proc. of the Working Notes of the 3rd Workshop on AI and Software Engineering. 1995. 6.

[33] Bellon S, Koschke R, Antoniol G, Krinke J, Merlo E. Comparison and evaluation of clone detection tools. IEEE Trans. on Software Engineering, 2007,33(9).

[34] Dang Y, Zhang D, Ge S, Chu C, Qiu Y, Xie T. XIAO: Tuning code clones at hands of engineers in practice. In: Proc. of the 28th Annual Computer Security Applications Conf. ACM, 2012. 369–378.

[35] Baxter ID, Yahin A, Moura L, Sant'Anna M, Bier L. Clone detection using abstract syntax trees. In: Proc. of the Int'l Conf. on Software Maintenance. IEEE, 1998. 368–377.

[36] Krinke J. Identifying similar code with program dependence graphs. In: Proc. of the 8th Working Conf. on Reverse Engineering. IEEE, 2001. 301–309.

[37] Li Z, Lu S, Myagmar S, Zhou Y. CP-miner: Finding copy-paste and related bugs in large-scale software code. IEEE Trans. on Software Engineering, 2006,32(3):176–192.

[38] Roy CK, Cordy JR. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In: Proc. of the 16th IEEE Int'l Conf. on Program Comprehension. IEEE, 2008. 172–181.

[39] Wei H, Li M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code. In: Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence. AAAI Press, 2017. 3034–3040.

[40] Wang W, Godfrey MW. Recommending clones for refactoring using design, context, and history. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME). 2014. 331–340.

[41] Sajnani H, Saini V, Svajlenko J, Roy CK, Lopes CV. SourcererCC: Scaling code clone detection to big-code. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). 2016. 1157–1168.

[42] Jiang L, Misherghi G, Su Z, Glondu S. Deckard: Scalable and accurate tree-based detection of code clones. In: Proc. of the 29th Int'l Conf. on Software Engineering. IEEE Computer Society, 2007. 96–105.

[43] Hotta K, Yang J, Higo Y, Kusumoto S. How accurate is coarse-grained clone detection: Comparision with fine-grained detectors. In: Proc. of the Electronic Communication of the European Association of Software Science and Technology. 2014. 63.

[44] Ducasse S, Rieger M, Demeyer S. A language independent approach for detecting duplicated code. In: Proc. of the IEEE Int'l Conf. on Software Maintenance. IEEE, 1999. 109–118.

[45] Lee S, Jeong I. SDD: High performance code clone detection system for large scale source code. In: Proc. of the Companion to the 20th Annual ACM SIGPLAN Conf. on Object-oriented Programming, Systems, Languages, and Applications. ACM, 2005. 140–141.

[46] Roy CK, Cordy JR. A mutation/injection-based automatic framework for evaluating code clone detection tools. In: Proc. of the IEEE Int'l Conf. on Software Testing, Verification, and Validation Workshops. 2009. 157–166.

[47] Livieri S, Higo Y, Matushita M, Inoue K. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder. In: Proc. of the 29th Int'l Conf. on Software Engineering. 2007. 106–115.

[48] Yuan Y, Guo Y. Boreas: An accurate and scalable token-based approach to code clone detection. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2012. 286–289.

[49] Murakami H, Hotta K, Higo Y, Igaki H, Kusumoto S. Folding repeated instructions for improving token-based code clone detection. In: Proc. of the 12th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM). IEEE, 2012. 64–73.

[50] Murakami H, Hotta K, Higo Y, Igaki H, Kusumoto S. Gapped code clone detection with lightweight source code analysis. In: Proc. of the 21st Int'l Conf. on Program Comprehension (ICPC). IEEE, 2013. 93–102.

[51] Wang P. CCAligner: A token based large-gap clone detector. In: Proc. of the 40th Int'l Conf. on Software Engineering. ACM, 2018. 1066–1077.

[52] Li L, Feng H, Zhuang W, Meng N, Ryder B. CCLearner: A deep learning-based clone detection approach. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2017. 249–260.

[53] Yan X, Han J, Afshar R. CloSpan: Mining: Closed sequential patterns in large datasets. In: Proc. of the 2003 SIAM Int'l Conf. on Data Mining. SIAM, IEEE, 2003. 166–177.

[54] Svajlenko J, Roy CK. Evaluating clone detection tools with BigCloneBench. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2015. 131–140.

[55] AMoZ J. Identification of common molecular subsequences. Journal of Molecular Biology, 1981,147(1):195−197.

[56] Mayrand J, Leblanc C, Merlo E. Experiment on the automatic detection of function clones in a software system using metrics. In: Proc. of the 1996 Int'l Conf. on Software Maintenance. 1996. 244.

[57] Wahler V, Seipel D, Wolff J, Fischer G. Clone detection in source code by frequent itemset techniques. In: Proc. of the 4th IEEE Int'l Workshop on Source Code Analysis and Manipulation. IEEE, 2004. 128–135.

[58] White M, Tufano M, Vendome C, Poshyvanyk D. Deep learning code fragments for code clone detection. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. 2016. 87–98.

[59] Kontogiannis KA, DeMori R, Merlo E, Galler M, Bernstein M. Pattern matching for clone and concept detection. Automated Software Engineering, 1996,3(1-2):77–108.

[60] Kodhai E, Kanmani S, Kamatchi A, Radhika R, Saranya BV. Detection of type-1 and type-2 code clones using textual analysis and metrics. In: Proc. of the Int'l Conf. on Recent Trends in Information, Telecommunication and Computing. IEEE, 2010. 241–243.

