文章编号:1671-4598(2024)12-0280-08 DOI:10.16526/j.cnki.11-4762/tp.2024.12.040 中图分类号:TP311.5 文献标识码:A

面向静态分析的软件测试工具评估方法研究

曾福萍[1], 王泽宇[1], 李宇佳[2], 王杰凯[3]
(1. 北京航空航天大学 可靠性与系统工程学院, 北京 100191;
2. 中国电力科学研究院有限公司, 北京 100192;
3. 百度在线网络技术有限公司, 北京 100089)

摘要: 软件是否可靠运行将直接影响系统的可靠运行, 急需有手段保障软件质量; 静态分析因具有全自动运行、能更早实施、不需要执行程序等特点, 在软件测试领域得到了广泛的使用, 已成为保障软件质量的重要手段; 测试工具可以极大地提高软件测试的效率; 目前面向静态分析的软件测试工具数量众多, 不同的工具具有不同的特点和缺陷检测能力, 且都存在不低的误报率和漏报率, 如何评估和选择测试工具成为软件静态分析时亟待解决的问题; 基于 CWE 缺陷类型的基准测试集和精确度、召回率、F_1 值、CWE 覆盖率以及 Overall-Score 的评估指标, 详细阐述了面向静态分析的软件测试工具评估流程, 结合 CppCheck、TscanCode 和 Flawfinder 三个开源软件静态分析工具开展了案例应用, 为软件静态分析工具的评估与选择提供指导和参考。

关键词: 软件测试; 静态分析工具; 自动控制软件; 评估; 召回率

Study on Evaluation Method of Software Testing Tools for Static Analysis

ZENG Fuping[1], WANG Zeyu[1], LI Yujia[2], WANG Jiekai[3]
(1. School of Reliability and Systems Engineering, Beihang University, Beijing 100191, China;
2. China Electric Power Research Institute, Beijing 100192, China;
3. Baidu Online Network Technology Co., Ltd., Beijing 100089, China)

Abstract: Whether the software runs reliably will directly affect the reliable operation of the system, and there is an urgent need to have means to guarantee software quality. Static analysis has been widely used in the field of software testing due to its features such as fully automatic operation, earlier implementation, no need to execute code, etc. It has become an important means to guarantee software quality. Testing tools can greatly improve the efficiency of software testing. At present, there are many software testing tools for static analysis, different tools have different characteristics and defect detection capabilities, and all of them have not low false positive rate and false negative rate, how to evaluate and select the testing tool has become an urgent problem to be solved. Based on the benchmark test set of CWE, and the five evaluation metrics of precision, recall, F_1, CWE coverage and Overall-Score, the evaluation process of software testing tools for static analysis is elaborated. A case study was conducted using three open-source software static analysis tools, CppCheck, TscanCode, and Flawfinder, which provides guidance and reference for the evaluation and selection of software static analysis tools.

Keywords: software testing; static analysis tool; automation control software; evaluation; recall

0 引言

软件在现代社会的各个领域发挥着至关重要的作用, 软件是否可靠运行将直接影响系统的可靠运行, 例如电网自动控制软件存在缺陷会导致设备误控、联络线功率大幅波动等问题, 严重危害电网安全稳定运行。因此, 保障软件质量是至关重要的。软件静态分析[1-2]是一种常用的静态测试技术, 通常用于自动化的程序缺陷检测, 可以根据一些先验的缺陷代码模式, 自动化地对代码进行扫描, 检测程序中潜在的缺陷, 例如空指针、死锁、缓冲区溢出、变量未初始化等。软件静态分析因其具有全自动运行、能更早实施、不需要执行程序等特点, 在业界得到了广泛的使用, 成为保障软件质量不可缺少的手段之一。软件静态分析离不开测试工具的支持, 面向静态分析的软件测试工具也称为静态分析工具。从 1979 年第一个广泛使用的静态分析工具 (SAT, static analysis tool) Lint[3] 发布以来, 研究人员在静态分析领域做了大量的工作, 研发了许多静态分析工具, 例如商业工具有 LDRA Testbed、Perforce Klocwork、Parasoft C/C++test、Synopsys Coverity 等; 开源工具有 Flawfinder、FindBugs、SonarQube 等。许多公司[4-6]已经在它们的软件产品中使用静态分析工具, 提升了

收稿日期:2024-07-17; **修回日期**:2024-10-07。
基金项目:国家电网公司科技计划项目(SGLNDLOODKJS250326)。
作者简介:曾福萍(1977-),女,博士,助理研究员。
引用格式:曾福萍,王泽宇,李宇佳,等. 面向静态分析的软件测试工具评估方法研究[J]. 计算机测量与控制,2024,32(12):280-287.

工作效率。

最理想的静态分析工具是能检测到尽可能多缺陷且覆盖尽可能多的缺陷种类，同时误报率最小。目前静态分析工具一方面种类繁多，能检测的缺陷种类不同；另一方面缺陷检测能力都存在漏报率和误报率，且漏报率和误报率不是正相关。每个工具发布时会有一些公开资料，但只能对工具有一个大致了解，很难确定该工具适合分析的缺陷类型以及分析效果，只有试用并对其进行评估，才能充分了解工具的优点、缺点以及实际分析能力，所以测试工具的评估对工具使用人员和开发人员都具有非常重要的意义。

目前在静态分析工具评估方面，文献［7－16］中前人已做了一些探索、研究和工作，但在缺陷类型覆盖数量以及评估方法通用性方面有待改进。本文在前人研究的基础上，围绕工具评估的改进需求，首先明确了面向通用缺陷列表（CWE，common weakness enumeration）缺陷类型的基准测试集，选取了精确度、召回率、F_1 值、CWE 覆盖率以及 Overall-Score 的考虑误报和漏报多角度的评估指标；在此基础上，给出了面向静态分析的软件测试工具评估流程；最后开展了 CppCheck、TscanCode 和 Flawfinder 三个开源软件静态分析工具评估的案例应用，为软件静态分析工具的评估与选择提供指导和参考。

