软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(8):2294−2305 [doi: 10.13328/j.cnki.jos.005523]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向软件安全性缺陷的开发者推荐方法*

孙小兵[1,2], 周 澄[1], 杨 辉[1], 李 斌[1]

[1](扬州大学 信息工程学院,江苏 扬州 225127)

[2](上海市数据科学重点实验室(复旦大学),上海 201203)

通讯作者: 孙小兵, E-mail: xbsun@yzu.edu.cn; 李斌, E-mail: lb@yzu.edu.cn

摘 要: 软件开发与维护过程中常会出现一些安全性缺陷,这些安全性缺陷会给软件和用户带来很大的风险.安全性缺陷在修复过程中,其修复级别和质量要求往往高于一般性的缺陷,因此,推荐出富有安全性经验的开发者及时、有效地修复这些安全性缺陷非常重要.现有的开发者推荐技术在推荐开发者时仅仅考虑了开发者的历史开发内容,很少考虑到开发人员的安全性缺陷修复经验和修复质量等因素,所以这些技术不适用于安全性缺陷的开发者推荐.针对安全性缺陷的修复,提出了一种有效的软件开发者推荐方法 SecDR.SecDR 在推荐开发者时不仅考虑了开发者的历史开发内容(与安全性相关),还分析了开发者的修复质量和历史修复缺陷的复杂度等因素.此外,SecDR 还实现了开发者的多经验级别推荐:推荐初级开发者修复简单的安全性缺陷、高级开发者修复复杂的安全性缺陷.在 3 个开源项目(Mozilla,Libgdx,ElasticSearch)上分别对 SecDR 推荐开发者进行有效性验证.对比实验表明,SecDR 针对安全性缺陷推荐开发者相比于其他方法(如 DR_PSF)的推荐精度平均高出 19%~42%.另外,实验对比了 SecDR 与实际开发人员的分配情况,结果显示,SecDR 可以更好地规避不合理的软件开发者的推荐.

关键词: 安全性缺陷;开发者推荐;缺陷库;缺陷分配;软件维护

中图法分类号: TP311

中文引用格式: 孙小兵,周澄,杨辉,李斌.面向软件安全性缺陷的开发者推荐方法.软件学报,2018,29(8):2294−2305. http://www.jos.org.cn/1000-9825/5523.htm

英文引用格式: Sun XB, Zhou C, Yang H, Li B. Developer recommendation for software security bugs. Ruan Jian Xue Bao/ Journal of Software, 2018,29(8):2294−2305 (in Chinese). http://www.jos.org.cn/1000-9825/5523.htm

Developer Recommendation for Software Security Bugs

SUN Xiao-Bing[1,2], ZHOU Cheng[1], YANG Hui[1], LI Bin[1]

[1](School of Information Engineering, Yangzhou University, Yangzhou 225127, China)

[2](Shanghai Key Laboratory of Data Science (Fudan University), Shanghai 201203, China)

Abstract: Security bugs are commonly emerged bugs during the software development and maintenance, which cause security risks during software deployment. Security bugs need to be fixed with high quality and patched faster than other types of bugs. Recommending developers to fix security bugs is one of the important tasks during the security bug fixing process. Some developer recommendation techniques have been proposed to fix the bugs, but most of these techniques did not recommend developers considering their security

* 基金项目: 国家自然科学基金(61402396, 61472344, 61611540347); 计算机软件新技术国家重点实验室(南京大学)开放课题(KFKT2018B12); 江苏省青蓝工程; 中国博士后科学基金(2015M571489); 扬州市自然科学基金(YZ2017113)

Foundation item: National Natural Science Foundation of China (61402396, 61472344, 61611540347); Open Funds of State Key Laboratory for Novel Software Technology (Nanjing University) (KFKT2018B12); Jiangsu Qin Lan Project; China Postdoctoral Science Foundation (2015M571489); Natural Science Foundation of Yangzhou City (YZ2017113)

本文由数据驱动的软件智能化开发方法与技术专题特约编辑谢冰教授、魏峻研究员、彭鑫教授、孙海龙副教授推荐.

收稿时间: 2017-07-17; 修改时间: 2017-09-28, 2017-12-22, 2018-01-12; 采用时间: 2018-01-24; jos 在线出版时间: 2018-03-13

CNKI 网络优先出版: 2018-03-13 17:18:04, http://kns.cnki.net/kcms/detail/11.2560.TP.20180313.1717.006.html

experience and bug fixing quality. In this paper, an approach, SecDR (security developer recommendation), is proposed to recommend developers by considering the historical data on the quality and complexity of their security bug fixes. In addition, SecDR recommends junior developers for simple bugs, and recommends senior developers for complex bugs. An empirical study on three open source subjects (Mozilla, Libgdx and ElasticSearch) are conducted to evaluate the effectiveness of SecDR. In this study, SecDR is also compared with the state-of-art developer recommendation technique, DR_PSF, to evaluate the effectiveness of developer recommendation. Results show that the accuracy of SecDR is improved over DR_PSF with gain values ranging from 19% to 42%. Moreover, the results of SecDR is also compared with actual developer allocation, and results show that SecDR can effectively recommend developers, which is even better than the developer allocation in the real bug assignment environment.

Key words: security bug; developer recommendation; bug repository; bug assignment; software maintenance

计算机软件开发与使用的过程中,经常出现一些安全性缺陷[1],导致软件系统产生漏洞.黑客通过漏洞恶意攻击他人计算机,如“勒索病毒”等的蔓延,造成了大量的数据泄露,给用户和软企双方带来巨大的利益损失[2,3].因此,安全性缺陷受到软件开发者的广泛关注.如何及时有效地修复安全性缺陷,已经成为工业界和学术界共同关注的问题[3,4].

相关研究结果表明,安全性缺陷的修复级别、修复复杂度、质量和时效性要求往往比一般性缺陷要高[2].例如,在 firefox 项目中,安全性缺陷的修复速度比性能性缺陷快 2.8 倍,被 reopen 的可能性却是性能性缺陷的 2.5 倍、其他类型缺陷的 4.5 倍[2].在缺陷修复分配时,安全性缺陷的分配难度更大,往往由于不能很好地及时分配给合适的开发者而面临重新分配.据统计,安全性缺陷的再分配次数是一般性缺陷的 3.5 倍[2].同时,安全性缺陷也是开发者修复历史中相对较少的缺陷类型.表 1 中给出了 Mozilla 项目中开发者安全性缺陷修复数量分布情况,修复数量少于 10 个的开发者有 195 位,占总人数的 86%;只修复过 1 个的开发者有 88 位,占总人数的 39%.绝大部分开发者修复安全性缺陷的经验匮乏.

