# 程序静态分析报告处理方法综述

黄　松,龚士豪
(陆军工程大学 指挥控制工程学院,江苏 南京 210007)

**摘　要**:在软件测试过程中,使用静态分析工具自动化扫描程序是发现程序中缺陷和漏洞的有效方法之一。然而,分析工具自身的局限性会导致分析报告中存在大量误报,进而致使审核分析报告成本过高,这不仅降低了工具的实用性,也大大延长了测试周期。为了减轻测试人员审核分析报告的工作量并提高工具的可用性,国内外研究人员提出了多种静态分析报告处理方法。对近些年来国内外研究人员在静态分析报告处理方面的研究工作进行综述。首先,对静态分析技术与静态分析报告处理进行简要介绍,基于处理方法的基本思想给出了方法的分类。接着,依次总结了各类处理方法的研究成果,并在方法之间进行横向对比,全面分析了当前主流方法的优缺点。最后,详细指出了该领域目前存在的问题,并给出了相应的研究建议,为相关研究人员全面深入了解程序静态分析报告处理方法提供了基础性参考。

**关键词**:静态分析;警报;融合;分类;排序
**中图分类号**:TP311　　**文献标识码**:A　　**文章编号**:1673-629X(2023)01-0014-07
doi:10.3969/j.issn.1673-629X.2023.01.003

# A Survey of Processing Methods of Program Static Analysis Report

HUANG Song, GONG Shi-hao
(School of Command & Control Engineering, Army Engineering University of PLA, Nanjing 210007, China)

**Abstract**: During software testing, automated scanning of programs using static analysis tools is one of the effective ways to find bugs and vulnerabilities in programs. However, the limitations of the analysis tool itself will lead to a large number of false positives in the analysis report, and then the cost of reviewing the analysis report is too high, which not only reduces the usefulness of the tool, but also greatly prolongs the testing period. In order to reduce the workload of testers reviewing analysis reports and improve the usability of tools, researchers at home and abroad have proposed a variety of static analysis report processing methods. We review the research work of domestic and foreign researchers on static analysis report processing in recent years. Firstly, the static analysis technology and static analysis report processing are briefly introduced, and the classification of methods is given according to the basic idea of processing methods. Then, the research results of various processing methods is summarized in turn, a horizontal comparison between the methods is made, and the advantages and disadvantages of the current mainstream methods are analyzed comprehensively. Finally, the existing problems in this field are pointed out in detail, and corresponding research suggestions are given, which provides a basic reference for relevant researchers to comprehensively and deeply understand the processing method of program static analysis report.

**Key words**: static analysis; alerting; fusion; classification; sorting

## 0　引　言

随着信息化程度的不断提高,软件的复杂性也在不断增加,这对提高软件质量、保证其行为的可信性带来了一定的困难。许多软件产品在发布甚至广泛应用后,往往仍包含各种各样的缺陷和漏洞,这不仅降低了用户体验,同时也使得软件面临安全风险。静态分析在寻找代码缺陷和漏洞方面具有举足轻重的作用,已成为软件质量保证实践中不可或缺的一部分[1]。静态分析工具的应用可以在一定程度上提高软件质量,但分析报告中的大量误报以及人工检查报告所需的花费会降低静态分析工具的可用性,于是各种报告处理方法不断涌现。该文将静态程序分析报告处理方法定义为“对静态程序分析工具输出的报告进行处理,以达到降低工具误报率和漏报率目的的一系列方法”。通过对报告进行后续处理,上述问题能够得到缓解,分析工具的可用性得以提高。迄今为止,机器学习、统计分析、数据融合等技术已被应用于报告处理领域之中。

通过调研国内外近些年在程序静态分析报告处理

**收稿日期**:2022-03-16　　**修回日期**:2022-07-18
**基金项目**:国家重点研发计划重点专项项目(2018YFB1403400);陆军工程大学基础前沿科技创新工程前沿创新项目(KYZYJQZL2203)
**作者简介**:黄　松(1979-),男,博士,教授,博导,CCF 高级会员(29597S),研究方向为软件测试与质量评估;通讯作者:龚士豪(1998-),男,硕士研究生,研究方向为软件测试。