1　面向 CWE 缺陷类型的 JTS 基准测试集

基准测试集是静态分析工具评估的基础。基准测试集的质量直接影响工具评估的结果，静态分析工具评估亟需得到广泛认可的测试集。虽然采用现实世界出现的代码缺陷具有真实性、代表性等优点，但存在没有足够多的缺陷等问题。

CWE 是一种针对软件常见缺陷漏洞的分类列表，包含了共计 924 种常见的计算机软硬件相关的缺陷。美国国家安全局（National Security Agency）软件质量中心（Center for Assured Software）在 CWE 缺陷分类基础上，发布了一个名为 Juliet Test Suite（以下简称 JTS）测试基准集。JTS 提供了一系列的测试用例，这些用例设计用于触发 CWE 中列出的缺陷；同时支持为每个包含的 CWE 单独构建目标，这些新目标允许每个测试用例设计两种版本：好的（good）和坏的（bad）。好的版本不包含缺陷，而坏的版本包含缺陷。JTS 的构建和运行脚本支持自动化测试，并能够记录测试结果。JTS 还具有以下特性，所以优先推荐作为工具评估的基准测试集：

1）免费公开。

2）测试用例结构相对简单，便于分析。

3）经过多版本迭代，各类缺陷得以修复。

4）缺陷类型完整，包含目前常见的各类缺陷。

5）是一个用于评估软件工具能力的测试套件，通过检测工具能否正确识别 CWE 缺陷可以验证和比较不同工具在发现和报告软件缺陷方面的效能。

6）应用较为广泛，相当数量的研究者选择 JTS 作为基准测试集。

JTS 测试集共包括两个部分：一部分涵盖了 C 和 C++ 编程语言的常见缺陷；另一部分则针对 Java 语言的常见缺陷。整个 JTS 的 V1.3 版本基准集面向 C/C++ 编程语言的部分共包含 64 099 个测试用例，包括 Windows 环境中的 12 741个测试用例和 Linux 环境中的 51 358 个测试用例。每个测试用例均为某一类 CWE 缺陷类型的具体实例。整个测试集中包含的文件总数为 105234（包括 .c，.cpp 和 .h 文件），一个测试用例可能包含多个文件。

2　考虑误报和漏报多角度的评估指标

在使用静态分析工具时，重点会关注静态分析工具可以识别多少缺陷、哪些类型的缺陷、缺陷告警是否准确以及综合能力如何。基于此，本文采用表 1 工具评估矩阵，代表静态分析工具在识别代码缺陷方面性能的不同方面。为了从不同角度进一步全面评估工具的性能和效果，选择了精确度、召回率、F_1 值、CWE 覆盖率以及本文提出的 Overall-Score 等对静态分析工具进行评估，有助于确保工具在准确性、全面性、平衡性和行业标准遵循方面达到高水平。

表 1　具评估矩阵

软件报告	有告警	无告警
有缺陷	TP	FN
无缺陷	FP	TN

TP（True Positive）：工具告警且软件实际有缺陷的个数，即真实缺陷个数。

FP（False Positive）：工具告警但软件实际没有缺陷的个数，即误报个数。

FN（False Negative）：工具无告警但软件实际有缺陷的个数，即漏报个数。

TN（True Negative）：工具无告警且软件实际无缺陷的个数，即真实无缺陷个数。

1）精确度反映了被分类器判定的正样本中真正的正样本的比重，计算公式为：

$$Precison = \frac{TP}{TP + FP} \tag{1}$$

精确度回答了“静态分析工具的缺陷告警是否精确”的问题，描述了静态工具的可信程度，反映工具的误报情况。精确度越高，代表静态分析工具给出的告警更有可能确实是缺陷（即该告警的可信度越高）。相反地，若静态工具精确度低，则说明静态工具的告警很可能是错误的，这会使得开发人员花费更多的时间去判别这个告警的真实性。

2）召回率（Recall）反映了被正确判定的正样本占所有正样本的比重，计算公式为：

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

召回率回答了“静态分析工具能识别多少缺陷”的问题，描述了工具告警对于整体缺陷的覆盖程度，反映工具

的漏报情况。召回率越高，代表静态分析工具找到了较多的缺陷，漏报率低。相反地，若召回率低，则说明工具告警不够全面，被测文件遗留较多缺陷未被检测，使得开发人员花费更多的时间去寻找缺陷。

3）F_1是精确度和召回率的调和平均，为综合指标，计算公式为：

$$F_1 = \frac{2 * Precison * Recall}{Precison + Recall} \tag{3}$$

F_1 反映静态分析工具在漏报和误报之间的平衡性。一个工具如果采用激进策略，选择将所有疑似缺陷都进行告警，则该工具尽管拥有非常高的召回率，但其精确度会非常低，在这种情况下，F_1 可以更好地反映工具的缺陷分析能力。

4）CWE 覆盖度（Coverage for CWE）是静态分析工具可以检出的缺陷类型数和测试集缺陷类型总数的比，计算公式为：

$$CWE_{cov} = \frac{CWE\_Tool}{CWE\_JTS} \tag{4}$$

其中：CWE _ Tool 为工具正确缺陷告警的 CWE 类型个数，CWE _ JTS 为测试集缺陷类型总数。

CWE 覆盖度回答了“静态分析工具可以识别哪些类型的缺陷”的问题，是对静态分析工具可以识别缺陷类型数量的评价，若工具在某种类型的测试文件中存在正报，则该工具可以识别该类型的缺陷。

