

Praca naukowa dotycząca porównania istniejących analyzerów statycznych

Maryia Babinskaya, Alicja Biezychudek, Mariia Saltykova
Vladyslav Khabanets, Tomasz Kosmulski, Maksim Zdobnikau
Uniwersytet Jagielloński

4 kwietnia 2025

Streszczenie

Tło: Analizatory statyczne odgrywają kluczową rolę w zapewnieniu jakości oprogramowania, wykrywając potencjalne błędy i luki w kodzie źródłowym. Pomimo ich szerokiego zastosowania, istnieje potrzeba systematycznego porównania ich skuteczności, wydajności i funkcjonalności.

Cel: Celem niniejszej pracy jest porównanie istniejących analyzerów statycznych pod kątem ich możliwości, ograniczeń i zastosowań w różnych kontekstach programistycznych.

Metody:

Wyniki:

Wnioski:

Słowa kluczowe: analizatory statyczne, jakość oprogramowania

1 Wprowadzenie

Analizatory statyczne są niezbędnym narzędziem w procesie tworzenia oprogramowania, umożliwiając wykrywanie błędów na wczesnych etapach rozwoju. Pomimo ich szerokiego zastosowania, istnieje wiele wyzwań związanych z ich skutecznością i wydajnością. Niniejsza praca ma na celu porównanie istniejących analyzerów statycznych, aby pomóc deweloperom w wyborze najbardziej odpowiedniego narzędzia dla ich potrzeb.

2 Powiązane prace

3 Metodologia

Badanie systematycznie porównywało narzędzia statycznej analizy kodu poprzez:

1. **Projekt bazy danych:** Stworzenie relacyjnego schematu (`metryki`) do przechowywania metryk, narzędzi, języków programowania i ich powiązań. Schemat obejmował 15 tabel znormalizowanych do trzeciej postaci normalnej (3NF), z uwzględnieniem relacji wiele-do-wielu między narzędziami a metrykami.
2. **Gromadzenie danych:** Ekstrakcja metryk (złożoność cykliczna, duplikacje, pokrycie kodu itp.) z 15 narzędzi analitycznych, w tym SonarQube, Qodana, PMD i Lizard. Dane zbierano dla czterech języków programowania (C++, Java, Python, TypeScript) na dwóch poziomach szczegółowości: całych repozytoriów oraz pojedynczych plików źródłowych.
3. **Normalizacja:** Strukturyzacja danych w tabelach (`tools`, `metricNames`, `metricValues`) z zastosowaniem technik transformacji danych, w tym:
 - Standaryzacja nazw metryk między różnymi narzędziami
 - Konwersja jednostek miar do wspólnego formatu
 - Mapowanie podobnych metryk z różnych analizatorów
4. **Automatyzacja:** Opracowanie procedury `InsertMetricValue2` w celu standaryzacji wstawiania metryk. Procedura realizuje:
 - Automatyczne wyszukiwanie ID języka, narzędzia i metryki
 - Walidację spójności danych przed wstawieniem
 - Rejestrację źródła danych (`sourceID`)
 - Obsługę relacji między encjami

4 Przegląd analizatorów

W badaniu uwzględniono następujące narzędzia analityczne:

- **Narzędzia wielojęzyczne:**
 - Qodana (wsparcie dla 60+ języków)
 - SonarQube (analiza jakości kodu i wykrywanie podatności)

- PVS-Studio (C/C++, C, analiza zgodności z MISRA)
- **Narzędzia specjalistyczne:**
 - **Java:** PMD, Checkstyle, JaCoCo (pokrycie kodu)
 - **Python:** PyDev, PyCharm, Pylint
 - **C/C++:** cppdepend, OCLint, Lizard
 - **TypeScript:** FTA, cyclomatic-complexity
 - **Analiza repozytoriów:** FREGE (analiza metadanych projektów open-source)

Kluczowe obserwacje

- **Różnorodność metryk:** Wykryto 7 różnych implementacji złożoności cyklomatycznej i 4 różne podejścia do wykrywania duplikatów kodu. Na przykład:
 - Lizard wymaga minimum 100 tokenów dla uznania fragmentu za duplikat
 - SonarQube stosuje próg 10 linii kodu
- **Specjalizacja:** 60
 - JaCoCo dedykowane wyłącznie dla Javy
 - PyLint specjalizowany w analizie kodu Python
- **Integracje:** 30

5 Analiza metryk jakości kodu

W niniejszej sekcji przedstawiono definicje kluczowych metryk oraz omówiono ich znaczenie w ocenie jakości kodu.