方法方面的研究进展,简要综述程序静态分析报告处理方法的研究动向,同时根据调研内容,讨论程序静态分析报告处理方法未来可能的研究方向。

## 1　静态分析及其报告处理

程序分析技术以分析过程“是否需要运行软件”为准则,可以划分为静态分析技术和动态分析技术两大类。动态分析通过运行程序获取程序的输出或者内部状态等信息来验证程序是否存在缺陷和漏洞,针对特定的输入,通常可以取得精度较高的分析结果,但对于其他输入则无法保证[2]。静态分析无需运行程序,通过语法分析、定理证明、符号执行等技术来发现程序中存在的缺陷。与动态分析相比,静态分析在编码阶段就可以实施,能更早地发现问题,并且对程序执行路径分析的更全面,可以发现更多缺陷。

目前流行的静态分析工具,例如 Coverity、Klocwork、Fortify、Parasoft 等,大多采取多种分析技术相结合的分析策略,旨在发现更多的缺陷和漏洞。但根据 Rice 定理:不存在可以判定任何程序是否具有某种非平凡属性的通用算法[3]。例如程序路径的可行性就是不可判定的,因此无法要求静态分析工具对程序进行精确的分析,只能采用流不敏感或上下文不敏感等分析策略给出保守的结果,这将不可避免地导致误报和漏报的产生[4]。误报即分析工具报告程序某处存在缺陷,但实际上此缺陷并不存在。漏报即程序某处实际存在缺陷,而分析工具没有报告。静态分析工具的分析性能大多使用漏报率和误报率来衡量。

研究人员发现,静态分析工具的误报率通常高达 30% ~100%,每千行代码就会产生 40 个警报[5]。据估计,确认一个警报的正确性大约需要 5.28 分钟,假设一个工具报告了 500 个警报,那么需要测试人员每天工作 8 小时,连续 5.5 天才能将这份报告全部检查完[6]。这种体验往往会让测试人员拒绝再使用这些工具。

对此,多年来,人们在静态程序分析报告处理方面做了大量的研究工作,取得了很大的进步。通过提高静态分析工具的分析精度或对报告进行有效地后续处理可以在一定程度上解决上述问题[7]。对于提高分析精度这一方法,许多学者进行了研究[8-10],但由于验证问题通常是不可判定的,并且分析工具在设计时也必须在精度与速度之间做出取舍,因此报告中存在误报是无法避免的。针对第二种方法——对报告进行有效地后续处理,国内外的学者也提出了许多方案,根据处理方法的基本思想可分为:基于多工具报告融合的处理技术、基于警报分类的处理技术,以及基于警报排序的处理技术。这些方法通常是互补的,可以独立工作,也可以通过不同的方式组合在一起。

## 2　基于多工具报告融合

虽然市面上存在大量静态分析工具,但由于不同静态分析工具的工作原理与运行机制大不相同,单一工具通常只在小部分问题上表现的很好。例如,对于词法工具来说,当漏洞规则库不同、漏洞定义的等级不同时,分析结果会出现很大差异[4]。因此,在检查源代码时,为了尽可能发现其中的缺陷,不能仅使用单一的工具,但每个工具在生成分析报告时遵循的风格不同,工具之间的缺陷检测能力的优势很难结合起来。若能将不同分析工具的分析结果通过某种方式融合在一起,则可以充分利用分析工具各自的优势,让报告间相互补充,相互验证,降低报告的漏报率与误报率,有效提高工具的效率和性能。使用户轻松地从多个静态分析工具中受益,从而提升软件质量。

Kong 等人[11]提出了一种基于数据融合的源代码静态分析漏洞检测技术。他们首先将来自不同分析工具的分析报告中的漏洞信息提取出来,以一个六元组序列的形式存储。六元组包括漏洞的文件名、漏洞所在行数、危险等级、漏洞类型、导致漏洞的函数和漏洞威胁评分。通过这种统一的格式,多份报告可以融合为一份报告。此外,融合过程中每条漏洞被赋予一个估分随机变量 $X$ 代表该漏洞为真实漏洞的概率。其值的定义准则为:

(1)多个工具同时识别的漏洞分值高;