5）Overall-Score 反映了精确度、召回率和 CWE 覆盖度的综合指标。本文借鉴 F_1 的思路，提出 Overall-Score 指标，基于精确度、召回率和 CWE 覆盖度构建，使用调和平均数计算方法（即变量倒数的算术平均数的倒数），计算公式为：

$$Overall\text{-}Score = \frac{3}{\frac{1}{Precison} + \frac{1}{Recall} + \frac{1}{CWE_{cov}}} \tag{5}$$

Overall-Score 回答了“静态分析工具综合能力如何”的问题，融合了工具误报情况的精确度、漏报情况的召回率以及缺陷类型覆盖程度的 CWE 覆盖度。

3 面向静态分析的软件测试工具评估流程

面向静态分析的软件测试工具评估流程如图 1 所示。

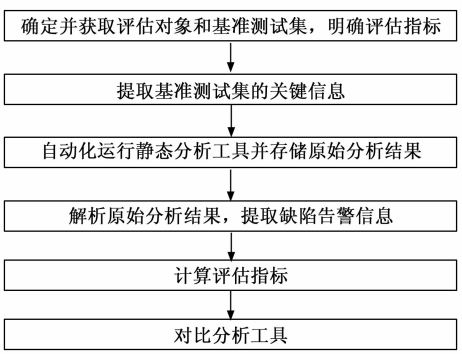

图 1 评估流程

3.1 明确评估对象、基准测试集以及评估指标

工具评估的第一步是明确对哪些静态分析工具基于什么基准测试集进行评估，评估结果用什么指标进行衡量。

3.2 提取基准测试集的关键信息

基准测试集的关键信息包括测试文件数量、测试文件名、是否注入缺陷、缺陷位置等，并对这些信息格式化进行存储。

JTS 基准测试集的关键信息来源于所提供的 XML 文件形式的测试用例，以成对的标签进行数据标注和格式化展示。图 2 为 JTS 基准测试集某个测试用例的 XML 文件示例，第 1 行和第 5 行构成一组“testcase”标签，表明是一个测试用例的相关信息；“testcase”标签内的一组“file”标签（第 2 行和第 4 行），表明该测试用例只包含一个具体的测试文件，而“file”标签内的“flaw”标签表明该测试文件中具体注入缺陷的位置（line 关键字）和注入缺陷的 CWE 类型信息（name 关键字）。

```
1 <testcase>
2   <file path="CWE114_Process_control_w32_char_connect_socket_06.c">
3     <flaw line="128" name=" cwe-114:Process Control" />
4   </file>
5 </testcase>
```

图 2 JTS 基准测试集某个测试用例的 XML 文件示例

根据该测试用例，可以提取基准测试集关键信息如表 2 所示，其中对于包含“flaw”标签的文件（即有注入缺陷的相关信息）用数字 1 表明该测试文件包含缺陷，反之不包含“flaw”标签的文件，用数字 0 表示该测试文件不包含缺陷。

表 2 基准测试集关键信息提取示例

名称	内容
测试文件名	CWE114_Process_Control__w32_char_connect_socket_06.c
测试文件数	1
是否注入缺陷	1
缺陷注入位置	128
CWE 缺陷类型	CWE-114(Process Control)

3.3 基于基准测试集自动化运行静态分析工具

基于基准测试集，运行评估对象，开展软件静态分析，保存原始静态分析结果。由于测试集中被测文件较多，一般通过编程语言实现工具对基准测试集的自动批量执行，提高开展静态分析的效率。

3.4 解析原始分析结果，提取缺陷告警信息

不同静态分析工具的分析结果格式不同，包含的缺陷告警信息也不同，需要基于正则表达式、字符串切分等方式对原始静态分析结果中的是否发现缺陷这个关键信息进行解析。如发现缺陷，进一步提取缺陷告警信息，包括缺陷类型、缺陷所在行数、缺陷告警级别等，并使用统一格式存储于 json 文件。

3.5　计算评估指标

将工具发现的缺陷告警信息与第二步提取与缺陷相关的信息进行对比，统计出不同工具的 TP，FN，FP 三个基本指标和不同 CWE 的覆盖情况，并根据公式进一步计算精确度、召回率、F_1 等评估指标。在各指标统计过程中，进行如下约定：

1）每个测试文件根据行数均匀切分为 20 个代码块。

2）若工具对某个包含缺陷的代码块给出一个或多个缺陷告警，则该工具的 TP 统计值加 1。

3）若工具对某个不包含缺陷的代码块未给出告警，则该工具的 TN 统计值加 1。

4）若工具对某个包含缺陷的代码块未给出缺陷告警，则该工具的 FN 统计值加 1。

5）若工具对某个不包含缺陷的代码块给出 x 个缺陷告警，则认为该工具的 FP 统计值加 x。

6）若对于某 CWE 缺陷类型的所有测试文件，均发现 TP 为 0，则认为缺陷报告可以覆盖该类型的 CWE 缺陷，CWE 覆盖度加 1。

3.6　对比分析工具

根据第五步计算的评估指标结果，对比分析所评估的静态分析工具的优缺点、分析效率及具有的特性等，为静态分析工具的选择提供数据参考。

4　案例应用

4.1　明确评估对象

为了能获取到静态分析工具，选择开源的静态分析工具作为本次案例应用的评估对象。在开源社区中有不少软件静态分析工具，选择了目前应用较为广泛的 CppCheck、TscanCode 和 Flawfinder 三个静态分析工具作为案例应用的评估对象。

4.2　选取测试基准集文件

本次案例应用选取了 Juliet Test Suite v1.3 C/C++面向 Linux 内核环境的测试用例，共计 88 635 个测试文件，包含 91 个不同 CWE 类型、注入缺陷文件 56 062 个（占比 63.25%）、未注入缺陷文件 32 573 个（占比 36.75%）。