5.1 Metryki złożoności

- **Cyclomatic Complexity (Złożoność cyklomatyczna):** Mierzy liczbę niezależnych ścieżek przez kod, czyli ilość punktów decyzyjnych (np. instrukcji warunkowych, pętli). Wysoka wartość wskazuje, że kod jest bardziej skomplikowany, co utrudnia jego testowanie i utrzymanie.
- **Cognitive Complexity (Złożoność poznawcza):** Ocena trudności zrozumienia kodu przez programistę. Nawet przy niskiej złożoności cyklomatycznej kod może być trudny do zrozumienia, dlatego ta metryka pomaga wskazać fragmenty wymagające refaktoryzacji pod kątem czytelności.

- **NPath Complexity:** Oblicza liczbę możliwych ścieżek wykonania funkcji. Wysoka wartość może wskazywać na zbyt skomplikowaną logikę, która zwiększa ryzyko wystąpienia błędów i utrudnia pełne przetestowanie funkcji.

5.2 Metryki duplikacji kodu

- **Code Duplication Percentage (Procent duplikacji kodu):** Określa, jaki procent kodu stanowią fragmenty powtarzające się w różnych miejscach projektu. Wysoka duplikacja może prowadzić do niespójności, ponieważ zmiany w jednym miejscu wymagają modyfikacji w wielu innych.
- **Duplicated Blocks / Files / Lines:** Liczba zduplikowanych bloków, plików lub linii kodu. Pozwala to na szczegółową analizę, które fragmenty kodu wymagają refaktoryzacji.

5.3 Metryki wielkości i struktury kodu

- **Lines of Code (LOC):** Mierzy rozmiar kodu poprzez liczbę linii zawierających rzeczywisty kod. Choć sama liczba nie definiuje jakości, duże projekty mogą wymagać dodatkowej uwagi przy utrzymaniu.
- **Functions/Methods/Statements:** Liczba funkcji, metod i instrukcji, która pozwala ocenić stopień modularności kodu. Nadmierna liczba parametrów lub zbyt rozbudowane funkcje może wskazywać na problemy w projektowaniu.
- **Number of Parameters:** Liczba parametrów przyjmowanych przez funkcję lub metodę. Wysoka liczba parametrów często sugeruje zbyt skomplikowany interfejs i większe ryzyko błędów.

5.4 Metryki pokrycia testami

- **Test Coverage / Line Coverage / Branch Coverage:** Mierzą, jaki procent kodu jest wykonywany podczas testów jednostkowych. Wysokie pokrycie testami zwykle przekłada się na większą pewność co do poprawności działania kodu oraz ułatwia wykrywanie regresji.

5.5 Metryki Halsteada

- **Unique/Total Operators i Operands:** Te metryki liczą unikalne oraz całkowite wystąpienia operatorów (np. +, -, *) i operandów (np. zmienne, stałe) w kodzie. Na ich podstawie oblicza się kolejne miary, takie jak objętość programu, trudność, wysiłek czy szacunkową liczbę błędów.

5.6 Inne metryki

- **Comment Lines:** Liczba linii zawierających komentarze. Właściwie umieszczone komentarze pomagają w zrozumieniu kodu, natomiast ich nadmiar lub niewłaściwe użycie może wskazywać, że kod nie jest wystarczająco czytelny.
- **Assessment/FTA Score:** Zbiorcza ocena jakości kodu, która na podstawie innych metryk informuje, czy dany fragment kodu jest utrzymywalny, czy wymaga poprawy.

Znaczenie metryk:

- Umożliwiają obiektywną identyfikację potencjalnych problemów w kodzie, takich jak nadmierna złożoność czy duplikacja.
- Pomagają w ocenie czytelności i utrzymywalności kodu, co jest kluczowe przy wprowadzaniu nowych członków zespołu.
- Wspierają planowanie testów poprzez wskazanie obszarów niewystarczająco pokrytych testami.
- Pozwalają na oszacowanie wysiłku niezbędnego do utrzymania i dalszego rozwoju oprogramowania.