(2)漏洞本身危险等级高则分值高;

(3)来自不同分析工具的结果以工具自身的可信赖度权值体现对分值的贡献。

工具的可信赖度权值可以由用户指定,也可以由误报率和漏报率自动评估。融合以后,漏洞按照估分由高到低排列。该方法经过融合多个工具的报告并对最终报告中的警报进行排序,增加了检测出的漏洞数,继而降低了漏报率,并且最可能是真正漏洞的漏洞被放在前列,用户可以以较少的时间找出更多的漏洞。但此方法并未考虑不同工具的报告之间存在重复警报的问题,融合报告中的冗余警报仍需处理。

Dang[12]提出了一个混合模型,将数据融合与统计分析相结合,对不同工具生成的报告中的警报进行融合并排序。与 Kong 等人[11]的方法类似,在数据融合时,提取不同工具所产生的警报间共有的属性,将警报融合为统一格式。例如 Cppcheck 和 SciTool,它们的警报共有属性是文件名、警报类别和行号,而 PMD 和 FindBugs 则是文件名、优先级、行号和规则集。与 Kong 等人[11]不同的是,Dang 并不为单个警报排序,他认为工具生成的警报都与文件或类相关,因此可以

对文件或类排序。从融合后的警报中挖掘出统计特征之后,对这些特征使用主成分分析法(PCA),从而为文件或类计算得分并排序。该方法可以很好地将具有大量真实缺陷的文件放在警报列表顶部,将具有误报的文件放在列表底部,但并不会去除误报也不会检测正报。

Meng 等人[13]通过为缺陷定义缺陷模式,并提供一个通用规范使它们保持一致的描述风格,从而融合来自不同工具的分析报告。每个通用规范包含三个部分:摘要信息、所属缺陷模式类别和缺陷严重等级。当不同工具产生的结果被转换为一致的描述风格后,再应用优先级策略对其进行排序。优先级策略包括:

(1)根据警报严重等级。例如,属于“错误”的警报要排在“不良风格”前面;

(2)同一类中被多次报告的缺陷拥有更高的优先级。

经过上述操作,来源于不同工具但属于同一缺陷模式的警报被放到一起,并且危害等级高的警报位于报告前列。用户可以选择感兴趣的缺陷模式重点关注,将注意力放在更重要的缺陷上。但此方法仅适用于文中使用的三个 Java 静态分析工具,如果出现不属于任何缺陷模式的警报则无法对其进行处理。此外,虽然来源于不同工具但属于同一缺陷模式的警报被放到一起,但警报冗余问题依旧存在,并且也没有分析警报的正确性。

张丽等人[14]为了解决在分析报告简单融合时,融合报告中引入了每一种工具的误报的问题,重点研究了如何优化融合后分析报告的质量。她们利用缺陷数据的一些自然属性,例如报告该缺陷的工具、缺陷所在代码位置、缺陷所在代码的复杂程度等,通过机器学习算法对警报进行二元分类(正报和误报),从而剔除部分警报,实现集成优化。能够实现分类的机器学习算法很多,她们选用了朴素贝叶斯、逻辑回归、决策树和支持向量机。优化过程包括报告生成与解析、数据预处理、分类优化三个部分。其中的数据预处理部分又包括了数据清洗、标准化、函数粒度级别的聚合、向量化以及数据分割的过程,以将数据处理成机器学习算法所需的形式。经过在 Juliet 数据集上的大量实验可证实,优化后的分析报告在各项指标上均优于优化前,其中支持向量机和决策树获得了较好的分类效果。

## 3 基于警报分类

基于警报分类的处理技术是根据警报之间的某些关系或代码特征对警报进行分类的技术。在这类方法中,机器学习技术被大量使用以构建分类器,例如 k-最近邻(K-Nearest Neighbor)[15]、随机森林(Random Forest)[16]、支持向量机(Support Vector Machines)[17]、逻辑回归模型[18]等。在基于警报之间关系的分类技术中,通常一组警报中的某一个警报处于主导地位,其正确性被验证后,同组中的其他警报就无需再检查。基于代码特征的分类方式则直接将警报分为正报和误报,测试人员只需检查正报。两种方法都大大降低了警报审查的工作量,但错误的分类会导致部分正报被忽视,也就造成真实缺陷仍然遗留在程序中,因此对于安全关键型软件来说,该方法并不安全。