通过统计可以观察 88 635 个测试文件在 CWE 类型上的测试用例个数分布情况。如图 3 所示，为测试文件数超过 100 的 44 个 CWE 缺陷类型及其具体测试文件数分布，各个 CWE 类型上的测试文件个数差异较大。其中 CWE122（Heap Based Buffer Overflow）类型测试文件最多，共计 9 628 个，而 CWE377（Insecure Temporary File）类型的测试文件最少，仅 108 个；12 个 CWE 类型的测试文件数大于 3 000；11 个 CWE 类型的测试文件数在［1 000，3 000］之间；21 个 CWE 类型的测试文件数在［100，1 000］之间。

各个 CWE 类型包含缺陷的文件数占该类型总测试文件数比例最低的为 CWE500，占比为 50%，有 48 个 CWE 类型的测试文件全部注入了缺陷，各个 CWE 类型包含缺陷的测试文件平均占比为 82.2%。

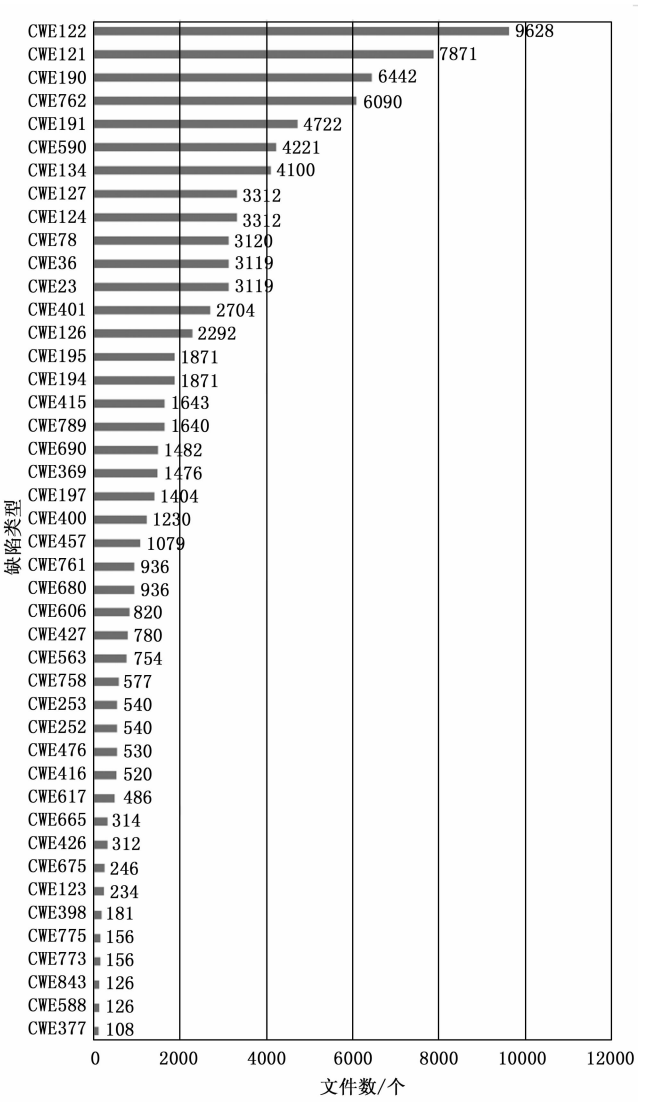

图 3　不同 CWE 类型的测试文件个数分布

4.3　应用过程及结果

在 Ubuntu20.04 环境中，安装部署 CppCheck，Flawfinder 和 TscanCode 三个静态分析工具，对 91 种 CWE 缺陷类型的 88 635 个测试文件进行分析，之后从两个角度对 3 个静态分析工具进行评估分析：一是在 CWE 测试文件上使用精确度，召回率和 F_1 进行评估；二是在 JTS 测试集上，使用精确度、召回率、F_1、CWE 覆盖度和 Overall-Score 进行评估。

4.3.1　CppCheck 评估结果

91 种 CWE 缺陷类型中，CppCheck 工具可以发现 27 种缺陷类型，其中 CWE440 和 CWE562 的测试文件数分别为 1 个和 3 个测试文件，在这两类 CWE 上计算精确度、召回率等指标不具有广泛的表征意义。CppCheck 对其他 25 种 CWE 缺陷类型的评估结果如图 4 所示，横坐标以 F_1 的大小进行确定，左侧的 CWE 类型的 F_1 越大，右侧越小。

由图 4 不同 CWE 缺陷类型 CppCheck 评估结果可发现以下现象：

1）整体趋势方面，按照 F_1 的降序将 25 种 CWE 进行排序，发现不同类型的召回率基本上呈现整体的下降趋势，F_1 和召回率的趋势相似，但精确度没有明显的趋势在不同位置都出现了较高值。

2）F_1 方面：CppCheck 在 CWE762 类型上取得最大值 44.98%；在 CWE195 类型上取得最小值 1.89%；共计 7 个 CWE 缺陷类型的 F_1 值大于 20%，占 25 种类型的 28%；F_1 在［10%，20%］区间的 CWE 缺陷类型同样有 7 种；在剩余 11 种 CWE 缺陷类型上，CppCheck 的 F_1 只有个位数。

3）精确度方面：CppCheck 在 CWE369 类型上取得最大值 87.50%；在 CWE672 类型上取得最小值 2.53%；CppCheck 共在 7 个 CWE 缺陷类型的精确度大于 60%，但这 7 个 CWE 缺陷类型与 F_1 值大于 20%的 7 个 CWE 缺陷类型重合度较低；共有 8 种 CWE 缺陷类型的精确度低于 20%。

