BAZA DANYCH BŁĘDÓW W OTWARTOŹRÓDŁOWYCH PROJEKTACH C++

Dokumentacja projektowa PZSP2

WERSJA 0.1

Semestr 5

Zespół nr 12 w składzie:
Łukasz Szydlik
Dominik Śledziewski
Jan Szwagierczak
Tomasz Okoń

Mentor zespołu: Krzysztof Cabaj
Właściciel tematu: Robert Nowak

Spis treści

1 Wprowadzenie

1.1 Cel projektu

Celem projektu jest stworzenie **bazy danych typu Ground Truth** dla zadań uczenia maszynowego, wraz z systemem ulatwiającym jej stworzenie. Baza będzie zawierać:

- fragmenty kodu źródłowego w języku C++ zawierające błędy oraz ich poprawione wersje,
- etykiety opisujące typ błędu (np. *memory-leak*, *null-pointer*, *formatting*),
- możliwość wykorzystania danych do trenowania modeli uczenia maszynowego (np. transformerów typu *CodeBERT*).

Baza ma stanowić zbiór danych uczących i testowych dla modeli klasyfikujących lub oceniających jakość kodu C++.

1.2 Wstępna wizja projektu

Projekt zakłada implementację kompletnego narzędzia analityczno–danych obejmującego:

- analizę otwartych repozytoriów projektów C++ (np. Firefox, Blender, XGBoost),
- automatyczne wykrywanie commitów związanych z poprawą błędów,
- ekstrakcję fragmentów kodu przed i po poprawce,
- statyczną analizę kodu i automatyczne przypisanie etykiet błędów,
- zapisanie rekordów w bazie NoSQL (MongoDB),
- narzędzia do analizy statystycznej i wykorzystania danych w modelach ML (CodeBERT, PyTorch).

System będzie elastyczny i rozszerzalny — umożliwi dodawanie nowych repozytoriów oraz etykiet błędów.

2 Metodologia wytwarzania

Projekt realizowany jest w oparciu o **iteracyjno-przyrostowy model wytwarzania oprogramowania**, obejmujący etapy: analizy wymagań, projektowania, implementacji, testowania oraz walidacji. Każdy etap kończy się weryfikacją zgodności z wymaganiami i przeglądem postępów z mentorem. Wymagania są rozwijane w miarę postępu prac, a ich realizacja dokumentowana i kontrolowana. Kluczowym elementem jest ciągła analiza jakości kodu oraz poprawność generowanych danych w bazie MongoDB. Projekt prowadzony jest w środowisku Linux z wykorzystaniem otwartego oprogramowania (PyDriller, cppcheck, MongoDB, PyTorch).

Podział zadań w zespole:

- **Analiza repozytoriów i pozyskiwanie commitów:** odpowiedzialny za implementację modułów pobierających dane z repozytoriów Git,

- **Analiza błędów i etykietowanie kodu:** odpowiedzialny za integrację narzędzi cppcheck, clang-tidy oraz systemu klasyfikacji błędów,
- **Projekt i obsługa bazy danych MongoDB:** projekt struktury rekordów, implementacja skryptów zapisu i odczytu danych,
- **Integracja z modelem uczenia maszynowego:** przygotowanie danych do treningu oraz test z wykorzystaniem modelu CodeBERT,
- **Dokumentacja i raportowanie:** opracowanie dokumentacji technicznej i raportu końcowego projektu.

3 Analiza wymagań

3.1 Wymagania użytkownika i biznesowe

Wymagania biznesowe

1. Ułatwienie badaczom i studentom tworzenia oraz testowania modeli AI analizujących kod źródłowy C++.
2. Stworzenie zestawu danych analogicznego do *Defects4J*, lecz dla języka C++.
3. Umożliwienie automatycznej klasyfikacji błędów kodu w dużych projektach open source.
4. Udostępnienie ustrukturyzowanej bazy do dalszych badań naukowych.

Wymagania użytkownika

1. Użytkownik może zaimportować kod C++ z repozytoriów open source.
2. Użytkownik może przeszukiwać i filtrować rekordy błędów wg projektu lub etykiety.
3. Użytkownik może uruchomić model ML w celu klasyfikacji błędów.
4. Użytkownik może analizować statystyki etykiet błędu (np. liczba błędów typu *memory-leak* w danym repozytorium).
5. Użytkownik ma do dyspozycji rekordy z minimum 3 etykietami.

