Otwarte repozytoria kodu i pomiar oprogramowania laboratorium

adam.roman@uj.edu.pl
jaroslaw.hryszko@uj.edu.pl

Instytut Informatyki i Matematyki Komputerowej UJ

Semestr letni 2024/2025

SkanUJkod

Wymagania Systemowe i Wizja Rozwoju

1 Wprowadzenie

Niniejszy dokument prezentuje założenia oraz wymagania systemowe dla projektu **SkanUJkod** – narzędzia przeznaczonego do skanowania kodu źródłowego w celu zbierania licznych metryk. Rozwiązanie to ma na celu dostarczenie kompleksowego zestawu statystyk dotyczących jakości, bezpieczeństwa i struktury kodu, a także wspomaganie oceny ryzyka i identyfikacji potencjalnych problemów.

2 Wizja systemu

- **Open-source:** Projekt SkanUJkod jest tworzony jako oprogramowanie otwarte, umożliwiające współpracę społeczności w zakresie rozwoju i konserwacji.
- **Modułowa architektura:** Proponuje się zaimplementowanie architektury zgodnej z zasadą *Open-Closed Principle*, aby łatwo można było dodawać wsparcie dla kolejnych języków programowania lub nowych metryk bez modyfikowania już istniejących komponentów.
- **Elastyczność rozwoju:** Początkowa implementacja może składać się z prostego interfejsu *CLI*, który w razie potrzeby będzie można rozszerzyć o *GUI* (np. front-end webowy).

3 Kluczowe funkcjonalności

3.1 Metryki statyczne kodu

- Zestaw metryk inspirowanych m.in. narzędziami *SonarQube* oraz podobnymi skanerami.
- Możliwość integracji parserów (np. AntLR) w celu analizy drzewa składniowego (AST) i wyliczania właściwości kodu (m.in. złożoność cyklomatyczna, liczba linii, liczba klas/metod, zagnieżdżenie bloków).
- Niezależność od konkretnych bibliotek (lub ograniczenie do minimum) – preferowana jest implementacja własnych algorytmów lub komponentów open-source w taki sposób, by uniknąć nadmiernego powiązania z jednym producentem.

3.2 Metryki projektowe (dane z Gita)

- Analiza historii repozytorium *Git* (logi, liczba commitów, autorzy, itp.).
- Uwzględnienie wskaźników *Kamei* stosowanych w *just-in-time defect prediction* (np. częstość zmian, liczba deweloperów w pliku, rozmiar commitów) – szczegółowy opis można znaleźć w pracy *Kamei* i wsp.[4].
- Generowanie raportów, pozwalających na porównanie aktywności i trendów w czasie.

3.3 Wykrywanie *code smells* i luk bezpieczeństwa

- Zestaw reguł i heurystyk pozwalających identyfikować potencjalnie nieefektywne lub niebezpieczne wzorce w kodzie (np. *god class*, *long method*, możliwe miejsca podatne na *SQL injection* itp.).
- Raportowanie priorytetów (wysoka / średnia / niska) w zależności od wpływu na bezpieczeństwo oraz utrzymanie kodu.

3.4 Metryki pokrycia (instrumentacja kodu)

- Obsługa pokrycia:
  - *Statement coverage*
  - *Branch coverage*
  - Bardziej zaawansowane miary, takie jak *condition coverage*, *MC/DC coverage*, *loop coverage*, *prime path coverage*.
- Mechanizm instrumentacji kodu na poziomie plików źródłowych lub bajtkodu (np. w Javie), aby umożliwić śledzenie przepływu wykonywania testów.
- Zaleca się przeprowadzenie dodatkowego *researchu* w zakresie narzędzi i metod instrumentacji, aby dobrać najbardziej efektywne rozwiązanie.