[61] Abd-El-Hafiz SK. A metrics-based data mining approach for software clone detection. In: Proc. of the 36th IEEE Annual Computer Software and Applications Conf. IEEE, 2012. 35–41.

[62] Raheja K, Tekchandani R. An emerging approach towards code clone detection: Metric based approach on byte code. Int'l Journal of Advanced Research in Computer Science and Software Engineering, 2013,3(5).

[63] Agrawal A, Yadav SK. A hybrid-token and textual based approach to find similar code segments. In: Proc. of the 4th Int'l Conf. on Computing, Communications and Networking Technologies (ICCCNT). IEEE, 2013. 1–4.

[64] Komondoor R, Horwitz S. Using slicing to identify duplication in source code. In: Proc. of the Int'l Static Analysis Symp. Springer-Verlag, 2001. 40–56.

[65] Hummel B, Juergens E, Heinemann L, Conradt M. Index-based code clone detection: incremental, distributed, scalable. In: Proc. of the 2010 IEEE Int'l Conf. on Software Maintenance (ICSM). IEEE, 2010. 1–9.

[66] Rivest R. The MD5 message-digest algorithm. 1992. https://tools.ietf.org/html/rfc1321

[67] Liu C, Chen C, Han J, Yu PS. GPLAG: Detection of software plagiarism by program dependence graph analysis. In: Proc. of the 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. ACM, 2006. 872–881.

[68] Higo Y, Yasushi U, Nishino M, Kusumoto S. Incremental code clone detection: A PDG-based approach. In: Proc. of the 18th Working Conf. on Reverse Engineering (WCRE). IEEE, 2011. 3–12.

[69] Funaro M, Braga D, Campi A, Ghezzi C. A hybrid approach (syntactic and textual) to clone detection. In: Proc. of the 4th Int'l Workshop on Software Clones. ACM, 2010. 79–80.

[70] Saini V, Farmahinifarahani F, Lu Y, Baldi P, Lopes C. OREO: Detection of clones in the twilight zone. In: Proc. of the 2018 ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2018. 354–365.

[71] Higo Y, Sawa K, Kusumoto S. Problematic code clones identification using multiple detection results. In: Proc. of the 16th Asia-Pacific Software Engineering Conf. IEEE, 2009. 365–372.

[72] Baldi P, Chauvin Y. Neural networks for fingerprint recognition. Neural Computation, 1993,5(3):402–418.

[73] Zibran MF, Saha RK, Asaduzzaman M, Roy CK. Analyzing and forecasting near-miss clones in evolving software: An empirical study. In: Proc. of the 16th IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS). IEEE, 2011. 295–304.

[74] Gabel M, Jiang L, Su Z. Scalable detection of semantic clones. In: Proc. of the 30th Int'l Conf. on Software Engineering. ACM, 2008. 321–330.

[75] Saha RK, Asaduzzaman M, Zibran MF, Roy CK, Schneider KA. Evaluating code clone genealogies at release level: An empirical study. In: Proc. of the 10th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation. IEEE, 2010. 87–96.

[76] Rysselberghe FV, Demeyer S. Evaluating clone detection techniques from a refactoring perspective. In: Proc. of the 19th IEEE Int'l Conf. on Automated Software Engineering. IEEE Computer Society, 2004. 336–339.

[77] Shafieian S, Zou Y. Comparison of clone detection techniques. Technical Report, Queen's University, 2012. https://www.researchgate.net/publication/267840728_Comparison_of_Clone_Detection_Techniques

[78] Rahman MS, Roy CK. A change-type based empirical study on the stability of cloned code. In: Proc. of the 14th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM). IEEE, 2014. 31–40.

[79] Andrews JH, Briand LC, Labiche Y. Is mutation an appropriate tool for testing experiments. In: Proc. of the 27th Int'l Conf. on Software Engineering. IEEE, 2005. 402–411.

[80] Allamanis M, Barr ET, Devanbu P, Sutton C. A survey of machine learning for big code and naturalness. ACM Computing Surveys (CSUR), 2018,51(4):81.

[81] Göde N, Koschke R. Incremental clone detection. In: Proc. of the 13th European Conf. on Software Maintenance and Reengineering. IEEE, 2009. 219–228.

[82] Nguyen TT, Nguyen HA, Al-Kofahi JM, Pham NH, Nguyen TN. Scalable and incremental clone detection for evolving software. In: Proc. of the 25th IEEE Int'l Conf. on Software Maintenance. IEEE, 2009. 491–494.

[83] Gotel OC, Finkelstein CW. An analysis of the requirements traceability problem. In: Proc. of the 1st IEEE Int'l Conf. on Requirements Engineering. IEEE, 1994. 94–101.

[84] Heckman S, Williams L. A systematic literature review of actionable alert identification techniques for automated static code analysis. Information and Software Technology, 2011,53(4):363–387.

[85] Tan L, Yuan D, Krishna G, Zhou Y. /* icomment: Bugs or bad comments /*. ACM SIGOPS Operating Systems Review, 2007,41: 145–158.

[86] Allamanis M, Khademi M, Brockschmidt M. Learning to represent programs with graphs. arXiv Preprint arXiv: 1711.00740, 2017.

[87] Yan S, Xu D, Zhang B, Zhang H-J, Yang Q, Lin S. Graph embedding and extensions: A general framework for dimensionality reduction. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2007,29(1):40–51.

[88] Lin Z, Zou Y, Zhao J, Xie B. Improving software text retrieval using conceptual knowledge in source code. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. IEEE, 2017. 123–134.

陈秋远(1995－),男,四川成都人,博士生,主要研究领域为智能软件工程,软件仓库挖掘.

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.

李善平(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布式计算,软件工程,操作系统内核.

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.