Table 1 Distribution of the number of developers and their fixing of security bugs in the Mozilla project

表 1 Mozilla 项目中开发者安全性缺陷修复数量分布

缺陷数量	≥100	50~100	10~50	<10	=1
开发者数量	2	3	27	195	88

目前,很多相关工作都在研究如何推荐合适的开发者解决软件中新出现的缺陷.例如,shokripour 等人提出了 Time-Text-Based 方法[5],该方法利用主题词语和时间因素预测系统中的开发人员对缺陷的熟悉程度,从而推荐出一个最合适的开发者.这些推荐技术都可称为泛推荐技术,主要思想是:将新缺陷的描述信息与历史修复记录相匹配,遵循匹配值最高推荐原则,广泛面向软件缺陷管理库中的所有缺陷进行推荐,缺失了特定类型缺陷推荐的针对性.推荐结果不可避免地偏向于历史开发数据相对丰富的开发者,而这些开发者却不一定拥有丰富的安全性知识和较高的安全性缺陷修复技能,泛推荐技术并不完全适用于安全性缺陷[6,7].

本文针对软件项目中安全性缺陷的修复,提出一种新的开发者推荐方法——SecDR(security developer recommendation).SecDR 采用分级推荐机制,不仅考虑了开发者与安全性缺陷相关的历史开发内容,还考虑了开发者的修复质量和历史修复缺陷的复杂度.首先,SecDR 预测安全性缺陷的修复复杂度等级,综合评估开发者的历史修复经验预测开发者的修复等级;再对推荐的开发者进行排序,排序时主要考虑开发者的经验等级、修复质量和与新缺陷相关的安全性知识匹配度等因素,并给出一个开发者推荐列表.SecDR 不仅可以有效地推荐出开发经验丰富的高级开发者解决难度较大的软件安全性缺陷,还可以推荐出开发经验较少的初级开发者解决比较简单的安全性缺陷,从而实现开发者的多经验级别推荐.

1 面向安全性缺陷的开发者推荐技术

SecDR 包括 6 个步骤,如图 1 所示.首先,SecDR 提取软件缺陷库中的安全性缺陷,构成独立的安全性缺陷库.当出现新缺陷(下文特指新的安全性缺陷),将其描述与安全性缺陷库历史信息进行匹配,查找相关历史安全性缺陷,同时对新缺陷的修复复杂度进行预测;另一方面,配出的缺陷开发者的经验等级、安全性缺陷修复质量和

历史修复安全性缺陷的复杂度等因素,综合推荐出一个合适的开发者推荐列表.

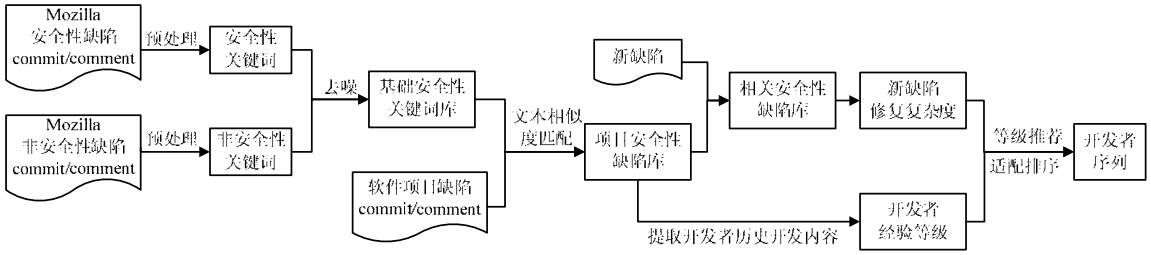

Fig.1 Process of software developer recommendation

图 1 SecDR 软件开发者推荐流程图

1.1 提取安全性缺陷

SecDR 主要针对安全性缺陷推荐富有安全性缺陷相关经验的开发者.然而,大部分开源系统中并没有将安全性缺陷独立管理,因此,本文首先识别出缺陷库中的安全性缺陷,创建独立安全性缺陷库作为 SecDR 的训练数据.文中识别安全性缺陷的技术主要借鉴 Gegick 等人所提出的基于文本挖掘的识别技术[1].如图 1“提取安全性缺陷库”模块所示,本文主要采用开源项目(如 Mozilla(https://www.mozilla.org/en-US/security/advisories/))中所有缺陷对应的 commit 库和 comment 数据;抽取关键词,构建安全性关键词词库;最后,根据安全性关键词词库提取项目中与安全性相关的缺陷组成一个新的安全性缺陷库.

1.1.1 软件缺陷库关键词分析

本文使用自然语言处理技术(NLP)对软件缺陷库进行关键词分析.关键处理步骤如下[8].

(1) 分词.根据骆驼式命名法(camel-case)以及单词的下划线分割单词.比如,“TestCase”或者”test_case”划分后的结果是“test”和“case”.

(2) 停用词去除.比如 the、by、on、and、no 等,这些停用词的移除可以降低语义技术中的噪声数据.

(3) 词根还原.为了减少词汇量,将剩下来的单词进行词根还原.其中,还原算法依据 Porter stemmer 算法[9],例如,“searching”还原后的词根为“search”.

(4) 同义词/近义词扩展.为了解决自然语言中的语义多义性,利用 WordNet 英文词典扩展名词和动词的同义词和近义词[10],例如,“evaluate”是“test”的同义词.

最终,Mozilla 项目中现有的安全性关键词和非安全性关键词被提取出来,主要是一些具有实际意义的动词和名词,下述步骤将利用这些关键词进行文本相似度计算.

1.1.2 安全性关键词提取与分析

上一步骤中,根据 Mozilla 项目中的安全性缺陷和非安全性缺陷进行关键词提取分别获得安全性关键词库和非安全性关键词库.但在安全性关键词中会有很多与安全性不相关的冗余关键词,如表 2 所示为安全性关键词词库中词频排序前 15 的关键词列表,可见 call,code 等与安全性不相关的关键词.