4 Założenia projektowe

4.1 Architektura programu

- **Open-Closed Principle:** Każdy nowy element (np. język programowania) powinien dać się dodać w postaci modułu, bez konieczności ingerencji w rdzeń aplikacji.
- **Możliwość rozszerzania:** Rozszerzenia mogą obejmować nowe rodzaje metryk, obsługę dodatkowych repozytoriów kontroli wersji (np. *Mercurial*, *SVN*), czy alternatywne sposoby instrumentacji.
- **Podział na warstwy:** Rekomenduje się rozdzielenie warstwy analizy od warstwy prezentacji (CLI/GUI).

4.2 Research i dokumentacja metryk

- **Źródła do analizy:**
  - *Vector: PC-lint Plus Metrics* [1],
  - *Ator's GBS metrics* [2],
  - *PMD Java Metrics Index* [3],
  - Metryki zmian z pracy *Kamei* [4].
- **Definicja operacyjna każdej metryki:**
  - Jasne określenie sposobu jej liczenia i koniecznych danych wejściowych.
  - Wskazanie, w jaki sposób wynik należy interpretować w kontekście jakości i utrzymania kodu.
  - Przykłady praktyczne, jeśli to możliwe (np. fragmenty kodu, minimalne projekty testowe).
- Dokumentacja powinna być spójna, aktualna i łatwo dostępna (np. w repozytorium *GitHub*, w formie `README` lub LaTeX / *MkDocs*).

4.3 Instrumentacja kodu

- Należy opracować lub włączyć istniejące narzędzia do instrumentacji kodu, aby liczyć pokrycie testowe.
- Rekomenduje się rozpoczęcie od najprostszych miar (*function coverage*, *statement coverage*) i stopniowo dodawać kolejne (*branch*, *condition*, *MC/DC*, itp.).
- Możliwe jest dokonanie instrumentacji na poziomie bajtkodu (w przypadku języków takich jak Java), co może zwiększyć elastyczność i wydajność procesu.

5 Możliwe kierunki rozwoju

- **Rozszerzony *GUI***: opracowanie intuicyjnego interfejsu graficznego prezentującego statystyki oraz wykresy (opcjonalnie aplikacja webowa).
- **Integracja z narzędziami CI/CD**: automatyczne uruchamianie skanów w ramach potoku (pipeline) w narzędziach typu *Jenkins*, *GitLab CI*, *GitHub Actions*.

- **Analiza wielojęzyczna**: moduły przeznaczone dla różnych języków programowania (Java, Python, JavaScript, C/C++, itp.).
- **Dalsze metryki projektowe**: rozszerzenie analizy Gita o bardziej zaawansowane modele predykcji defektów, analizy trendów, cross-project learning.

6 Podsumowanie

SkanUJkod ma być uniwersalnym narzędziem do analizy kodu, pozwalającym na szybkie zdiagnozowanie potencjalnych problemów związanych z jakością, bezpieczeństwem oraz utrzymaniem aplikacji. Elastyczna architektura i staranna dokumentacja metryk pozwolą na długofalowe utrzymanie i rozwój projektu. Początkowa wersja *CLI* może z czasem ewoluować w bardziej rozbudowany ekosystem z graficznym interfejsem użytkownika, integracją z systemami CI/CD oraz dodatkowymi modułami odpowiadającymi na potrzeby współczesnego środowiska programistycznego.

Literatura

[1] Vector. *PC-lint Plus Metrics Documentation*. `https://help.vector.com/pclintplus/Metrics.html` (dostęp: 27.02.2025).

[2] Ator's GBS. *Metrics Documentation*. `https://ator1699.home.xs4all.nl/Work/GBS/Doc_and_download/doc/metrics.html` (dostęp: 27.02.2025).

[3] PMD. *PMD Java Metrics Index*. `https://docs.pmd-code.org/pmd-doc-6.55.0/pmd_java_metrics_index.html` (dostęp: 27.02.2025).

[4] Yasutaka Kamei *et al.*, *Studying Just-In-Time Defect Prediction using Cross-Project Models*, EMSE, 2016. `https://rebels.cs.uwaterloo.ca/papers/emse2016_kamei.pdf` (dostęp: 27.02.2025).