6 Analiza danych

Kluczowe wnioski z analizy 47 rekordów z tabeli `metricDiscrepancies`:

1. Różnice w pomiarach:

- Średnia różnica dla metryk złożoności wyniosła
- Maksymalna rozbieżność w wykrywaniu duplikatów osiągnęła
- Dla metryk pokrycia kodu odnotowano różnice do

2. Różnice definicyjne:

- JaCoCo wyklucza gałęzie wyjątków z pokrycia kodu, podczas gdy SonarQube je uwzględnia
- PMD i Checkstyle stosują różne algorytmy obliczania złożoności NPATH
- 5 narzędzi stosuje różne progi dla wykrywania długich metod

3. Stronniczość narzędzi:

- CppDepend konsekwentnie zaniża liczbę linii kodu o 15-20

- Analizatory zintegrowane z IDE (PyCharm) wykazują tendencję do zawyżania metryk jakości kodu
- Narzędzia oparte na analizie statycznej (PVS-Studio) dają bardziej konserwatywne wyniki

7 Tabela obliczeń statycznych

Tabela 1: Obliczenia statyczne

toolMetricID	Liczba	Min	Max	Średnia
2	6.00	16.00	77.00	36.33
24	5.00	80.00	171 890.00	35 858.20
25	2.00	90.00	3208.00	1649.00
26	2.00	37.00	453.00	245.00
27	2.00	2387.00	56 716.00	29 551.50
28	2.00	421.00	12 920.00	6670.50
29	2.00	26 198.00	209 767.00	117 982.50
31	2.00	316.00	3487.00	1901.50
32	2.00	77 789.00	1 512 284.00	795 036.50
33	2.00	42 489.00	1 103 159.00	572 824.00
34	2.00	3570.00	84 368.00	43 969.00
36	2.00	28 933.00	654 752.00	341 842.50
39	2.00	0.00	0.00	0.00
40	2.00	0.00	0.00	0.00
41	2.00	27 230.00	643 287.00	335 258.50
67	1.00	187.00	187.00	187.00
74	1.00	191.00	191.00	191.00
76	1.00	318.00	318.00	318.00
77	1.00	325.00	325.00	325.00
78	6.00	10.00	71.00	28.00
79	6.00	22.00	29.00	24.67
80	6.00	66.00	177.00	119.33
81	6.00	136.00	584.00	313.67
82	6.00	145.00	721.00	406.83
83	6.00	281.00	1305.00	720.50
84	6.00	90.00	203.00	144.00
85	6.00	1824.21	10 003.26	5277.96
86	6.00	25.09	61.11	39.30
87	6.00	48 092.83	529 720.83	244 514.64

toolMetricID	Liczba	Min	Max	Średnia
88	6.00	2671.82	29 428.94	13 584.15
89	6.00	0.61	3.33	1.76
90	6.00	88.00	528.00	273.83
91	6.00	48.59	71.36	59.79
109	6.00	102.00	210.00	144.33
111	6.00	1.00	4.20	2.17
112	6.00	1.00	28.00	8.83
113	6.00	16.40	69.30	42.27
114	6.00	88.00	528.00	273.83
115	6.00	1.00	4.20	2.17
116	6.00	1.00	28.00	8.83
117	6.00	16.40	69.30	42.27
118	6.00	88.00	528.00	266.33
121	1.00	1397.00	1397.00	1397.00
122	1.00	1845.00	1845.00	1845.00
123	1.00	2077.00	2077.00	2077.00
129	6.00	88.00	528.00	273.83
130	6.00	0.00	0.00	0.00
131	6.00	2.00	19.00	9.50
138	4.00	30.00	43.00	36.75
139	4.00	31.00	42.00	38.00
142	7.00	0.00	100.00	51.00
143	7.00	0.00	100.00	51.80
144	11.00	30.00	132.00	55.27
145	11.00	21.00	80.00	53.64
148	12.00	68.00	347.00	163.42
149	11.00	33.00	491.00	189.00
151	12.00	0.00	33.00	11.75
152	12.00	68.00	368.00	163.42
153	11.00	33.00	813.00	258.91
154	11.00	29.00	120.00	51.27
155	5.00	22.00	100.00	50.80
156	7.00	0.00	100.00	48.86
157	1.00	1.00	1.00	1.00
158	1.00	1.00	1.00	1.00
159	1.00	10 863.00	10 863.00	10 863.00
160	1.00	5055.00	5055.00	5055.00
161	1.00	1198.00	1198.00	1198.00
162	1.00	6.00	6.00	6.00
163	1.00	4.00	4.00	4.00