Table 2 List of security bug related keywords in the Mozilla project

表 2 Mozilla 项目中安全性缺陷相关的关键词列表

1	crash	6	object	11	window
2	test	7	regress	12	load
3	script	8	check	13	buffer
4	assert	9	code	14	over
5	frame	10	execute	15	call

为了进一步删除冗余关键词,首先在安全性缺陷库和非安全性缺陷库中分别提取等量的缺陷(1 000 个缺陷),对这些缺陷进行关键词提取,然后计算每个单词出现的次数,从而得到一个键值对{*key*:*value*},*key* 代表关键词,*value* 代表关键词在 1 000 个缺陷中出现的次数.最后计算相同关键词 *value* 的差值,提取差值为正的所有关

键词组成一个安全性关键词库.

1.1.3 安全性缺陷库提取

提取安全性关键词库后,根据这个词库对项目中的缺陷库筛选一些与安全性相关的缺陷,构建独立安全性缺陷库.首先,对每一个历史缺陷信息进行关键词提取,利用余弦函数计算该缺陷与安全性词库中信息的文本相似度,如公式(1)所示.

$$S_h=\frac{|k_h\cap k_l|}{|k_h|} \tag{1}$$

其中,k_h表示每一个历史缺陷的关键词数量,$k_h\cap k_l$表示该缺陷与安全性词库共有的关键词数量,S_h表示该缺陷与安全性词库中信息的文本相似度值.当 S_h 大于某一个界定值θ时,即认为该缺陷是安全性缺陷.这样可以很好地提取项目中的安全性缺陷,从而组成一个安全性缺陷库.

1.2 提取安全性缺陷库文本关键词

此步骤的文本关键词提取来自于软件项目中安全性缺陷库,因此提取出来的关键词都是适合该项目的安全性关键词.其提取步骤与第 1.1.1 节的步骤类似,主要用自然语言处理技术(NLP)对文本信息进行分词、停用词去除、词根还原和同义词/近义词扩展等一系列步骤处理.

1.3 提取相关软件安全性缺陷

为了预测新缺陷的修改复杂度,首先需要分析与新缺陷相关的历史缺陷.因此,此步骤主要提取与新缺陷相关的安全性缺陷.SecDR 主要利用文本相似度计算缺陷之间的关系,具体计算如公式(2)所示.

$$S_n=\frac{|k_n\cap k_r|}{|k_n\cup k_r|} \tag{2}$$

其中,$|k_n\cap k_r|$表示新缺陷与安全性缺陷库中每条历史缺陷共有的关键词数量,$|k_n\cup k_r|$代表新缺陷的关键词与安全性缺陷库中每条历史缺陷关键词的合集,S_n 代表缺陷之间的相似度值.此步骤主要根据新缺陷在已有的安全性缺陷库中找到相关的历史缺陷,从而组成一个相关安全性缺陷库.

1.4 安全性缺陷修复复杂度等级预测

在相关安全性缺陷库中,可以获取以下信息:(1) 相关缺陷在修复过程中是否被 reopen 重新修复;(2) 相关缺陷的修复是否被其他缺陷阻止(block)、已经阻止的缺陷数量;(3) 每条缺陷对应的所有 commit 中,一共修改过的文件数量.

根据以上信息,可以预测新缺陷的修复复杂度.首先,统计整体安全性缺陷库中被 reopen 的缺陷比例、被阻止的缺陷比例以及每条缺陷平均修改的源码文件数量;然后,计算相关安全性缺陷库中相对的 3 个数值;最后,预测新缺陷的修复复杂度,其计算方法如公式(3)所示.

$$C_n=\lceil r_r-r_w\rceil+\lceil b_r-b_w\rceil+\left\lceil\frac{f_r-f_w}{f_r+f_w}\right\rceil \tag{3}$$

其中,r_r和 r_w分别代表相关安全性缺陷库和整体安全性缺陷库中被 reopen 的缺陷比例,b_r和 b_w分别代表相关安全性缺陷库和整体安全性缺陷库中被阻止的缺陷比例,f_r和 f_w分别代表相关缺陷库和整体安全性缺陷库中平均每条缺陷修改的文件数量,C_n 代表新缺陷修复复杂度的度量值.从公式(3)中可以看到,每项的取值都在(−1,1)的范围内,而所有项都取原值的上界,即最终取值{0,1}两种情况.所以 C_n 的最终取值有 4 种情况:{0,1,2,3}.当 C_n=0 时,定义新缺陷的修复复杂度为“简单”;当 C_n=1 或 2 时,定义新缺陷的修复复杂度为“一般”;当 C_n=3 时,定义新缺陷的修复复杂度为“困难”.

1.5 开发者经验等级判别

由于安全性缺陷修复的及时性、高质量要求等特征,本文主要从两个角度分析软件开发人员的历史经验:(1) 软件开发者的历史开发内容;(2) 软件开发者缺陷修复的专业性(质量和复杂度两个因素).

SecDR 主要利用一些主题词语反映软件开发者的历史开发内容,这些主题词语主要来源于:(1) 开发者历史修复过的缺陷的描述;(2) 修复缺陷时修改过源码文件的相关 diff 内容;(3) 相关缺陷对应的 comment 信息.

SecDR 主要从 3 个方面反映开发者修复安全性缺陷的专业能力:(1) 开发者历史修复的被阻止缺陷的比例(复杂度因素);(2) 被 reopen 的比例(质量因素);(3) 平均每条缺陷的修改文件的数量(复杂度因素).本文假设:开发者修复复杂缺陷的程度越好,他们修复该缺陷的专业性就越强.

