Opis technik używanych przez studentów z poprzedniego roku

1 Wstęp

Studenci poprzedniego rocznika w analizie jakości kodu źródłowego wykorzystywali szeroki wachlarz narzędzi i technik. Dominowały narzędzia statycznej analizy kodu, które umożliwiały wykrywanie typowych błędów programistycznych, nieużywanych zmiennych oraz naruszeń standardów stylistycznych. Poniżej przedstawiono szczegółowy opis wykorzystywanych narzędzi oraz różnice pomiędzy nimi.

2 Wykorzystane narzędzia i ich charakterystyka

- **Qodana** – wielojęzyczny analizator od JetBrains, wspierający ponad 60 języków. Szczególnie przydatny w pracy zespołowej. Skupia się na jakości i standaryzacji kodu.
- **SonarQube** – rozbudowane narzędzie do analizy statycznej z naciskiem na bezpieczeństwo. Umożliwia wykrycie duplikacji, luk bezpieczeństwa i tzw. code smells. W porównaniu do innych narzędzi oferuje bardziej kompleksowe analizy.
- **PMD** – wykrywa typowe błędy programistyczne (np. nieużywane zmienne, puste bloki). Wyróżnia się podejściem skupionym na jakości praktycznej i wydajności kodu.
- **JaCoCo** – specjalistyczne narzędzie służące do analizy pokrycia kodu testami jednostkowymi.
- **Checkstyle** – wspiera zgodność z ustalonymi konwencjami kodowania. Charakteryzuje się dużą konfigurowalnością.
- **Lizard** – analizator cyklomatycznej złożoności, prosty i szybki, ale mniej szczegółowy niż cppdepend czy SonarQube.
- **cppdepend** – zaawansowane narzędzie dla języków C/C++, oferujące dogłębną analizę jakości kodu.
- **OCLint** – analizator C/C++ i Objective-C, mniej dokładny niż cppdepend, ale przydatny w prostych analizach.

- **PVS-Studio** – skupia się na bezpieczeństwie, wykrywa błędy zgodne ze standardami CERT i MISRA.
- **PyCharm, PyDev** – IDE dla Pythona, wspierające analizę kodu, debugowanie i testowanie, choć mniej wyspecjalizowane niż dedykowane analizatory.

3 Techniki analityczne

Do szczegółowych analiz metryk takich jak:

- cyklomatyczna złożoność kodu,
- liczba parametrów funkcji,
- wykrywanie zduplikowanych fragmentów kodu,

– wykorzystywano narzędzia takie jak **Lizard, OCLint, cppdepend** oraz **FTA**. Studenci tworzyli również własne skrypty (np. w połączeniu z OCLint), umożliwiające dostosowanie wyników do specyfiki projektu.

4 Porównanie i różnice między analizatorami

Podczas pracy zauważono znaczne rozbieżności w wynikach analiz. Wynikały one z:

- różnych sposobów zliczania linii kodu i komentarzy,
- odmiennych definicji metryk (np. liczby parametrów, złożoności),
- różnych poziomów dokładności i zaokrągleń,
- pominięcia niektórych struktur przez wybrane analizatory.

Przykład: cppdepend wykazywał niższą liczbę linii kodu niż OCLint, który uwzględniał także linie zawierające komentarze. Lizard natomiast zaokrąglał wyniki cyklomatycznej złożoności, co powodowało pozornie nieistotne, ale istotne w analizie różnice.

5 Wnioski

Techniki statycznej analizy kodu okazały się niezwykle skuteczne, lecz wymagały świadomego doboru narzędzi oraz konfiguracji. Studenci musieli wykazać się umiejętnością krytycznej analizy wyników, szczególnie w kontekście porównań między narzędziami. Zastosowanie wielu narzędzi pozwalało na kompleksową ocenę jakości kodu, jednak interpretacja danych często wymagała ręcznej weryfikacji i harmonizacji.