Ruthruff 等人[18]使用逻辑回归模型,基于警报和其所涉及代码中的信息来预测警报是否为误报或可操作警报。值得关注的是,他们在构建模型过程中使用了一种筛选方法,该方法会丢弃预测能力较低的指标,节省了预测成本。在谷歌进行的一项案例研究表明,该模型在预测误报方面的准确率超过 85%,在识别可操作警报方面的准确性超过 70%。

张大林等人[19]提出基于缺陷关联的报告处理技术。他们根据不同缺陷间的依赖关系将缺陷分组,当缺陷 a 可靠地依赖于缺陷 b,那么如果缺陷 b 被确定为误报,则缺陷 a 也是误报,缺陷 b 就被称为主导缺陷。缺陷间的依赖关系通过切除一个缺陷的错误状态来确定,如果一个缺陷的错误状态被切除后会影响另一个缺陷的警报生成,则依赖关系存在。而警报是否为误报是根据缺陷的错误状态是否可达来判断的,错误状态不可达则代表警报为误报。该方法可以有效减少警报的确认工作,为大规模软件使用静态分析工具提供了有力支持。但如何在静态分析工具所提供的抽象域下精确地切除缺陷错误状态仍需改进。Lee 等人[20]提出的警报聚类方法与张大林[19]的方法类似,通过寻找主导警报减少检查警报的负担。在 14 个开源基准测试中,他们的聚类方法可以识别出 45% 的警报是非主导的,相当于减少了 45% 的警报检查的工作量。

Reynolds 等人[21]通过研究不同类型的误报信息,总结出 14 个核心的误报模式,实现了自动判断一条警报是否为误报。在研究过程中,由于判断一条警报是否为误报并不容易,因此他们选择了 Juliet 测试套件作为数据集,因为在这个测试套件中漏洞所在位置已经添加了注释来指示。当发现一条误报时,他们会对产生这条误报的源代码进行删减,直到删除下一个元素会导致误报消失。通过这种方式,可以去掉源代码中与误报无关的所有元素,剩下的代码即为误报模式。然后为这些模式分类就可以确定代码库中反复出现的源代码结构,正是这些结构会导致静态分析工具产生误报。这项工作减少了开发人员和测试人员需要验证的警报数量,但同时也具有一些局限性。首先,Juliet

测试套件是人造数据集,不能代表实际代码。其次,缺乏对误报模式的规范描述,这种描述应该是独立于语言的。最后,数据不完整,研究中缺乏误报模式的出现频率。

在进一步研究中,Koc 等人[22]在预处理阶段引入了两种技术来缩减代码,分别是方法体和程序切片。方法体是一种较为简单的方法,仅仅是将包含错误行的方法的主体部分保留下来。但需要注意的是,与警报相关的许多代码位置并不在警报方法的主体中,很多情况下警报的原因跨越了多个方法和类,因此这种缩减并不完美。程序切片是在程序中给定一个位置,将程序缩减到最小形式,但在该位置仍保持相同行为的技术。在训练模型阶段,同样使用了两种技术,分别是基于贝叶斯推理的学习模型和基于神经网络的长短期记忆网络(LSTM)。在国际通用数据集 OWASP benchmark 上的实验表明,朴素贝叶斯模型在方法体和程序切片方法下的分类准确率分别为 63% 和 72%,LSTM 则分别为 89.6% 和 85%,均取得了不错的分类结果。但该实验仅针对 OWASP benchmark 下的 SQL 注入缺陷,数据集和缺陷类型都比较片面。

Flynn 等人[16]分别使用了 Lasso 回归模型、分类回归树(CART)、随机森林(Random Forest)和极端梯度提升(Extreme Gradient Boosting)四种分类技术,基于 28 个特征训练分类器。并且使用多个静态分析工具,1 个开源数据集和 3 个来自合作者的匿名数据集生成报告。测试结果表明,使用多个静态分析工具可以有效提高分类的准确性,但许多类型的缺陷标记的不够充分,导致对这些缺陷的预测准确性较差。