通过对开发者的经验等级进行评估,SecDR 可根据缺陷修复的复杂度差异有针对性地推荐出不同经验级别的开发者.从而实现:推荐初级软件开发者修复简单的安全性缺陷、高级软件开发者推荐复杂的安全性缺陷.这样可以避免总是高级软件开发者的偏向推荐,从而让开发团队更好地协作工作.具体评估方法如公式(4)所示.

$$E_p=\left\lceil\frac{k_p-k_a}{k_p+k_a}\right\rceil+\lceil b_p-b_a\rceil+\lceil r_p-r_a\rceil+\left\lceil\frac{f_p-f_a}{f_p+f_a}\right\rceil \tag{4}$$

其中,k_p 和 k_a 分别表示该开发者拥有的关键词量和系统中平均每位开发者拥有的关键词量;b_p 和 b_a 分别表示该软件开发者和系统中平均每位开发者修复过的被阻止缺陷的比例;r_p 和 r_a 分别表示该软件开发者和系统中平均每位开发者修改过被 reopen 的缺陷的比例;f_p 和 f_a 分别代表该开发者和系统中平均每位开发者在修复缺陷时,平均每个缺陷修复的文件数量.E_p 代表该软件开发者的等级系数,可以看出,公式(4)中的每项都是取值上界,取值范围为{0,1}.因此,E_p 的取值范围在{0,1,2,3,4}这 5 种情况,当 E_p=0 或 1 时,假定该开发者为“初级开发者”;当 E_p=2 时,该开发者为“中级开发者”;当 E_p=3 或 4 时,该开发者为“高级开发者”.因此,最终开发者的经验等级被分为初级、中级、高级这 3 个等级.

1.6 开发者推荐

开发者推荐主要分为两个阶段:相关开发者推荐和排序.

SecDR 根据文本相似度推荐出与新缺陷有相关经验的开发人员.首先,从相关安全性缺陷库中抽取出所有关键词并进行查重,进一步去除重复单词;然后,将这些关键词与系统中每一位软件开发者拥有的关键词进行相似度计算,具体计算如公式(5)所示.

$$R_p=\frac{|k_p\cap k_r|}{|k_p|} \tag{5}$$

其中,k_p 为该开发者拥有的关键词数量,$k_p\cap k_r$ 表示该开发者与相关安全性缺陷共同拥有的关键词数量.公式(5)的分母只考虑了软件开发者的关键词,而没有考虑相关安全性缺陷库中关键词.这样可以将初级开发者更有效地推荐出来,避免了高级软件开发者的偏向推荐,以备下一步结合新缺陷的复杂度合理推荐出相应经验级别的开发者.

在推荐出相关开发者后,对他们进行排序.

首先,SecDR 计算每位相关开发者的排序权值,其计算方法如公式(6)所示.

$$W_p=R_p\times\theta^{(|C_n-E_p|+r)} \tag{6}$$

其中,R_p 是软件开发者的历史开发经验与新缺陷的相关系数,其代表了开发者的历史开发内容与新缺陷的匹配程度,其计算方法见公式(5);θ表示一个权值调整参数,其取值范围为(0,1);C_n 表示新缺陷的修改复杂度值;E_p 表示开发者的经验等级值,其差值很好地实现了开发者不同经验等级的推荐;r 代表开发者修复的缺陷是否被 reopen 过,这个因素可以说明开发者的历史修复质量.3 个参数的具体取值方法如公式(7)~公式(9)所示.

$$C_n=\begin{cases}0, & \text{简单}\\ 1,2, & \text{一般}\\ 3, & \text{复杂}\end{cases} \tag{7}$$

$$E_p=\begin{cases}0,1, & \text{初级}\\ 2, & \text{中级}\\ 3,4, & \text{高级}\end{cases} \tag{8}$$

$$r=\begin{cases}0, & \text{没有} reopened\cdot bug\\ 1, & \text{有} reopened\cdot bug\end{cases} \tag{9}$$

在公式(7)中,当缺陷复杂度等级为简单时,取值为 1,一般时取值 2,复杂时取值 3.公式(8)中,软件开发者为初级开发者时取值 1,中级开发者取值 2,高级取值 3.公式(9)中,当该软件开发者修复过的缺陷没有被 reopen 时取值 0,否则取值 1.

最终,每位开发者都对应一个排序权值 W_p,SecDR 根据 W_p 对所有相关软件开发者排序,得到一个排序列表,排序在前的开发者更适合修复该安全性缺陷.

2 实验验证

2.1 实验对象

为了验证 SecDR 推荐开发者的有效性,分别在 Mozilla,Libgdx(http://libgdx.badlogicgames.com/nightlies/docs/api/)和 ElasticSearch(https://www.elastic.co/)这 3 个开源项目上进行实验验证.经过第 1.1 节中安全性缺陷库的提取,这 3 个项目的安全性相关信息显示在表 3 中.其中,“项目名称”展示了实验对象的项目名称.对于 Mozilla 项目,“安全性缺陷数量”和“安全性开发人员数量”分别表示了本实验中应用到安全性缺陷的数量和对应的修复这些缺陷的开发者数量.在 Libgdx 和 ElasticSearch 两个项目中,“安全性缺陷数量”表示本实验中安全性缺陷数量,“安全性开发人员数量”代表了修复这些提取后的缺陷的软件开发者数量.“时间阶段”表示本实验应用到所有安全性缺陷报告的时间跨度.

Table 3 Characteristics of studied subjects

表 3 实验对象及其相关信息

项目名称	安全性缺陷数量	安全性开发人员数量	时间阶段
Mozilla	3 258	234	2005 年 9 月~2016 年 7 月
Libgdx	5 281	485	2010 年 2 月~2016 年 5 月
ElasticSearch	3 261	192	2010 年 3 月~2016 年 5 月

在表 3 中,Mozilla 项目以 Mozilla Firefox 和 Mozilla Thunderbird 等为主,项目中的所有缺陷发布在 bugzilla 平台上,由人工分类安全性软件缺陷并有针对性地管理和修复.Libgdx 是一个跨平台的 2D/3D 的游戏开发框架,由 Java/C/C++语言编写而成,提供独立的接口支持各种游戏开发.ElasticSearch 是一个基于 Lucene 的搜索服务器,提供了一个分布式多用户能力的全文搜索引擎.

2.2 实验设计

为了验证 SecDR 推荐的有效性,本文提出了以下 3 个实验研究问题.

- 问题 1:SecDR 推荐开发者的精度如何?

开发者推荐的精确度直接影响到软件维护的效率,因此,问题 1 主要验证 SecDR 能否有效地推荐合适的开发者修复安全性缺陷.

- 问题 2:与一般性缺陷的泛推荐技术相比,如 DR_PSF,SecDR 推荐软件开发者的精度提高了多少?

