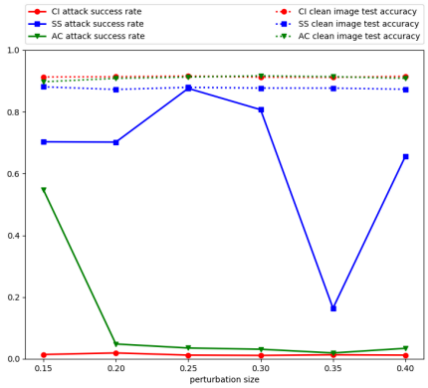

Рисунок 1 - Коефіцієнт успішності та точність атаки на чистому тестовому наборі перенавчених нейронних мереж для трьох захистів

Література.

1. Міллер Д. Дж., Сян Чж., Кесідіс Г. Adversarial Learning in Statistical Classification: A Comprehensive Review of Defenses Against Attacks // Proceedings of the IEEE. 2020. URL: https://arxiv.org/abs/1904.06292.

УДК 004.41

Д.В. Коваль, Г.Б. Цуприк, к.т.н., доц..
Тернопільський національний технічний університет імені Івана Пулюя, Україна

МОДУЛЬНА РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ФІНАНСОВОГО КОНТРОЛЮ НА ANDROID ІЗ ЗАСТОСУВАННЯМ ПРИНЦИПІВ ІНВЕРСІЇ УПРАВЛІННЯ ТА ШАБЛОНУ MVVM

D.V. Koval, H.B. Tsupryk, Ph.D., Assoc. Prof.

MODULAR IMPLEMENTATION OF FINANCIAL CONTROL FUNCTIONALITY ON ANDROID USING INVERSION OF CONTROL PRINCIPLES AND THE MVVM PATTERN

Мобільні застосунки для фінансового контролю набули значної популярності серед користувачів, оскільки вони допомагають ефективно управляти особистими фінансами. Такі програми дозволяють стежити за доходами та витратами, створювати бюджети, аналізувати фінансові звіти та досягати фінансових цілей. Однак для забезпечення надійної та ефективної роботи таких застосунків важливо правильно організувати їх внутрішню структуру, щоб забезпечити гнучкість, масштабованість і легкість у підтримці.

Одним з основних підходів до розробки мобільних застосунків є використання модульної архітектури, що передбачає поділ програми на окремі компоненти. Кожен з цих компонентів має свою чітко визначену функцію і може бути розроблений, тестований і оновлений незалежно від інших частин програми. Наприклад, в фінансовому застосунку один модуль може відповідати за облік доходів і витрат, інший — за категоризацію витрат, а ще один — за аналіз бюджету чи формування фінансових цілей.

Такий підхід до структуризації дає можливість працювати з кожною частиною програми окремо, що значно спрощує роботу розробників. Модульність також дає можливість розподіляти завдання між кількома розробниками, зменшуючи час розробки і підвищуючи ефективність роботи над проєктом. Крім того, модульність дозволяє простіше додавати нові функції в застосунок без потреби переписувати вже готовий код, що робить програму більш гнучкою і стійкою до змін.

Шаблон MVVM (Model-View-ViewModel) є одним із найбільш поширених підходів до побудови архітектури мобільних застосунків, оскільки він дозволяє чітко розділити відповідальність між різними частинами програми. У цьому підході є три основні компоненти: Model, View і ViewModel.

Model відповідає за обробку даних. У випадку фінансового застосунку це можуть бути операції з транзакціями, категоріями витрат, обчислення залишку на рахунку тощо. Модель не має відношення до відображення інформації на екрані або взаємодії з користувачем, вона лише обробляє і надає необхідні дані для інших частин програми.

View — це частина програми, яка безпосередньо взаємодіє з користувачем. Вона відповідає за відображення інтерфейсу та приймання введених даних. Важливо, що View не повинна містити складної логіки — вона лише відображає дані, що отримуються з ViewModel. Така структура дозволяє значно спростити тестування інтерфейсу, оскільки View не містить складних залежностей від інших частин програми.