Heckman[23]使用 51 个候选特征,15 种机器学习算法建立模型对警报分类,获得了 88% ~97% 的平均正确率。构建模型分为四个步骤:

(1)收集静态分析产生的警报的特征。

(2)使用属性评估算法来选择重要的警报特征集合。

(3)使用选定的特征集合和机器学习建立模型。

(4)通过指标评估选择出最佳模型。

他们发现,不同项目之间选用的警报特征和最佳模型是不同的。也就是说,当面对不同的项目时,需要重新选择警报特征并训练模型才能取得较好的分类效果,这在工程实践中显然是不可取的,如何使分类模型更加通用是值得考虑的问题。

## 4　基于警报排序

基于警报排序的处理技术一般通过计算每一条警报是正报的概率来为警报排序,将最有可能揭示真实错误的警报放到报告顶端,而具有较低正报可能性的警报被放到报告底部。排序有助于测试人员在给定时间内发现更多的缺陷,提高工作效率。

当前有许多不同类型的排序算法,它们仅针对特定的程序语言,并非通用。排序算法的实现方法主要分两类:第一类是通过分析报告中的每一条警报,寻找错误警报之间的关系和它们之间的相同点,找出错误警报的分布规律,从而计算得出更高效的报告排序;第二类是通过统计的方法,首先经过统计学的分析,使用贝叶斯网络构造模型,再利用模型建立测试集与训练集,从而预测每条警报为正报还是误报。但是一般来说,所有的警报都需要被检查,由于警报只是被排序而没有过滤,因此这种方法也被认为是一种安全的方法。

Kremenek 和 Engler[5]提出一种基于统计分析对警报进行排序的技术——Z-Ranking。他们认为,在通常情况下,源代码中的缺陷密度很低,因此在针对某个问题检查源代码时,未触发该检查的位置应该较多,而触发该检查的位置相对较少。相反,如果检查导致很多位置触发检查,则这些触发很可能是误报。实验表明,在报告的前 10% 条警报中,Z-Ranking 能够比随机排序算法发现平均多 3 ~7 倍的错误。Jung 等人[24]使用另外一种统计方法——贝叶斯统计,来计算警报为正报的概率,然后根据这些概率对警报进行排序,支持用户设置概率阈值,只有超过阈值的警报才会报告给用户。Xypolytos 等人[25]提出了一个基于工具性能统计数据的排序框架。该框架的前提是要完成一个基准测试,在基准测试中,使用精确度和召回率衡量静态分析工具针对不同类型的软件缺陷的分析性能,然后用置信度得分来表示并据此对警报进行组合和排序。初步实验结果表明,该方法在提高工具效率和有用性方面具有一定潜力,但基准测试对排序结果的影响较大,基准测试数据集的选择直接影响工具的置信度得分,从而导致最终报告中警报排序结果并不准确。

Kremenek 等人[26]在进一步的研究中,将警报之间的相关性用于警报排序。经过分析来自两个大型系统(Linux 和某商业系统)的历史漏洞数据中的聚类行为,他们发现漏洞和误报通常按代码位置聚集,这种聚集可以发生在函数、类等级别,因此利用聚类可以大幅提高警报排序技术。并且他们将反馈机制与聚类相结合,提出了反馈排序算法。当测试人员在验证了部分警报的正确性后,剩余的警报可以根据测试人员的反馈结果动态更新排名。此方案克服了 Z-Ranking[5]不会随检查结果更新排名的缺点,充分利用了警报间的相关性。

Heckman[27]也将开发人员的反馈用于警报自适应排序。首先使用类型精度、代码位置和是否产生测试失败这三个因素来计算警报为正报的概率,并根据此

概率为警报排序。其中类型精度是指基于警报类型,该警报是正报的概率,代码位置是指基于警报在代码中的位置,该警报是正报的概率,是否产生测试失败是指针对某个警报测试用例是否会失败。在当前排序的基础上,开发人员以过滤警报的方式提供反馈,剩余的警报将根据反馈自动修改排名。Shen 等人[28]在设计基于静态分析工具 FindBugs 的警报排序策略时也将反馈机制纳入排序过程。首先,通过统计数据,为每个缺陷模式分配预定义的缺陷可能性,并根据缺陷可能性来为缺陷报告初始化排序。然后,通过测试人员的反馈来自适应地处理初始排名,这个过程是自动执行的,基于相同的缺陷模式的错误报告的相关性。通过在三个广泛使用的 Java 项目(AspectJ、Tomcat、Axis)上进行实验,结果表明,该策略在精度、召回率和 f1 得分方面明显优于 FindBugs 对警报的原始排序。