目前已有不少针对开发者的泛推荐技术[11−13],这些推荐技术能够根据 Bug 报告推荐出合适的开发者.之前我们也提出了一种个性化的开发者推荐技术 DR_PSF[7],DR_PSF 利用协相关主题模型进行开发者的推荐,实验结果说明了 DR_PSF 比现有的其他开发者推荐技术效果更好.但是这些技术在推荐开发者时很少针对安全性缺陷的特性进行推荐.而 SecDR 在推荐开发者时综合考虑了开发者的安全性缺陷修复的经验,而且实现了不同经验级别开发者的推荐.所以,问题 2 主要用于对比泛推荐方法和本方法在推荐开发者时的精确性,验证 SecDR 能否提高开发者的推荐精度.在前期工作中,DR_PSF[7]已显示出比现有其他泛推荐技术更好的推荐效果,本文仅对比了 DR_PSF 和 SecDR 的推荐精度.

- 问题 3:与已有项目开发者的实际分配相比,SecDR 推荐开发者的合理性如何?

在缺陷实际修复过程中,很多缺陷的修复需要 reopen,或者修复一段时间后需要重新修复,此时的缺陷分配可能不合理.而 SecDR 在推荐软件开发者时采用分级推荐机制,主要实现了“初级开发者修复简单缺陷、高级开发者修复复杂缺陷”的思想.所以,问题 3 主要验证 SecDR 推荐不同经验等级的开发者能否改善缺陷实际情况分配不合理的现象.

2.3 实验方法

本实验选择了安全性缺陷库中最近修复的 200 个缺陷作为测试集,其他缺陷作为 SecDR 的训练集.

问题 1:为了验证 SecDR 的推荐精度,使用 *recall* 值对本方法进行度量[11].*recall* 的计算方法如公式(10)所示.

$$recall@k=\frac{1}{r}\times\sum_{i=1}^{r}\frac{|RD(r_i)\cap AD(r_i)|}{|AD(r_i)|} \tag{10}$$

其中,r 代表训练缺陷的数量;k 代表针对每条缺陷,SecDR 推荐开发者人数的数量;$RD(r_i)$代表 SecDR 针对 bug_i 推荐的软件开发者;$AD(r_i)$代表实际修复 bug_i 的软件开发者.在计算 *recall* 值时,针对每一条缺陷,分别推荐不同数量的开发人员,例如,令 k=1,5,10.

本实验不采用另一个普遍的度量标准 *precision*,是因为在实际修复过程中,只有 1 个软件开发者修复软件缺陷,所以推荐结果最多只有 1 个是正确的,而其他的推荐结果都是错误的,所以 *precision* 不适合本实验的度量.

问题 2:SecDR 主要针对安全性软件缺陷推荐合适的开发人员,而现有的开发者推荐方法并没有针对安全性缺陷推荐.因此,本问题主要对比泛推荐方法 DR_PSF 和 SecDR 在推荐安全性缺陷开发者时的效果.在对比时,主要应用 *gain* 值进行推荐结果的比较,其计算方法如公式(11)所示.

$$gain@k_{SecDR-DR\_PSF}=\frac{recall@k_{SecDR}-recall@k_{DR\_PSF}}{recall@k_{DR\_PSF}}\times 100\% \tag{11}$$

其中,$recall@k_{SecDR}$ 和 $recall@k_{DR\_PSF}$ 分别代表 SecDR 和 DR_PSF 推荐 k 个软件开发者的 *recall* 值,其计算方法如公式(10)所示.

问题 3:本问题主要对比 SecDR 推荐与实际分配修复人员的合理性.在 SecDR 推荐开发者时,很多推荐结果与实际修复缺陷的人员不一致,此时,需要对比 SecDR 的推荐结果与实际修复人员哪一个更合理.主要从两方面进行对比.

1) 首先统计了推荐 10 位开发人员时,实际开发者不在推荐列表中的所有缺陷;然后,根据 SecDR 预测缺陷的复杂度等级分别比较实际开发者的经验等级和推荐列表中第一位开发者的经验等级,从而评估两者之间的合理性.具体评估步骤如下:(1) 统计缺陷复杂度与实际开发者经验等级一致的缺陷比例;(2) 统计缺陷复杂度与推荐列表中排名第一开发者的经验等级一致的缺陷比例;(3) 比较两者比例数据,如果实际开发者的比例较大,说明实际分配较为合理;否则,推荐结果较为合理.
2) 另一方面,找出测试集中所有被 reopen 或被重新修复的缺陷,统计出实际开发者在推荐列表中的比例和不在推荐列表中的缺陷比例.如果实际开发者不在推荐列表的比例较大,说明 SecDR 可以更好地规避不合理的软件开发者的推荐.

这样,通过以上步骤可以对 SecDR 的推荐结果与实际分配的情况进行对比,从而评估出 SecDR 推荐开发者的有效性.

2.4 实验结果

问题 1:本实验中,SecDR 分别推荐 1 位、5 位和 10 位软件开发者作为推荐结果.为了验证结果的有效性,本文主要利用 *recall* 作为度量标准分别计算推荐 1 位、5 位、10 位开发人员时的 *recall* 值.表 4 的“SecDR“列主要显示了本方法的 *recall* 结果值.从表 4 中可看到,针对实验中的 3 个开源项目(Mozilla,Libgdx,ElasticSearch),SecDR 在推荐 1 位开发者时,有效值在 8.0%~11.0%范围内,平均推荐结果是 9.5%;推荐 5 位开发者时,有效值范围在 17.5%~23.5%,平均结果为 20.5%;当推荐 10 位开发者时,范围在 36.0%~51.5%,其平均结果是 45.3%.实验结果表明:SecDR 在推荐 1 位、5 位和 10 位开发者时,平均精度分别为 9.5%、20.5%和 45.3%.因此,SecDR 针对安

全性修复的开发者推荐效果是可接受的.

问题 2:为对比 SecDR 与已有的开发者推荐方法(如 DR_PSF)推荐安全性软件开发者的精确性,本文计算 SecDR 对比于 DR_PSF 的增益(*gain* 值).其计算结果在表 4 的“*gain* 值”列中.可以看到,SecDR 相比于 DR_PSF 的 *gain* 值范围在 9.3%~69.2%.在推荐 1 位、5 位、10 位软件开发者时,*gain* 值的平均结果为 41.8%、19.2%、38.5%.这说明了当分别推荐 1 位、5 位、10 位软件开发者时,SecDR 推荐开发者的精度比 DR_PSF 平均提高了 41.8%、19.2%、38.5%.因此,SecDR 相比于泛推荐技术 DR_PSF 可以更有效地针对安全性缺陷推荐开发人员.