4）召回率方面：CppCheck 在 CWE843 类型上取得最大值 37.50%，在 CWE195 类型上取得最小值 0.96%；CppCheck 仅在 10 种 CWE 缺陷类型上的召回率超过 10%；在 CWE194 和 CWE195 两种类型上，CppCheck 的召回率低于 1%。

根据以上信息，重点分析了 CppCheck 在 CWE762、CWE843 和 CWE369 这 3 个 CWE 缺陷类型上的情况。

在 CWE762 缺陷类型中，共包含 6 090 个测试文件，共有 3 562 个测试文件被注入缺陷，而在该类型中，CppCheck 的精确度和召回率分别为 74.45%和 32.23%，分别在精确度和召回率中排到第四位和第二位，这说明 CppCheck 对 CWE762 这一类缺陷的分析报告具有非常高的可信度，同时相较于其他类型缺，该类缺陷遗漏较少，这些表现也从 F_1 的最高值上进行了综合体现。

在 CWE369 缺陷类型中，尽管 CppCheck 的精确度高达 87.50%，但召回率仅有 1.62%，这说明 CppCheck 对这一类的缺陷采用了一个泛用性较差但识别效果非常优秀的缺陷判别规则，使得 CppCheck 对于该类缺陷的可信度非常高，但也会产生较多的遗漏情况，这也导致了平衡性指标 F_1 的 3.18%低值。

在 CWE843 缺陷类型中，CppCheck 具有 37.50%的最大召回率，但精确度仅仅 17.86%，这反映了 CppCheck 对这一类缺陷采用了较为宽松的判别策略，给出了较多的缺陷告警从而提高了召回率，但这些缺陷告警中错报较多，可信度相对较低。

4.3.2 TscanCode 评估结果

91 种 CWE 缺陷类型中，TscanCode 工具可以发现 13 种缺陷类型，对 13 种 CWE 缺陷类型的具体评估结果如图 5 所示，横坐标以 F_1 的大小进行确定，左侧的 CWE 类型的 F_1 越大，右侧越小。

由图 5 不同 CWE 缺陷类型 TscanCode 评估结果，可以发现以下情况：

1）整体趋势方面：在按照 F_1 的降序将 13 种 CWE 缺陷类型进行排序后，发现不同类型的召回率也呈现整体的下降趋势，F_1 和召回率的整体趋势相同。但精确度则没有明显的趋势，在不同位置都出现了较高值。

2）F_1 方面：TscanCode 在 CWE476 类型上取得最大值 41.55%；在 CWE690 类型上取得最小值 0.63%；仅 2 个 CWE 缺陷类型的 F_1 值大于 20%；F_1 在［10%，20%］区间的 CWE 缺陷类型只有 4 种；在剩余 7 种 CWE 缺陷类型