Boogerd 和 Moonen[29]提出使用代码执行可能性来确定警报的优先级。对于每个警报涉及的代码位置,计算其执行可能性。如果该位置很可能被执行,那么警报就会获得高优先级。如果该位置不太可能被执行,则警报的优先级较低。这项技术可以使测试人员将注意力放在具有高执行可能性位置的警报上。然而,具有低执行可能性位置的警报也可能很重要,并且正是由于该位置执行概率较低,其中的严重错误更难以被检测到。

Kim 和 Ernst[30-31]提出基于历史的警报排序方法。他们通过分析软件更改历史,为每个警告类别计算生命周期,即警报出现与消失之间的时间。如果某类警报的生命周期较短,也就是说该类警报会很快被开发人员修复,则说明该类别中的警报很重要,在报告中应放在前列。如果一个警报长时间没有被移除,或许说明它不值得修复,那么该类别的警报可以被忽略。在三个开源项目(Columba、Lucene、Scarab)上的实验表明,基于历史的警报排序算法可以将顶部警报的正报精确度分别提高 17%、25% 和 67%,并且该排序算法是通用的,可以用于任何警报。但是如果能将分析工具对警报设置的优先级与该算法为警报设置的优先级结合起来,应该可获得更高精度。Williams 和 Hollingsworth[32]也提出了一个类似的排序方案,该方案基于缺陷的修复记录和在源代码存储库中挖掘的信息。

Ribeiro 等人[33]为了实现排序技术尽可能通用,将排序所需数据限制在了报告本身,而不使用其他任何信息。为了获得良好且充足的数据,首先,将不同工具生成的警报转换为统一格式,并且为每个警报贴上标签,以区分它们是正报还是误报。然后,将每个警报关联到特定的特征,这些特征组成一个数据集用来训练预测模型。特征中的触发警报的工具名称、警告的严重等级等可以通过一次检查一个警报来获取,但其他特征需要处理整个汇总的报告才能获得,其中包括:

(1)报告中指出同一位置有缺陷的次数;

(2)给定警报位置周围出发的警报数量;

(3)警报所指的软件漏洞类别;

(4)其他分析工具在同一位置生成警报;

(5)为当前警报所指向的同一文件生成的警报数。

使用 AdaBoost 算法训练几个弱分类器,经过组合得到一个强分类器。在对测试数据集进行分类时,达到了 0.805 的准确率和 0.958 的召回率。

上文中三类处理方法的总结与对比如表 1 所示。

表 1 程序静态分析报告处理方法对比

| 思想 | 简介 | 技术 | 方法 | 优点 | 缺点 |
|---|---|---|---|---|---|
| 基于融合 | 为多个工具产生的报告提供统一的警报描述格式 | 数据融合 | 六元组序列、通用规范 | 风格统一 | 不关心警报正确性 |
| | | 数据融合与统计分析 | 统一格式+主成分分析 | | |
| 基于分类 | 将警报分为两类(正报和误报)或多类 | 机器学习 | K-Nearest Neighbor<br>RandomForest<br>Support Vector Machines<br>Logic Regression<br>Classification And Regression Tree<br>Extreme Gradient Boosting | 大大减少了警报审查的工作量 | 分类准确性至关重要 |
| | | 其它 | 缺陷关联、误报模式 | | |
| | | 统计分析 | 贝叶斯统计 | | |
| 基于排序 | 将警报按照正报可能性由高到低排序 | 其他 | 基于工具性能<br>基于警报相关性<br>自适应排序<br>代码执行可能性<br>基于警报生命周期 | 提高漏洞修复效率 | 不会减少警报数量 |