Wymagania systemowe

1. System automatycznie wykrywa commit-y naprawiające błędy.
2. System pobiera fragment kodu przed i po poprawce.
3. System analizuje kod za pomocą narzędzi cppcheck, clang-tidy.
4. System umożliwia manualne rozpatrzenie niearbitralnych błędów.
5. System generuje etykiety błędów i zapisuje je w bazie danych.
6. System zapewnia interfejs CLI umożliwiające pracę z danymi.
7. System pozwala na eksport danych do formatów JSON/CSV.

3.2 Wymagania funkcjonalne i niefunkcjonalne

Patrz Tabela 1 oraz Tabela 2.

Wymagania funkcjonalne

	Opis funkcji	Priorytet
1	Pobranie repozytorium Git	MUST
2	Analiza commitów zawierających frazy „fix”, „bug”, „leak”	MUST
3	Generowanie rekordów zawierających code_buggy i code_fixed, oraz etykietę błędu	MUST
4	Analiza kodu przez cppcheck/clang-tidy	SHOULD
5	Półautomatyczne przypisanie etykiet błędów	MUST
6	Automatyczne wstawienie rekordów par code_buggy i code_fixed wraz z przydzieloną etykietą	MUST
7	Analiza statystyczna i raport błędów	SHOULD
8	Wygenerowane rekordy będą umożliwić trenowanie modeli ML	MUST
9	Minimalna ilość etykiet: 3	MUST
10	Pary code_buggy i code_fixed zawierają kontekst na poziomie klas (pliki .cpp /.cc i .h)	SHOULD
11	Baza danych będzie zawierać minimum 20 projektów open source	MUST
12	Dodawanie nowych etykiet	COULD
13	Aplikacja konsolowa umożliwiająca tworzenie bazy oraz zarządzanie nią	MUST
14	Eksportowanie rekordów do formatów JSON/CSV	COULD
15	Ręczne dodawanie rekordów	MUST

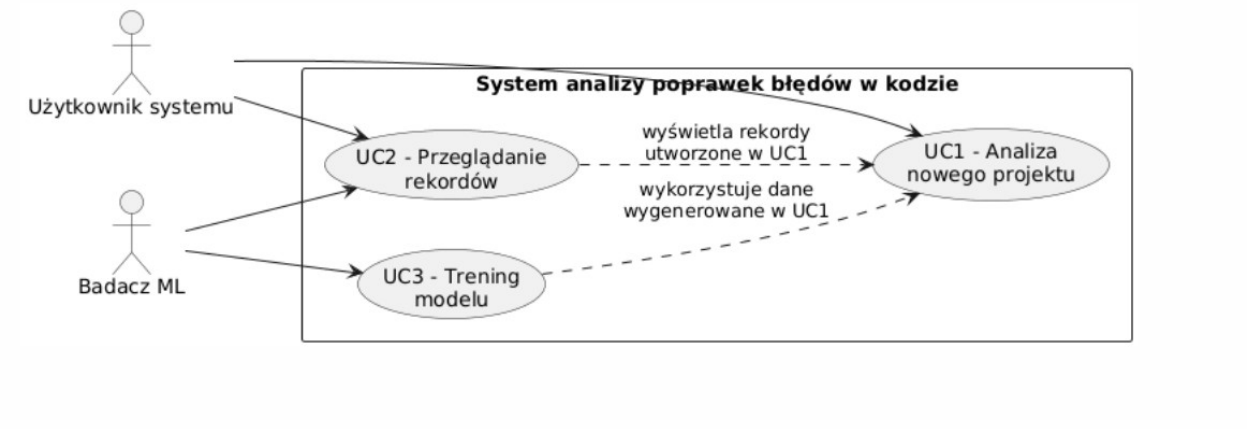
Tabela 1. Wymagania funkcjonalne

Wymagania niefunkcjonalne

	Opis wymagania	Typ	Kryterium
1	System działa w środowisku Linux	środowiskowe	test systemowy
2	Wszystkie pliki kodu UTF-8, bez tabulacji	jakościowe	walidacja skryptem
3	Czas analizy jednego projektu ≤ 5 min (dla 100 commitów)	wydajność	test czasowy
4	Baza danych powinna obsłużyć ≥ 1000 rekordów	skalowalność	test obciążeniowy
5	System umożliwia rozszerzenie o nowe etykiety	elastyczność	test integracyjny
6	Kod źródłowy narzędzia open source	prawne	licencja MIT/GPL
7	Dla próbki kodu, predykcja zostanie wykonana w mniej niż 100 ms	wydajność	Test czasowy