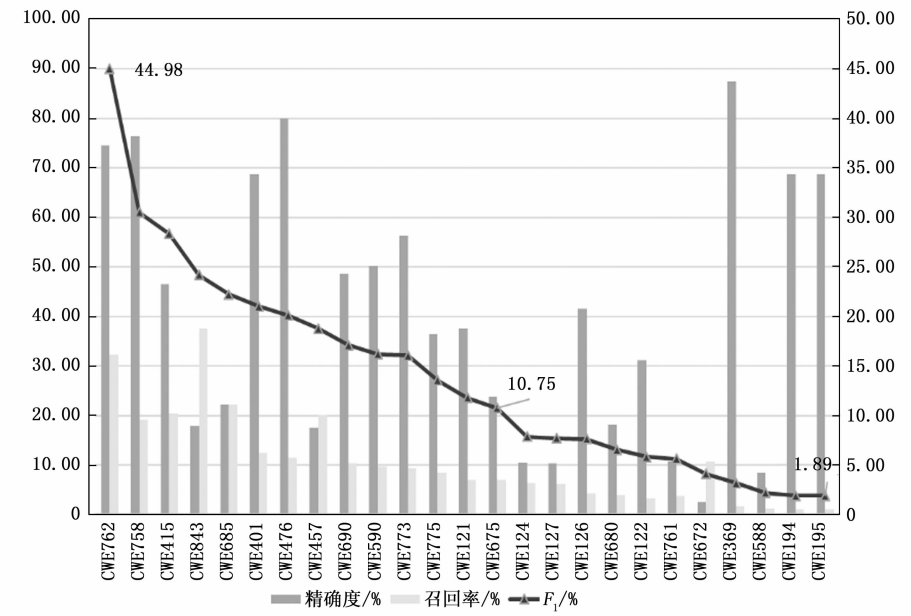

图 4 不同 CWE 缺陷类型 CppCheck 评估结果

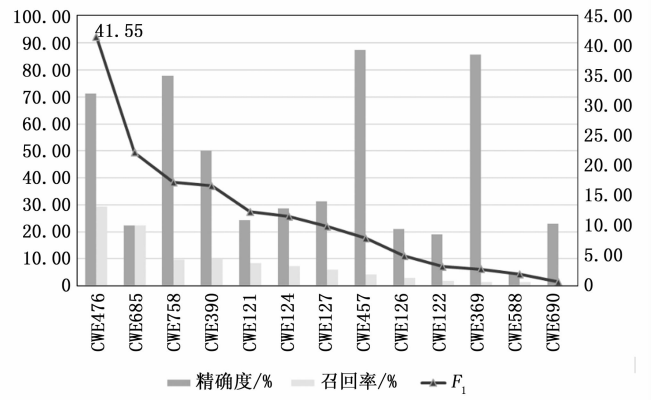

图 5　不同 CWE 缺陷类型 TscanCode 评估结果

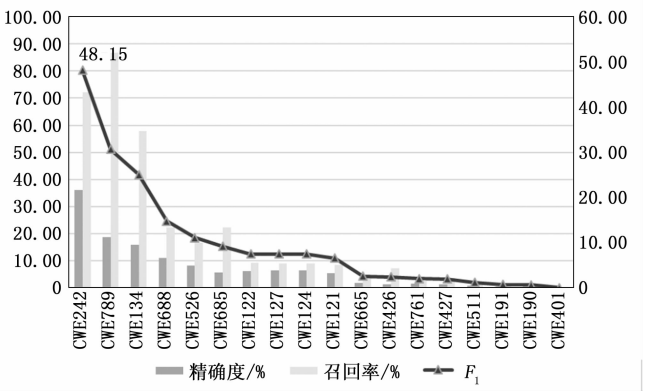

图 6　不同 CWE 缺陷类型 Flawfinder 评估结果

上，TscanCode 的 F_1 平均值仅 4.46%。

3）精确度方面：TscanCode 在 CWE457 类型上取得最大值 87.50%；在 CWE588 类型上取得最小值 4.17%；TscanCode 共在 4 个 CWE 缺陷类型的精确度大于 70%；只有 2 种 CWE 缺陷类型的精确度低于 20%。

4）召回率方面：TscanCode 在 CWE476 类型上取得最大值 29.31%，在 CWE690 类型上取得最小值 0.32%；TscanCode 仅在 3 种 CWE 缺陷类型上的召回率超过 10%；在 CWE122、CWE369 和 CWE588 在 3 种类型上，TscanCode 的召回率仅有 1%。

根据以上信息，重点分析了 TscanCode 在 CWE476、CWE588 和 CWE369 这 3 个 CWE 缺陷类型上的情况。

在 CWE476 缺陷类型中，TscanCode 的召回率和 F_1 均为最高值，分别为 41.55%和 29.31%，而 71.33%的精确度，也排到了第四位，这说明 CppCheck 对 CWE476 这一类缺陷的分析报告的可信度相当高，同时相比于其他类型，在该类缺陷的遗漏数量最少。

在 CWE369 缺陷类型中，TscanCode 的精确度高达 85.71%，虽然不是最高值，但相差也仅 1.79%，而在高精确度的情况下，召回率仅有 1.39%，而 CppCheck 静态分析工具在 CWE369 这一类缺陷下，表现和 TscanCode 非常相似。这说明 TscanCode 对这一类缺陷的处理办法采用了和 CppCheck 相同的处理逻辑，通过一个泛用性较差但识别效果非常优秀的缺陷判别规则，谨慎地给出可信度较高的缺陷报告，但这确实导致了较多的缺陷遗漏情况。

在 CWE588 缺陷类型中，TscanCode 具有 4.17%的最低精确度和倒数第二低的召回率 1.25%，错报和漏报情况非常严重。这说明 TscanCode 难以分析这一类缺陷，对这一类缺陷的判别策略亟待改进。

4.3.3　Flawfinder 评估结果

91 种 CWE 缺陷类型中，Flawfinder 工具可以发现 18 种缺陷类型。Flawfinder 对这 18 种 CWE 缺陷类型的具体评估结果如图 6 所示，横坐标以 F_1 的大小进行确定，左侧的 CWE 类型的 F_1 越大，右侧越小。

由图 6 不同 CWE 缺陷类型下 Flawfinder 的评估结果，可以发现以下情况：

1）整体趋势方面：在按照 F_1 的降序将 18 种 CWE 缺陷类型进行排序后，发现不同类型的精确度和召回率同样也呈现整体的下降趋势，精确度、召回率和 F_1 的整体趋势相同。这一点与前两个工具有一定差异。

2）F_1 方面：Flawfinder 在 CWE242 类型上取得最大值 48.15%；在 CWE401 类型上取得最小值 0.06%；仅 3 个 CWE 缺陷类型的 F_1 值大于 20%；F_1 在［10%，20%］区间的 CWE 缺陷类型只有两种；在剩余 13 种 CWE 缺陷类型上，Flawfinder 的 F_1 平均值仅 3.88%。

3）精确度方面：Flawfinder 在 CWE242 类型上取得最大值 36.11%；在 CWE401 类型上取得最小值 0.06%；Flawfinder 仅在 4 个 CWE 缺陷类型的精确度大于 10%。

4）召回率方面：Flawfinder 在 CWE789 类型上取得最大值 86.67%，在 CWE401 类型上取得最小值 0.06%；Flawfinder 共在 6 种 CWE 缺陷类型上的召回率超过 10%；但同样在 3 种 CWE 缺陷类型上，Flawfinder 的召回率低于 1%。

根据以上信息，重点分析了 Flawfinder 在 CWE242、CWE789 和 CWE401 这 3 个 CWE 缺陷类型上的情况。

在 CWE242 缺陷类型中，Flawfinder 的精确度和 F_1 均为最高值，而 72.22%的召回率，也排到了第二位，但是这一类缺陷仅有 18 个测试文件，且这 18 份文件均注入了缺陷，因此尽管 Flawfinder 对 CWE242 这一类缺陷的分析结果较好，但在实际应用过程中，还需要更多的检验。

在 CWE789 缺陷类型中，Flawfinder 的召回率高达 86.67%，为最高值，但在高召回率的情况下，精确度仅有 18.65%，这说明 Flawfinder 对这一类的缺陷的处理办法采用了非常宽松的判断标准，给出了大量的缺陷报告，尽管这些缺陷报告覆盖了较多的真实缺陷，但同样造成非常严重的错报情况。

在 CWE401 缺陷类型中，Flawfinder 精确度和召回率和 F_1 均为最低值。这说明 Flawfinder 对这一类缺陷的缺陷分析结果不值得信任。

4.3.4 静态分析工具综合评估结果

将工具针对所有 CWE 缺陷类型的分析结果进行统计 *TP*、*FP*、*FN* 和可覆盖 CWE 缺陷类型数，得到如表 3 所示结果。根据相关指标公式，计算精确度、召回率、F_1、CWE 覆盖度和 Overall-Score，静态分析工具综合评估对比结果如图 7 所示。