## 5　静态分析报告处理的发展方向

随着报告处理技术的不断探索,静态分析在软件测试中发挥着越来越重要的作用。虽然目前静态分析仍面临诸多挑战,但相信经过研究人员更多的尝试与努力,静态分析会有更广阔的应用前景。下面对静态分析告处理领域未来的发展和研究方向提出一些展望:

(1)方法组合。虽然目前有许多报告处理方法,但不同类型的方法并非独立,如何将它们完美的组合在一起,使静态分析工具的优势发挥的更充分,是未来需要考虑的问题。

(2)思路拓展。当前的处理方法大都利用警报本身或代码中所包含的信息,没有考虑到静态分析工具的分析规则是否可以为警报处理做出贡献。由于静态分析工具基于规则库实现缺陷的识别,并且分析报告中的警报描述与规则也有所关联,或许未来可以利用分析规则探索一条警报处理的新道路。

(3)技术更新。当前一些热门技术,例如机器学习,不仅被广泛应用于程序静态分析领域,在分析报告优化方面机器学习技术也取得了较好的效果,但仍存在警报大量重复、警报正确性难以确定和警报排序不合理等问题。或许未来可以考虑从其他技术出发,研究一种全新的报告处理方法。

## 6　结束语

首先,对静态分析技术进行了简要介绍,指出了当前静态分析工具输出报告存在的主要问题。接着,总结了近年来国内外研究人员提出的对静态分析工具所产生的警报的各种处理方法,根据基本思想将其主要分为三类:基于融合、基于分类、基于排序,并分析了这三类方法的优点和缺点。最后,对静态分析报告处理技术未来的发展方向进行了展望并对全文进行了总结,希望能为读者在研究主题或开发方法时提供指引。

**参考文献:**


[1] 张　健,张　超,玄跻峰,等. 程序分析研究进展[J]. 软件学报,2019,30(1):80-109.

[2] 梅　宏,王千祥,张　路,等. 软件分析技术进展[J]. 计算机学报,2009,32(9):1697-1710.

[3] LANDI W. Undecidability of static analysis[J]. ACM Letters on Programming Languages and Systems,1992,1(4):323-337.

[4] 陈　超,李　俊,孔德光. 基于数据融合的源代码静态分析[J]. 计算机工程,2008,34(20):66-68.

[5] KREMENEK T,ENGLER D. Z-ranking:using statistical analysis to counter the impact of static analysis approximations[C]//Static analysis:10th international symposium. San Diego:Springer-Verlag,2003:295-315.

[6] HECKMAN S,WILLIAMS L. A systematic literature review of actionable alert identification techniques for automated static code analysis[J]. Information and Software Technology,2011,53(4):363-387.

[7] MUSKE T,SEREBRENIK A. Survey of approaches for handling static analysis alarms[C]//2016 IEEE 16th international working conference on source code analysis and manipulation. Raleigh:IEEE,2016:157-166.

[8] 肖　庆,杨朝红,宫云战. 提高静态缺陷检测精度方法[J]. 计算机辅助设计与图形学学报,2010,22(11):2037-2044.

[9] 王晓萌,张　涛,辛　伟,等. 深度学习源代码缺陷检测方法[J]. 北京理工大学学报,2019,39(11):1155-1159.

[10] 赵云山,宫云战,刘　莉,等. 提高路径敏感缺陷检测方法的效率及精度研究[J]. 计算机学报,2011,34(6):1100-1113.

[11] KONG D,ZHENG Q,CHEN C,et al. ISA:a source code static vulnerability detection system based on data fusion[C]//International conference on scalable information systems. Suzhou:[s. n.],2007:55-61.

[12] DANG B H. A Practical Approach for Ranking Software Warnings from Multiple Static Code Analysis Reports[C]//SoutheastCon 2020. Raleigh:IEEE,2020:1-7.

[13] MENG N, WANG Q, WU Q, et al. An app-roach to merge results of multiple static analysis tools (short paper)[C]//2008 The eighth international conference on quality software. Oxford:IEEE,2008:169-174.

[14] 张　丽. 代码静态分析工具的能力评估与集成优化技术研究[D]. 长沙:国防科学技术大学,2017.