toolMetricID	Liczba	Min	Max	Średnia
164	1.00	226.00	226.00	226.00
165	1.00	838.00	838.00	838.00
166	1.00	1.00	1.00	1.00
167	1.00	10.00	10.00	10.00
168	1.00	113.00	113.00	113.00
169	1.00	45.00	45.00	45.00
170	1.00	23.00	23.00	23.00
171	1.00	194.00	194.00	194.00
172	1.00	601.00	601.00	601.00
173	1.00	5.00	5.00	5.00
174	1.00	18 728.00	18 728.00	18 728.00
175	1.00	5587.00	5587.00	5587.00
176	1.00	1471.00	1471.00	1471.00
177	1.00	3.00	3.00	3.00
178	1.00	85.00	85.00	85.00
179	1.00	4.00	4.00	4.00
180	1.00	7.00	7.00	7.00
181	1.00	13 223.00	13 223.00	13 223.00
182	1.00	10 163.00	10 163.00	10 163.00
183	1.00	90.00	90.00	90.00
184	1.00	105.00	105.00	105.00
185	1.00	499.00	499.00	499.00
186	1.00	3565.00	3565.00	3565.00
187	1.00	1.00	1.00	1.00
188	1.00	1.00	1.00	1.00
189	1.00	65.00	65.00	65.00
190	1.00	1.00	1.00	1.00
191	1.00	5.00	5.00	5.00
192	1.00	10.00	10.00	10.00
193	1.00	5257.00	5257.00	5257.00
194	1.00	7.00	7.00	7.00
195	1.00	591.00	591.00	591.00
196	1.00	1270.00	1270.00	1270.00
197	1.00	1.00	1.00	1.00
198	1.00	3.00	3.00	3.00
199	1.00	2.00	2.00	2.00
200	1.00	242.00	242.00	242.00
201	1.00	40.00	40.00	40.00
202	1.00	33.00	33.00	33.00
203	1.00	254.00	254.00	254.00

toolMetricID	Liczba	Min	Max	Średnia
204	1.00	28.00	28.00	28.00
205	1.00	12.00	12.00	12.00
206	1.00	2.00	2.00	2.00
207	1.00	58.00	58.00	58.00
208	1.00	16.00	16.00	16.00
209	1.00	44.00	44.00	44.00
210	1.00	3.00	3.00	3.00
211	1.00	5.00	5.00	5.00
212	1.00	55.00	55.00	55.00
213	1.00	1.00	1.00	1.00
214	1.00	3577.00	3577.00	3577.00
215	1.00	34.00	34.00	34.00
216	1.00	114.00	114.00	114.00
217	1.00	569.00	569.00	569.00
218	1.00	19 134.00	19 134.00	19 134.00
219	1.00	142 076.00	142 076.00	142 076.00
220	1.00	418.00	418.00	418.00
221	1.00	166.00	166.00	166.00
222	1.00	2.29	2.29	2.29
223	1.00	2.70	2.70	2.70
224	1.00	2.66	2.66	2.66
225	4.00	1333.00	65 464 566.00	16 411 453.75
226	1.00	1333.00	1333.00	1333.00
227	1.00	1333.00	9 999 999.00	5 000 666.00

8 Wnioski

1. Konieczność standaryzacji:

- Należy opracować wspólny słownik metryk uwzględniający różnice definicyjne
- Wymagana jest unifikacja metod obliczeniowych dla kluczowych metryk

2. Wybór narzędzi:

- Dla projektów wielojęzycznych rekomenduje się kombinację SonarQube + Qo-dana
- W projektach Java warto stosować JaCoCo do pokrycia kodu i PMD do analizy jakości

- Dla C++ najbardziej spójne wyniki daje połączenie PVS-Studio i cppdepend

3. Walidacja i dalsze badania:

- Niestandardowe skrypty (np. dla OCLint) wymagają standaryzacji metod pomiaru
- Konieczne jest rozszerzenie badań o analizę false-positive/false-negative
- Warto zbadać wpływ wersji narzędzi na stabilność wyników

9 Dyskusja

Wyniki sugerują, że wybór analizatora statycznego powinien być uzależniony od konkretnych potrzeb projektowych. Istotnym wnioskiem jest także konieczność rozważenia aspektów praktycznych integracji narzędzi z istniejącymi procesami CI/CD, co znacząco wpływa na wydajność zespołu i jakość ostatecznych wyników analizy. Dodatkowo, zidentyfikowane rozbieżności wskazują na potrzebę przeprowadzenia dalszych badań nad przyczynami takich różnic, szczególnie w zakresie wpływu specyficznych ustawień i konfiguracji narzędzi. Przyszłe badania powinny również uwzględniać opinie użytkowników końcowych na temat użyteczności i klarowności raportów generowanych przez analizatory.

Podziękowania

Autorzy pragną podziękować Uniwersytetowi Jagiellońskiemu za wsparcie w realizacji niniejszego projektu.

Oświadczenie o wkładzie autorów

Maryia Babinskaya:

Alicja Bieżykudek:

Mariia Saltykova:

Vladyslav Khabanets:

Tomasz Kosmulski:

Maksim Zdobnikau:

Oświadczenie o konflikcie interesów

Autorzy deklarują brak konfliktu interesów.

Dostępność danych

Dane użyte w badaniu są dostępne na platformie example.com.

Literatura