Система підтримки прийняття рішень для верифікації якості коду прикладного програмного забезпечення

https://doi.org/10.31713/MCIT.2023.059

Владислав Паращенко
Кафедра комп'ютерних наук
Сумський державний університет
м. Суми, Україна

Олег Берест
Кафедра комп'ютерних наук
Сумський державний університету
м. Суми, Україна

Анотація—**запропонована архітектура системи підтримки прийняття рішень для контролю якості програмного коду прикладних кодових баз. Запропонований підхід є модульним та придатним до застосування в більшості проектів по розробці програмного забезпечення. Запропонована система дозволяє вивести емпіричні правила для довгострокової розробки та підтримки програмного забезпечення.**

Ключові слова—якість програмного коду; система підтримки прийняття рішень; метрики коду.

I. Вступ

Складність підтримки програмного забезпечення полягає в неможливості охоплення значних об'ємів даних в полі уваги однієї людини. Але електронні обчислювальні машини не мають подібної проблеми, величезні об'єми оперативної пам'яті дозволяють їм зберігати та оперувати станом надзвичайно складних систем. Недоліком комп'ютерів є те що вони можуть працювати лише за наперед заданим алгоритмом, який надзвичайно складно реалізувати для системи, яка має необмежену кількість станів. В просторі вирішення подібних проблем з'явилося машинне навчання, яке покликане вивести правила прийняття рішення з попереднього досвіду, в якому вже містяться необхідні дані. Цей підхід полегшує реалізацію, проте програє в точності перед системами, які мають задану систему правил.

Критерій підтримуваності є одним з характеристик якості програмного забезпечення згідно з ISO25010:2011[1]. Довгострокова розробка вимагає значних людських та часових ресурсів. Часто при передачі відповідальності від одного розробника до іншого втрачаються знання, які необхідні в процесі розробки. Тож ці знання треба зберігати у вигляді документації, яку необхідно вчасно оновлювати. Недоліком документації є її швидка втрата актуальності та невідповідність реалізації. Найбільш якісним методом буде автоматизована генерація документації з програмного коду, але й такий підхід має недоліки в зв'язку з обмеженістю та недосконалістю наявного інструментарію. З цього випливає, що побудова системи, яка мала б всі дані про кодову базу та могла б виводити правила та критерії підтримки є необхідністю сьогодення.

II. Основна частина

Рішення, яке могло б верифікувати показники підтримуваності все ще не існує. Якість розроблюваних систем майже повністю залежить від розробників. Для автоматизації знаходження розповсюджених проблем та вразливостей застосовують статичний аналіз програмного коду. Такі інструменти як SonarCube, PVS Studio чи DeepSource найбільш популярні в цій галузі. Програмні комплекси для статичного аналізу мають фіксований набір правил, який дозволяє запобігти поширеним помилкам. Також такі утиліти як правило звичайно мають інтерфейс розширення для додавання власних правил. Проте вагомою проблемою таких інструментів є обмеженість критеріїв, які можна застосувати. Кожне правило повинно бути розроблене та додане до проекту, подібні системи не виявляють емпіричні закономірності побудови кодових баз, що було б надзвичайно корисно в задачах довгострокової підтримки.

Рішенням зазначених проблем може стати система підтримки прийняття рішень для контролю якості програмного забезпечення, що має мати достатньо зібраної інформації для її нормального функціонування. Найбільш якісним джерелом буде сам програмний код, який на відміну від будь яких схем, діаграм чи документації, завжди буде актуальним. Як критерії якості можуть бути застосовані загально відомі метрики оцінювання програмного коду LCOM[2], CBO [3], NOC[4], WMC[5]. Вони дозволяють отримати числове представлення коду, яке визначає відповідність реалізації принципам SOLID[6]. Більшість популярних мов програмування, наприклад Java чи Python, містять готові рішення, які дозволяють визначити значення подібних метрик. Іншим підходом може бути створення власної реалізації, використовуючи абстрактне синтаксичне дерево(AST) в якості джерела даних. Наступним кроком буде навчання системи підтримки прийняття рішень. В якості моделі може бути використане класичне дерево рішень.

Альтернативні моделі нейромереж чи більш складних агрегаційних моделей по типу лісу випадкових дерев не дозволяють однозначно визначити та візуалізувати шлях прийняття рішення,

що є ефективним для трактування отриманих порад при застосуванні системи. Обчислювальна складність алгоритму дозволяє навчати СППР її з використанням лише центрального процесора, без застосування графічних прискорювачів, що значно полегшує вимоги до апаратного забезпечення кінцевих користувачів. Можлива функціональна схема системи зображена на Рисунок 1.

Рисунок 1. Функціональна схема системи

Починати збір даних для навчання необхідно зі стартом розробки, щоб використовувати кожен новий програмний внесок (коміт) в якості елемента навчальної вибірки. Завдяки такому підходу не треба витрачати час розробників для ручної розмітки даних. Можна почати використовувати і заздалегідь навчену модель на попередньо відібраних репозиторіях відкритого програмного забезпечення. Навчена модель може відразу почати використовуватися в існуючих проектах, проте вона не буде знати про особливості конкретних реалізацій, тож буде програвати в якості перед моделлю, яка донавчається в процесі розробки.

III. Висновки

Запропонована архітектура дозволяє побудувати систему підтримки прийняття рішень верифікації якості програмного забезпечення, яку можна експлуатувати в ході дострокової підтримки. Така система може виявляти правила та закономірності побудови, що не будуть задокументовані та зрозумілі розробникам, які не мають достатнього досвіду з конкретним проектом. Завдяки модульності рішення таку систему можна адаптувати для майже будь-якої мови програмування. Обчислювальні вимоги запропонованого методу досить незначні, що дозволяє інтегрувати запропоновану систему до будь-якого процесу розробки програмного забезпечення.