[15] ALIKHASHASHNEH E A,RAJE R R,HILL J H. Using machine learning techniques to classify and predict static code analysis tool warnings[C]//2018 I-EEE/ACS 15th international conference on computer systems and applications. Aqaba:IEEE,2018:1-8.

[16] FLYNN L,SNAVELY W,SVOBODA D,et al. Prioritizing alerts from multiple static an-alysis tools, using classify-cation models[C]//2018 IEEE/ACM 1st international workshop on software qualities and their dependencies. Gothenburg:IEEE,2018:13-20.

[17] YOON J,JIN M,JUNG Y. Reducing false alarms from an industrial-strength static analyzer by SVM[C]//2014 21st Asia Pacific software engineering conference. Jeju: IEEE,2014:3-6.

[18] RUTHRUFF J,PENIX J,MORGENTHALER J,et al. Predicting accurate and actionable static analysis warnings[C]//2008 ACM/IEEE 30th international conference on software engineering. Leipzig:IEEE,2008:341-350.

[19] 张大林,金大海,宫云战,等. 基于缺陷关联的静态分析优化[J]. 软件学报,2014,25(2):386-399.

[20] LEE W,LEE W,KANG D,et al. Sound non-statisti-cal clustering of static analysis alarms[J]. ACM Transactions on

Programming Languages and Systems,2017,39(4):1-35.
[21] REYNOLDS Z P,JAYANTH A B,KOC U,et al. Identifying and documenting false positive patterns generated by static code analysis tools[C]//2017 IEEE/ACM 4th international workshop on software engineering research and industrial practice. Buenos Aires:IEEE,2017:55-61.
[22] KOC U,SAADATPANAH P,FOSTER J S,et al. Learning a classifier for false positive error reports emitted by static code analysis tools[C]//Proceedings of the 1st ACM SIGPLAN international workshop on machine learning and programming languages. Barcelona:Association for Computing Machinery,2017:35-42.
[23] HECKMAN S,WILLIAMS L. A model building process for identifying actionable static analysis alerts[C]//2009 international conference on software testing verification and validation. Denver:IEEE,2009:161-170.
[24] JUNG Y,KIM J,SHIN J,et al. Taming false alarms from a domain-unaware C analyzer by a Bayesian statistical post analysis[C]//International static analysis symposium. [s. l.]:Springer,2005:203-217.
[25] XYPOLYTOS A,XU H,VIEIRA B,et al. A framework for combining and ranking static analysis tool findings based on tool performance statistics[C]//2017 IEEE international conference on software quality,reliability and security companion. Prague:IEEE,2017:595-596.
[26] KREMENEK T,ASHCRAFT K,YANG J,et al. Correlation exploitation in error ranking[J]. ACM SIGSOFT Software Engineering Notes,2004,29(6):83-93.
[27] HREMENEK S S. Adaptive probabilistic model for ranking code-based static analysis alerts[C]//29th international conference on software engineering. Minneapolis:IEEE,2007:89-90.
[28] SHEN H,FANG J,ZHAO J. Efindbugs:effective error ranking for findbugs[C]//2011 fourth IEEE international conference on software testing, verification and validation. Berlin:IEEE,2011:299-308.
[29] BOOGERD C,MOONEN L. Prioritizing software inspection results using static profiling[C]//2006 sixth IEEE international workshop on source code analysis and manipulation. Philadelphia:IEEE,2006:149-160.
[30] KIM S,ERNST M D. Prioritizing warning categories by analyzing software history[C]//Fourth international workshop on mining software repositories. Minneapolis:IEEE,2007:27.
[31] KIM S,ERNST M D. Which warnings should I fix first?[C]//Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on foundations of software engineering. Dubrovnik:ACM,2007:45-54.
[32] WILLIAMS C C,HOLLINGSWORTH J K. Automatic mining of source code repositories to improve bug finding techniques[J]. IEEE Transactions on Software Engineering,2005,31(6):466-480.
[33] RIBEIRO A,MEIRELLES P,LAGO N,et al. Ranking warnings from multiple source code static analyzers via ensemble learning[C]//Proceedings of the 15th international symposium on open collaboration. Sweden:[s. n.],2019:1-10.