Tabela 2. Wymagania niefunkcjonalne

3.3 Przypadki użycia



Rysunek 1. Zależnosci między przypadkami użycia

UC1 – Analiza nowego projektu

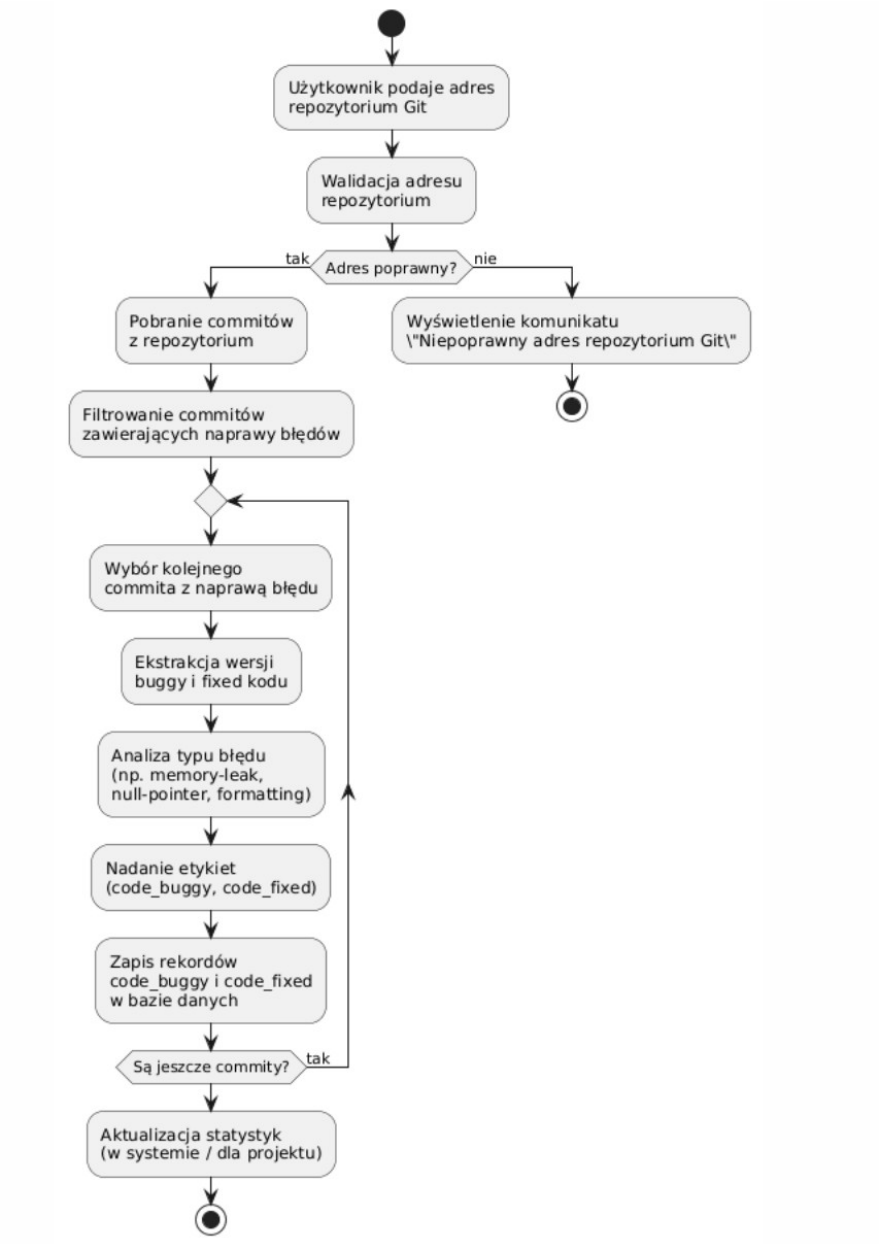
Patrz: Rysunek 2

Aktor: użytkownik systemu

Opis: użytkownik podaje adres repozytorium Git, system pobiera i analizuje commit-y zawierające naprawy błędów.

Rezultat: rekordy code_buggy i code_fixed w bazie.

Etykiety: memory-leak, null-pointer, formatting.