Table 4 Results of developer recommendation for SecDR and DR_PSF (recall@1,5,10 and gain@1,5,10)

表 4 SecDR 和 DR_PSF 开发者推荐结果(*recall*@1,5,10)以及对比结果(*gain*@1,5,10)

项目名称	*k*	SecDR (%)	DR_PSF (%)	*gain* 值(%)
Mozilla	1	8.0	5.5	45.5
	5	21.0	16.5	27.3
	10	51.5	32.0	60.9
Libgdx	1	11.0	6.5	69.2
	5	17.5	13.5	29.6
	10	36.0	32.0	12.5
ElasticSearch	1	9.5	8.0	18.8
	5	23.5	21.5	9.3
	10	48.5	34.0	42.6
平均情况	1	9.5	6.7	41.8
	5	20.5	17.2	19.2
	10	45.3	32.7	38.5

问题 3:在对比实际开发者的分配效果与 SecDR 的推荐结果时,主要从两个方面进行评估:首先,将缺陷的复杂度预测值与实际开发者的经验值进行匹配,当实际开发者的经验等级与缺陷的复杂度等级一致时,认为该缺陷分配有效;否则无效.经统计,图 2 中浅色的柱条给出了实际开发者与缺陷复杂度一致的缺陷比例,其中,ElasticSearch 项目的值为 32.8%,Libgdx 项目的值为 36.3%,Mozilla 的值为 29.2%,3 个项目的平均值为 32.8%.这说明了开发者实际分配的经验等级与缺陷复杂度等级一致的概率平均为 32.8%.

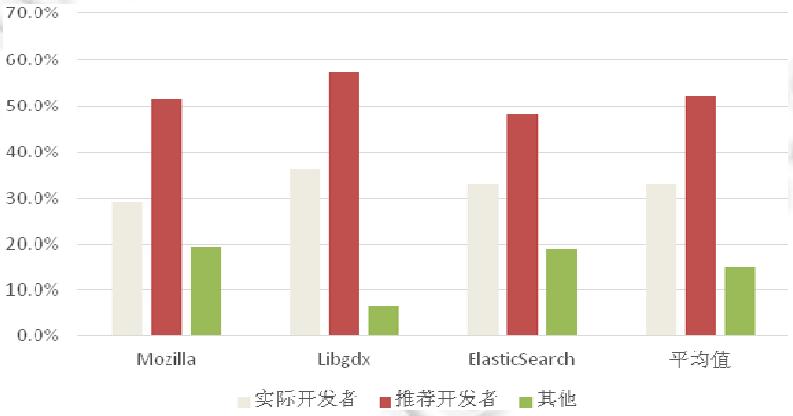

Fig.2 Results of developer experience and bug complexity for actual developers and recommended developers

图 2 实际开发者和推荐开发者的经验与缺陷复杂度匹配结果柱状图

然后,再匹配推荐列表中排名第一位的软件开发者的经验等级和缺陷的复杂度,同样计算匹配一致的缺陷比例.图 2 中深色的柱条代表了推荐结果与缺陷等级一致的缺陷比例,其中,ElasticSearch 项目的值为 48.1%,Libgdx 项目的值为 57.2%,Mozilla 的值为 51.4%,3 个项目的平均值为 52.2%.这说明了 SecDR 推荐出的开发者等级与缺陷复杂度等级一致的概率平均为 52.2%.因此,SecDR 的推荐结果更好地匹配了缺陷修复的复杂度,这

说明 SecDR 的推荐结果在软件开发者经验等级上推荐的更合理.

另一方面,找出测试集中所有被 reopen 或者重新修复的缺陷,分别统计出这些缺陷的实际开发者在推荐列表中的比例和不在推荐列表中的比例.从图 3 中可看到,浅色的柱条统计了实际开发者在推荐列表中的缺陷比例,其中,ElasticSearch、Libgdx 和 Mozilla 这 3 个项目的比例值分别是 27.6%、24.3%和 32.4%;而被 reopen 或者重新修复的缺陷的实际开发者不在推荐列表中的缺陷比例的平均值为 71.9%.因此,面对这些被多次修复的软件缺陷,SecDR 可以更好地规避不合理的软件开发者的推荐.

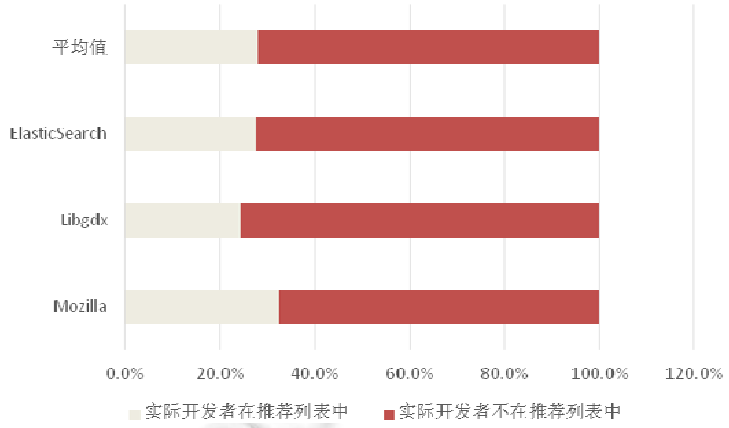

Fig.3 Developer recommendation results for reopened and re-fixed bugs in the experiment

图 3 面向测试集中被 reopen 或者重新修复缺陷的开发者推荐结果柱状图

3 有效性威胁

首先,在实验中,所有数据都抓取于开源网站.但是在抓取过程中,由于网络不稳定的原因,仍然有少量数据会被爬虫程序跳过,从而引起数据不全的现象.虽然遗失的这些数据可能会影响 SecDR 的结果,但这些数据往往是少量的,其影响不会很大.所以,本实验的可靠性也不会受到很大干扰.