表 3 静态分析工具综合评估指标

	CppCheck	TscanCode	Flawfinder
TP	3 266	1 053	3 713
FP	6 466	2 658	78 012
FN	53 064	55 277	52 617
CWE 覆盖数/个	27	13	18
精确度/%	32.04	29.43	4.61
召回率/%	5.80	1.64	6.71
F_1/%	9.82	3.10	5.46
CWE 覆盖度/%	25.93	9.88	17.28
Overall-Score/%	12.38	4.02	7.08

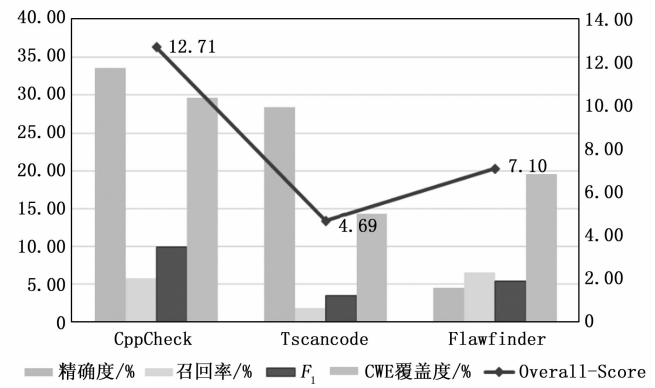

图 7 静态分析工具综合评估对比

在精确度方面，CppCheck 工具的精确度最高，达到 33.56%，TscanCode 次之，为 28.38%，而 Flawfinder 的精确度最低，仅 4.54%。但观察具体正报数 *TP* 和误报数 *FP* 可以发现 Flawfinder 的 *TP* 和 *FP* 是最高的，两个基础指标都远远高于 TscanCode，而 CppCheck 则位于这两个静态分析工具之间。

在召回率方面，精确度较高的 CppCheck 和 TscanCode 的召回率分别为 5.8% 和 1.87%，要低于 Flawfinder 的 6.59%。观察具体的基础指标可以发现，3 个静态分析工具在漏报数上差距不大，但由于 *TP* 值的明显差距，使得召回率上相差明显。

在 CWE 覆盖度上，CppCheck 可以覆盖的 CWE 缺陷类型最多，覆盖度达到 29.67%，最低的 TscanCode 仅覆盖 13 种 CWE 缺陷类型，覆盖度为 14.29%。

在平衡性指标 F_1 和 Overall-Score 方面，均是 CppCheck 取得最高，Flawfinder 次之，TscanCode 最低。

根据以上信息，可以发现：

1）CppCheck 工具在综合评估中表现最好，整体的缺陷分析性能在 3 个静态分析工具中最佳。CppCheck 在保持较高的精确度的情况下，其召回率相较于其他静态分析工具也并不属于低值，而 CWE 缺陷类型覆盖范围也较广。总的来说，CppCheck 较为全面，没有明显的缺点，缺陷分析性能较好。

2）TscanCode 工具则体现出比较谨慎的缺陷判定，给出的缺陷告警数是 3 个静态分析工具中最少的，TscanCode 更加关注于向使用者提供正确的缺陷报告，尽可能提高缺陷报告的可信度，但这也导致了 TscanCode 遗漏的缺陷报告是最多的（最低的召回率），同时也难以覆盖较多的 CWE 缺陷类型。但总的来看，TscanCode 的牺牲召回率，提高精确度的这一思路并没有较好的实现，精确度并没有达到很高的值，反而由于极其严重的漏报导致整体的 Overall-Score 值最低。

3）Flawfinder 工具与其他两个静态分析工具相比，整体缺陷判定准则较为宽松，整体的缺陷报告数量也是 3 个静态分析工具中最多的，这也可以看出其开发者的主要目标是希望尽可能多地找到目标程序中的所有潜在缺陷，其最高的召回率也是这一目标的体现。但大量的缺陷报告也意味着其中蕴含的错误报告较多，极大降低了 Flawfinder 的精确度。但总的来看，Flawfinder 和 TscanCode 类似，都采用了专注于改进误报或漏报，同时选择放弃另一个，但是都同样没有取得较好的效果，Flawfinder 召回率的提升有限，同时没有扩大 CWE 缺陷类型的覆盖范围，反而导致了精确度的下降，这些也从其 7.10% 的 Overall-Score 上进行了体现。

5 结束语

本文围绕软件静态分析工具评估需求，从评估基准测试集、评估指标、评估流程、案例应用等方面展开了研究，实现了对静态分析工具的系统评估。1）选择 CWE 软件缺陷集 Juliet Test Suite 作为评估基准测试集；2）使用精确度、召回率、F_1 常规评估指标，对 3 个开源静态分析工具 CppCheck、TscanCode、Flawfinder，在不同 CWE 缺陷类型下的缺陷分析能力进行初步评估；3）通过文中所提指标 Overall-Score，对工具的缺陷分析能力进行综合性量化评估。对比分析 3 个静态分析工具的评估结果，CppCheck 工具在综合评估中表现最好，Overall-Score 指标能体现不同工具的综合缺陷检测性能的差异，为静态分析工具的选择提供了支撑。后续将评估更多可用的静态分析工具，进一步验证 Overall-Score 指标对静态分析工具综合评估效果，以及评估基准测试集和评估流程的有效性。

参考文献：

[1] RAMASAMY S, SINGH A, SINGAL D. Enhancing the security of C/C++ programs using static analysis [J]. Indian Journal of Science and Technology, 2016, 9 (44): 1-4.