Rysunek 2. UC1

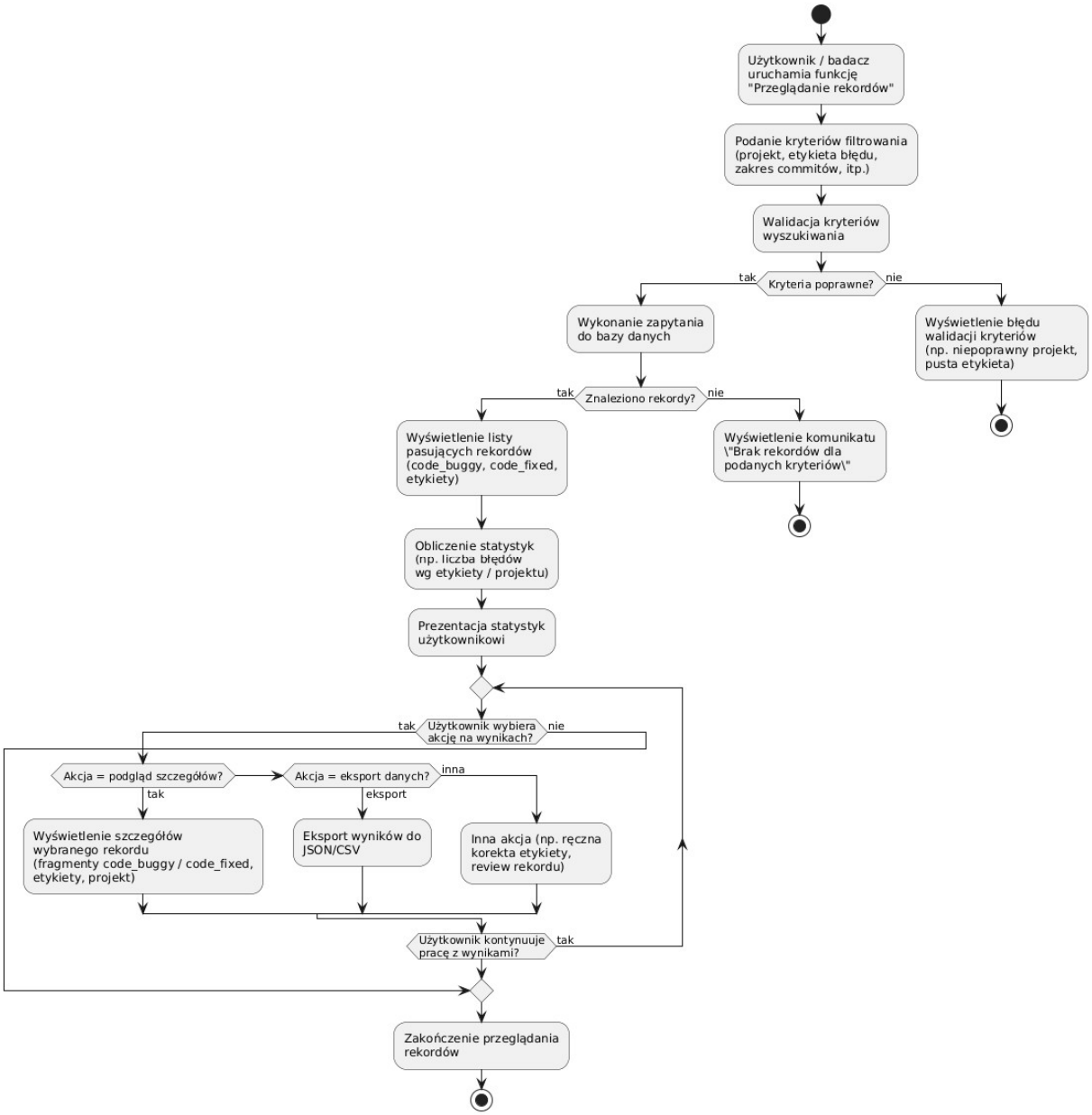
UC2 – Przeglądanie rekordów

Patrz: Rysunek 3

Aktor: użytkownik / badacz

Opis: użytkownik filtruje rekordy po typie błędu lub projekcie.

Rezultat: wyświetlenie listy wyników i statystyk.