第二,本实验主要建立在 3 个不同应用场景下的开源软件对象上,但对于一些企业内部的软件数据,暂时还无法获取,所以本文不能确保 SecDR 对于企业数据推荐的有效性.另外,本文的实验主要建立在一些相对比较成熟的软件项目上,这些项目里有很多经验丰富的高级软件开发者,同时也兼备一些经验不太丰富的初级开发者,此类数据的特征比较适合于 SecDR 的个性化推荐.而对于一些刚刚成立的软件项目,由于软件库中的历史数据不够丰富,很难分析出开发者的经验等级以及预测缺陷修复的复杂度,所以 SecDR 不太适合刚建立的软件项目推荐.另一方面,对于当今特别流行的一些软件项目,例如 Spark 等,由于这些项目非常流行,从而会吸引很多开发经验特别丰富的软件开发者新加入项目团队,SecDR 可能会将这些软件开发者分析为初级开发者,因为他们对该项目的开发情况可能还不太熟练.所以,SecDR 的推荐效果更适合一些成立时间较长但相对不太流行的软件项目.

第三,由于大部分软件项目中没有现成的安全性缺陷库,SecDR 主要根据 Mozilla 项目中的安全性缺陷库提取新项目中的一些与软件安全性相关的缺陷,并组成一个新的安全性缺陷库.由于项目的应用场景不同,可能会有很多缺陷被错误提取或者遗漏,从而导致 SecDR 处理的源数据的可靠性不足.另外,在提取安全性缺陷时,SecDR 主要根据 Mozilla 项目中的安全性关键词进行提取,例如 buffer overflow、memory leakage 等.但根据相关论文研究,利用关键词预测安全性缺陷时,很可能会遗漏很多安全性缺陷,但预测结果相对比较正确[1].这可以说明 SecDR 提取的安全性缺陷基本都是与软件安全性相关的.因此,利用提取出的缺陷数据分析软件开发者的安全性相关经验是可靠的,所以 SecDR 推荐的软件开发者在一定程度上也是可靠的.

第四,在预测安全性缺陷修复的复杂度时,首先找到了与该缺陷相关的其他安全性缺陷,然后根据这些相关

缺陷修复的复杂度对新缺陷进行预测.本文没有做缺陷预测结果准确性相关的实验,可能会有部分缺陷的预测结果会有偏差,从而影响SecDR对不同等级的软件开发者推荐的结果.另外,在研究实验问题3时,在第一方面验证还应用了缺陷的预测等级对推荐结果进行评估,这些数据可能都会影响实验结果.但本文主要研究的是软件开发者的个性化推荐,并不是预测缺陷修复的复杂度问题,因此在预测复杂度时可能考虑的因素还不够全面,对于此方面的问题,我们后面也会进一步研究.

第五,在回答问题1时,主要利用了 *recall* 值评估 SecDR 推荐的准确性,也许还有其他的度量标准,但 *recall* 度量是软件开发者推荐技术领域里使用最普遍的一种度量,具有很好的代表性[11].在比较 SecDR 与 DR_PSF 推荐安全性开发者时,DR_PSF也实现了软件开发者的个性化推荐,但DR_PSF更多是区分拥有不同开发经验内容的软件开发者,从而推荐相关软件开发者完成相应的软件缺陷;而本文主要是针对安全性缺陷进行推荐,安全性缺陷的修复除了考虑开发者的开发内容,可能更多地需要考虑软件开发者的安全性经验和修复质量.所以 DR_PSF 与 SecDR 的应用场景略有不同,其表现的结果差距也较大.但这也可以说明 SecDR 的推荐更适合于解决安全性缺陷.在评估问题 3 时,本文主要通过两方面角度对比推荐结果与实际分配,也许实际分配时的情况更加复杂,比如最合适的软件开发者暂时请假或有别的任务等,但这些因素在评估中均被忽略.然而,对于一些修复情况不太理想的软件缺陷,SecDR 推荐的结果与实际修复的软件开发者不一致的情况会更大,而对于修复情况较好的缺陷推荐精度较高,表明了本方法推荐结果的合理性.

最后,在研究问题3对比实际开发者的分配效果与SecDR的推荐结果时,我们采用了 *reopen* 等指标来评价实际分配修复人员的合理性.但是实际环境中,由于缺陷修复过程较为复杂,安全相关的缺陷修复尤为复杂,因此,依据 *reopen* 等指标来评价实际分配修复人员的合理性并不一定总是合理.后期我们将寻找更合适的度量来进一步验证该推荐的合理性.

4 相关工作

近年来,缺陷修复者推荐技术得到广泛关注.Hossen 等人提出了 iMacPro 方法,主要通过提取源码中的开发人员和维护人员,基于他们的维护历史推荐一个最合适的软件开发者[11].Zhang 等人提出了一种 KSAP 的推荐方法,KSAP通过构建复杂的缺陷网络推荐系统中的软件开发者[14].Xia等人提出了一种分析缺陷报告和开发者的组合方法 DevRec[15].Zhang 等人开发了一个工具 BUTTER,应用社交网络分析软件开发者的相关特征来推荐一系列的相关软件开发者[16].Wang 提出了一个方法 FixerCache,在推荐软件开发者时,分析了他们的历史开发行为进行推荐[17].Zhang 等人在推荐的过程中,主要结合了主题模型和软件开发者之间的相互关系来推荐一个最合适的软件开发者解决软件缺陷[18].YANG 等人实现了个性化推荐,在推荐时主要考虑了软件开发者的历史开发内容和开发习惯[7].Xia 等人提出了一个工具 DevRec,这个工具在推荐软件开发者时不仅分析了软件缺陷库,而且分析了软件开发者的历史开发经验[19].

相对于以上这些泛推荐技术,本文在针对安全性缺陷推荐开发者时不仅考虑了开发者的历史开发内容和软件安全性知识,还分析了开发者的历史修复质量以及历史修复缺陷的复杂度.此外,SecDR 预测了新缺陷的修复复杂性,实现了简单缺陷推荐初级软件开发者、复杂缺陷推荐高级软件开发者,从而实现了开发者的多经验级别推荐,进一步提高了开发者推荐的准确性.

