

УДК 004.8

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ, ИМИТИРУЮЩЕГО РАБОТУ РАЗРАБОТЧИКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

С. Н. ШЕМЯКОВА, В. А. ПЕТУШКОВ

Белорусско-Российский университет

Могилев, Беларусь

Усложнение архитектуры программных систем актуализирует потребность в специалистах для решения задач промышленной разработки, однако сохраняется существенный разрыв между академической подготовкой и требованиями рынка труда. Существующие образовательные платформы (LeetCode, HackerRank) фокусируются преимущественно на алгоритмической подготовке в идеализированных условиях. Подобный подход не позволяет полноценно моделировать реальный производственный цикл, где инженер сталкивается с неформализованными требованиями, глубокой отладкой стороннего кода, жесткими ресурсными ограничениями и соблюдением стандартов (SOLID, DRY, Clean Code). Следовательно, традиционные методы обладают низкой валидностью и не формируют устойчивых инженерных компетенций.

Для преодоления данного разрыва разработано веб-приложение, имитирующее работу разработчика ПО и смещающее акцент на системный анализ, профилирование ресурсов и экспертное ревью с применением ИИ. Ключевой особенностью системы является механизм динамической генерации технических заданий (ТЗ) с намеренно заложенными логическими противоречиями, что требует преанализа требований и формирования отчетов об ошибках (Bug Reports). Дополнительно реализованы режим целенаправленной отладки чужого кода и интеграция с большими языковыми моделями (LLM), выполняющими функции автоматизированного архитектурного код-ревью.

Для объективной и комплексной оценки инженерных компетенций в разработанном веб-приложении применена многокритериальная математическая модель грейдинга. Итоговый балл за сессию S вычисляется как взвешенная сумма коэффициентов, отражающих различные аспекты качества работы, с применением системы штрафов и бонусов:

$$S = \omega_b \cdot b + \omega_t \cdot t + \omega_r \cdot r + \omega_q \cdot q - P,$$

где b – коэффициент успешности анализа требований (валидация Bug Report, $b \in \{0; 1\}$); t – коэффициент функциональной корректности (доля пройденных unit-тестов в изолированной среде); r – коэффициент ресурсной эффективности (на основе профилирования CPU/RAM); q – коэффициент архитектурного качества (результат семантического анализа кода нейросетью); ω_i – весовые коэффициенты значимости критериев (сумма весов равна 100 баллов); P – кумулятивный штраф за использование подсказок и неэффективные попытки.

Функциональные требования формализованы диаграммой вариантов использования (рис. 1) с четырьмя акторами: Гость, Пользователь, Система и

Внешний LLM-сервис. Сценарии охватывают полный цикл: от выбора траектории и анализа ТЗ до отправки баг-репортов, отладки в веб-IDE и прохождения LLM-ревью.

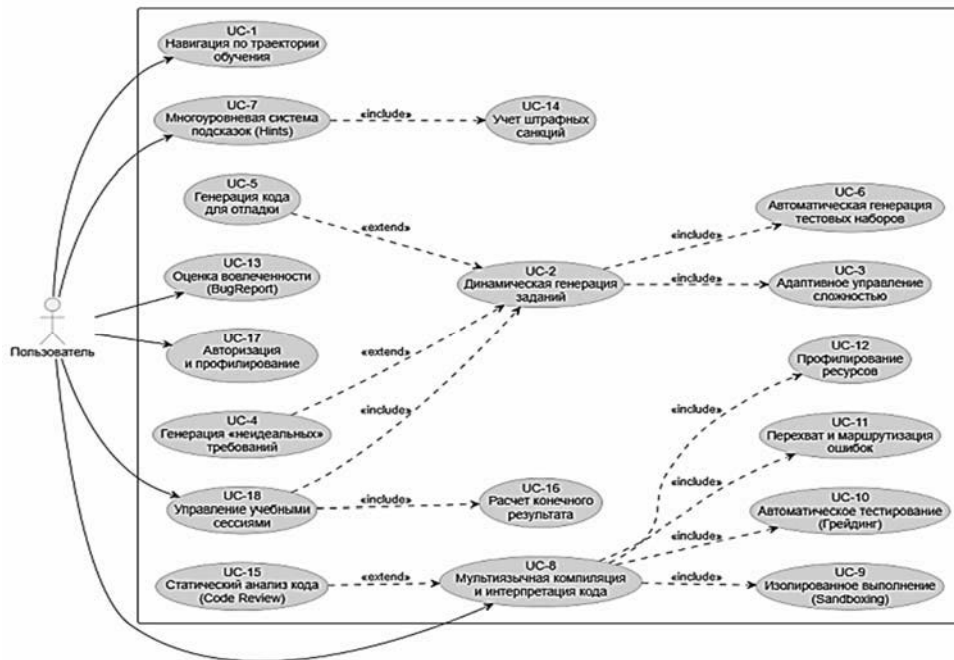


Рис. 1. Диаграмма вариантов использования для пользователя

Архитектура разработанного приложения базируется на клиент-серверном подходе с разделением ответственности. Клиентская часть на базе React 19 и Monaco Editor формирует полноценную среду веб-IDE. Серверный уровень (Node.js, Express 5) предоставляет REST API для оркестрации сессий и агрегации метрик. Безопасное выполнение кода с жестким профилированием ресурсов реализовано в изолированной песочнице (контекст vm для JavaScript, .NET SDK для C#). Интеллектуальные функции (генерация заданий с дефектами, валидация баг-репортов, рефакторинг) выполняет LLM-агент через OpenRouter API. Надежное долговременное хранение и строгий контроль одиннадцати связанных сущностей обеспечивает связка PostgreSQL и Prisma ORM.

Типовой сценарий начинается с инициации спринта, в ходе которого нейросеть генерирует ТЗ с логическим дефектом. После преанализа требований и написания кода решение направляется в песочницу для профилирования и прогона тестов. При успешном прохождении код передается на архитектурное ревью. По итогам спринта система агрегирует метрики по математической модели, формируя сводку с рекомендациями.

В результате работы реализовано веб-приложение, имитирующее работу разработчика ПО и закрывающее разрыв между академической подготовкой и требованиями индустрии. Корректность и надежность бизнес-логики подтверждены результатами тестирования по методологии TDD (Jest, Supertest) и ручной проверкой интерфейса.