Rysunek 3. UC2

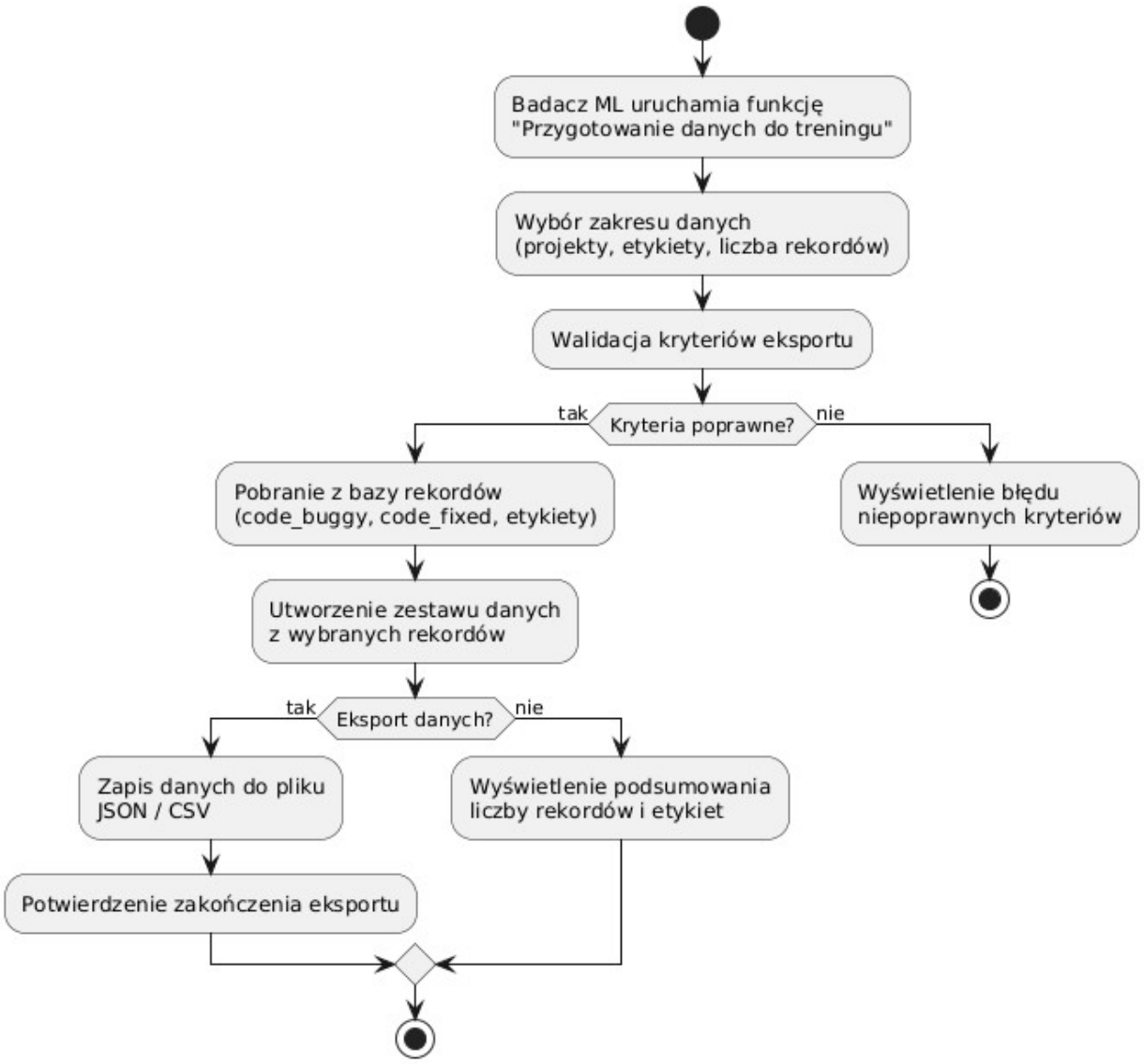
UC3 – Przygotowanie danych do treningu

Patrz: Rysunek 4

Aktor: badacz ML

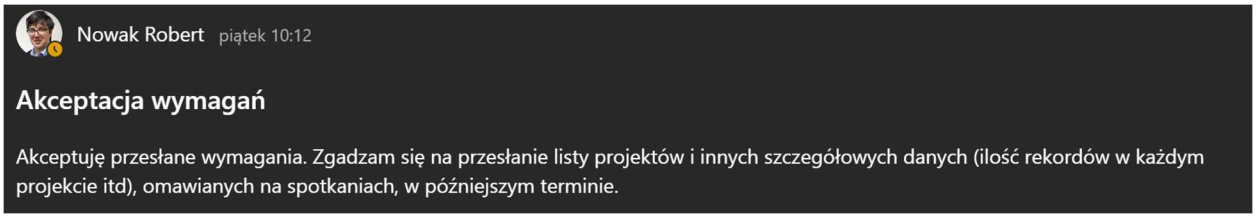
Opis: system eksportuje zadany zakres danych.

Rezultat: przygotowany zbiór danych pod uczenie modelu maszynowego.



Rysunek 4. UC3

3.4 Potwierdzenie zgodności wymagań



Rysunek 5. Akceptacja wymagań