5 总结与展望

本文针对安全性缺陷的修复提出一种新的开发者推荐方法 SecDR.SecDR 在推荐开发者时不仅考虑了开发者的历史开发内容和软件安全性知识,还分析了软件开发者的历史修复质量以及历史修复安全性缺陷的复杂度.此外,SecDR 预测了新缺陷的修复复杂性,实现了简单缺陷推荐初级软件开发者、复杂缺陷推荐高级软件开发者,从而实现了开发者的多经验级别推荐.为了验证 SecDR 的有效性,本文在 3 个开源项目上(Mozilla, Libgdx,ElasticSearch)做了相应的实验验证.在实验验证时,本文对比了 SecDR 和传统的非安全性缺陷推荐方法 DR_PSF 推荐软件开发者的精度.通过对比实验可发现:当分别推荐 1 位、5 位、10 位软件开发者时,SecDR 的

推荐精度比 DR_PSF 平均高出 19%~42%.另一方面,本文还对比了 SecDR 与实际开发人员的分配情况,通过相关统计发现,SecDR 推荐软件开发者更合理.然而,本方法在推荐软件开发者时首先识别了系统中已出现过的安全性缺陷,在识别安全性缺陷时,仅仅从主题词的角度进行了初步安全性预测,后期仍然需要进一步加强安全性缺陷的识别指标,从而更精确地构建出安全性缺陷库.另外,在评估安全性缺陷开发者推荐的实验中,将补充其他实验对象,选取其他实验度量指标,对比最新的相关开发者推荐技术来评估本文开发者推荐技术,进一步验证本文技术的有效性.目前的开发项目中,绝大部分开发者修复安全性缺陷的经验比较匮乏,也就是比较难找到富有安全性经验的开发者,未来将进一步完善推荐技术,利用我们所提出的个性化开发者推荐技术[6,20],除了推荐开发者以外,还推荐安全性缺陷相关的方案辅助开发人员理解缺陷,提高开发者的安全性缺陷的修复能力.

References:

[1] Gegick M, Rotella P, Xie T. Identifying security bug reports via text mining: An industrial case study. In: Proc. of the 7th Int'l Working Conf. on Mining Software Repositories. 2010. 11−20.

[2] Zaman S, Adams B, Hassan AE. Security versus performance bugs: A case study on firefox. In: Proc. of the 8th Working Conf. on Mining Software Repositories. 2011. 93−102.

[3] Witschey J, Zielinska O, Welk A, Murphy-Hill E, Mayhorn C, Zimmermann T. Quantifying developers' adoption of security tools. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). 2015. 260−271.

[4] Mitropoulos D, Gousios G, Spinellis D. Measuring the occurrence of security-related bugs through software evolution. In: Proc. of the 16th Panhellenic Conf. on Informatics. 2012. 117−122.

[5] Shokripour R, Anvik J, Kasirun ZM, Zamani S. A time-based approach to automatic bug report assignment. Journal of Systems and Software, 2015,102:109−122.

[6] Yang H, Sun X, Li B, Hu J. Recommending developers with supplementary information for issue request resolution. In: Proc. of the 38th Int'l Conf. on Software Engineering Companion. 2016. 707−709.

[7] Yang H, Sun XB, Li B, Duan YC. DR_PSF: Enhancing developer recommendation by leveraging personalized source-code files. In: Proc. of the 40th IEEE Computer Society Int'l Conf. on Computers, Software and Applications. 2016. 239−244.

[8] Sun X, Liu X, Hu J, Zhu J. Empirical studies on the nlp techniques for source code data preprocessing. In: Proc. of the 3rd Int'l Workshop on Evidential Assessment of Software Technologies (EAST 2014). 2014. 32−39.

[9] Porter MF. An Algorithm for Suffix Stripping. Morgan Kaufmann Publishers, Inc., 1997. 130−137.

[10] Sun X, Yang H, Leung H, Li B, Li HJ, Liao L. Effectiveness of exploring historical commits for developer recommendation: An empirical study. Frontier of Computer Science, 2018,12(3):528−544.

[11] Hossen H, Kagdi HH, Poshyvanyk D. Amalgamating source code authors, maintainers, and change proneness to triage change requests. In: Proc. of the 22nd Int'l Conf. on Program Comprehension (ICPC 2014). 2014. 130−141.

[12] Zhang W, Han G, Wang Q. Butter: An approach to bug triage with topic modeling and heterogeneous network analysis. In: Proc. of the 2014 Int'l Conf. on Cloud Computing and Big Data (CCBD 2014). 2014. 62−69.

[13] Wang S, Zhang W, Wang Q. FixerCache: Unsupervised caching active developers for diverse bug triage. In: Proc. of the 8th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement (ESEM 2014). 2014. 25:1−25:10.

[14] Zhang W, Wang S, Wang Q. Ksap: An approach to bug report assignment using KNN search and heterogeneous proximity. Information and Software Technology, 2016,70:68−84.

[15] Xia X, Lo D, Wang X, Zhou B. Dual analysis for recommending developers to resolve bugs. Journal of Software: Evolution and Process, 2015,27(3):195−220.

[16] Mitropoulos D, Gousios G, Spinellis D. Measuring the occurrence of security-related bugs through software evolution. In: Proc. of the 16th Panhellenic Conf. on Informatics. 2012. 117−122.

[17] Yin Z, Yuan D, Zhou Y, Pasupathy S, Bairavasundaram L. How do fixes become bugs? In: Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering (ESEC/FSE 2011). 2011. 26−36.

[18] Zhang T, Yang G, Lee B, Lua EK. A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In: Proc. of the 21st Asia-Pacific Software Engineering Conf. (APSEC 2014). 2014. 223−230.

[19] Xia X, Lo D, Wang X, Zhou B. Accurate developer recommendation for bug resolution. In: Proc. of the 20th Working Conf. on Reverse Engineering (WCRE 2013). 2013. 72−81.

[20] Sun X, Yang H, Xia X, Li B. Enhancing developer recommendation with supplementary information via mining historical commits. Journal of Systems and Software, 2017,134:355−368.

孙小兵(1985－),男,江苏姜堰人,博士,副教授,CCF 高级会员,主要研究领域为软件工程.

杨辉(1990－),男,硕士,主要研究领域为软件技术.

周澄(1985－),女,博士生,讲师,CCF 学生会员,主要研究领域为智能软件分析.

李斌(1965－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件技术.