[2] SUBBURAJ R, RAIKAR P U, SHRUTHI S P. Static analysis of security vulnerabilities in C/C++ applications [J]. Indian Journal of Science and Technology, 2016, 9 (20): 1-6.

[3] DARWIN I F. Checking C programs with lint [M]. America: OReilly Media, Inc., 1988.

[4] BALL T, COOK B. SLAM and static driver verifier: technology transfer of formal methods inside microsoft [C] //Integrated Formal Methods, 2004: 1-20.

[5] AYEWA N, PUGH W. The google findBugs fixit [C] //19th International Symposium on Software Testing and Analysis, 2010: 241-252.

[6] BALL T, BOUNIMOVA E, LEVIN V, et al. Efficient evaluation of pointer predicates with Z3 SMT solver in SLAM2 [J]. Discover the World's Research, 2010: 1-8.

[7] MANTERE M, UUSITALO I, RONING J. Comparison of static code analysis tools [C] //Third International Conference on Emerging Security Information, Systems and Technologies, 2009: 15-22.

[8] MAMUN M A A, KHANAM A, GRAHN H, et al. Comparing four static analysis tools for Java concurrency bugs [C] //Third Swedish Workshop on Multi-Core Computing (MCC-10), 2010: 2-7.

[9] MELEAN R K. Comparing static security analysis tools using open source software [C] //IEEE Sixth International Conference on Software Security and Reliability Companion, 2012: 68-74.

[10] DIAZ G, BERMEJO J R. Static analysis of source code security: assessment of tools against SAMATE tests [J]. Information and Software Technology, 2013, 55 (8): 1462-1476.

[11] IMPARATO A, MAIETTA R R, SCALA S, et al. A comparative study of static analysis tools for AUTOSAR automotive software components development [C] //IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2017: 65-68.

[12] MAHMOOD R, MAHMOUD Q. H. Evaluation of static analysis tools for finding vulnerabilities in Java and C/C++ source code [EB/OL]. 2018, ArXiv: 1805.09040.

[13] KAUR A, NAYYAR R. A comparative study of static code analysis tools for vulnerability detection in C/C++ and Java source code [J]. Procedia Computer Science, 2020 (171): 2023-2029.

[14] LENARDUZZI V, LUJAN S, SAARIMAKI N, et al. A critical comparison on six static analysis tools: detection, agreement, and precision [EB/OL]. 2021, ArXiv: 2101.08832.

[15] DESAI V V, JARIWALA V J. Comprehensive empirical study of static code analysis tools for C language [J]. International Journal of Intelligent Systems and Applications in Engineering, 2022, 10 (4): 695-700.

[16] MIDYA ALQARADAGHI, TAMáS KOZSIK. Comprehensive evaluation of static analysis tools for their performance in finding vulnerabilities in Java code [J]. IEEE Access, 2024 (12): 55824-55842.

(上接第 262 页)

[6] 李振华，王泓懿，李　洋，等. 大规模复杂终端网络的云原生强化设计 [J]. 计算机研究与发展，2024，61 (1)：2-19.

[7] 郭妍. 新时期医院信息化建设研究 [J]. 网络安全和信息化，2024 (3)：15-17.

[8] 滕晓燕，朱立峰，赵　艳. 高质量发展背景下医院运营管理信息化研究热点、演化与趋势 [J]. 中国数字医学，2024，19 (6)：1-10.

[9] 孙　谦，肖雪雯，胡俊峰，等. 医学影像诊断学自主学习系统的设计与实践 [J]. 计算机测量与控制，2019，27 (5)：160-163.

[10] 李坤成. 加强人工智能深入学习在医学影像学临床应用领域的研究 [J]. 中国医学影像技术，2019，35 (12)：1769-1770.

[11] 王乾梁，石宏理. 基于改进 YOLO V3 的肺结节检测方法 [J]. 中国医学物理学杂志，2021，38 (9)：1179-1184.

[12] CAO Z W, ZHANG Y C, GUAN J H, et al. Link weight prediction using weight perturbation and latent factor [J]. IEEE Transactions on Cybernetics, 2022, 52 (3): 1785-1797.

[13] 樊志杰，郑长松，曹志威. 基于动态策略的移动警务终端安全管控系统的设计与实现 [J]. 计算机测量与控制，2021，29 (6)：219-223.

[14] SEBBASTIAN B, DOMINIQUE M, KLAUS-ROBERT M, et al. Deep neural networks for no-reference and full-reference image quality assessment [J]. IEEE Transactions on Image Processing, 2019, 27 (1): 206-219.

[15] 樊志杰，胡正梁，熊已兴，等. 基于单向光的数据安全传输控制系统的设计与实现 [J]. 计算机测量与控制，2021，29 (2)：103-107.

[16] 曹志威，尹心明，杨金云，等. 基于分布式计算的证书应用审计系统 [J]. 信息网络安全，2018，18 (9)：30-34.

[17] LONG Y, WU M, LIU Y, et al. Graph contextualized attention network for predicting synthetic lethality in human cancers [J]. Bioinformatics, 2021, 37 (16): 2432-2440.

[18] 张福俊，庄　晓，李玉华，等. 基于 CiteSpace 的计算机视觉领域研究热点与前沿分析 [J]. 软件导刊，2020，19 (11)：272-278.

[19] ZHANG J S, ZHANG Y, KANG J Y, et al. Potential transmission chains of variant B.1.1.7 and co-mutations of SARS-CoV-2 [J]. Cell Discovery, 2021, 7 (44): 1-10.

[20] 吴绯红，赵煌旋. 医学影像＋人工智能的发展、现状与未来 [J]. 临床放射学杂志，2022，41 (4)：764-767.

[21] 左　艳，黄　钢，聂生东. 深度学习在医学影像智能处理中的应用与挑战 [J]. 中国图象图形学报，2021，26 (2)：305-315.