ViewModel є проміжним ланцюгом між Model і View. Вона отримує дані з Model, готує їх для відображення в View та забезпечує зв’язок між цими компонентами. ViewModel дозволяє відокремити бізнес-логіку від інтерфейсу, завдяки чому програма стає більш зрозумілою та зручною для розробників. Замість того, щоб безпосередньо маніпулювати даними, ViewModel відповідає за організацію логіки, яка визначає, які дані і коли повинні бути показані користувачу. Крім того, завдяки використанню механізмів спостереження за даними, таких як LiveData, зміни в даних автоматично призводять до оновлення інтерфейсу, що забезпечує зручну та динамічну взаємодію з користувачем.

Ще одним важливим аспектом є принцип інверсії управління. Це означає, що компоненти застосунку не створюють залежностей самостійно, а отримують їх ззовні. Такий підхід дозволяє значно знизити зв’язаність між компонентами і полегшує тестування та розширення програми. Замість того, щоб кожен модуль самостійно створював потрібні йому об’єкти, ці об’єкти передаються ззовні, що дає змогу централізовано керувати залежностями. В результаті, програмний код стає більш гнучким і зручним для подальших змін.

Завдяки використанню принципу інверсії управління, кожен компонент програми можна тестувати окремо, замінюючи реальні об’єкти на підроблені для цілей тестування. Це дозволяє легко виявляти помилки в кожному модулі без необхідності тестувати всю програму одночасно.

Застосування таких підходів, як MVVM і інверсія управління, в поєднанні з модульною архітектурою, дає значні переваги при розробці мобільних застосунків. Кожен компонент є незалежним, що полегшує їхнє тестування та оновлення. Крім того, цей підхід дозволяє значно спростити масштабування програми. Якщо з’являються нові вимоги або функції, їх можна легко додати до програми, не порушуючи існуючої структури. Також цей підхід дозволяє значно знизити ймовірність виникнення помилок, оскільки кожен компонент має свою чітко визначену роль і не залежить від інших частин програми.

У підсумку, модульна архітектура, шаблон MVVM та інверсія управління є потужними інструментами для створення стабільних, гнучких і зручних для користувача мобільних застосунків. Вони дозволяють розробникам створювати програмне забезпечення, яке легко підтримується, розширюється і адаптується до нових вимог користувачів. Ці підходи сприяють підвищенню якості коду, полегшують тестування та забезпечують надійність і стабільність застосунку в довгостроковій перспективі.

Література

1. Guide to app architecture | App architecture | Android Developers. Android Developers. URL: https://developer.android.com/topic/architecture (дата звернення: 05.05.2025).

2. Android Programming: The Big Nerd Ranch Guide / Б. Філліпс та ін. 4-те вид. Pearson Higher Education & Professional Group, 2019. 624 с.

3. Бойко І.В., Петрик М.Р., Цуприк Г.Б. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

4. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90.

5. Information System for Design of Thin Multilayer Film Processes Parameters Management based on Diffusion M.R., Petryk, Mykhaylo R., V., Chyzh, Vitalii, H.B., Tsupryk, H. B., O.Y., Petryk, Oksana Yu ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, October 23- 25, 2024, Ternopil, Ukraine, Opole, Poland, 2024, pp. 486-493

УДК 681.518

А.М. Ковтко; В.Б. Савків, к.т.н., доц.; Г.В. Козбур, к.т.н., доц.; І.Р. Козбур
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АНАЛІЗ ЕФЕКТИВНОСТІ МУТАЦІЙНОГО ТЕСТУВАННЯ

A.M. Kovtko; V.B. Savkiv, Ph.D., Assoc. Prof.; H.V. Kozbur, Ph.D., Assoc. Prof.; I.R. Kozbur;

ANALYSIS OF THE EFFECTIVNES OF MUTATIONAL TESTING

Мутаційне тестування програмного забезпечення є перспективним і ефективним підходом до здійснення перевірки якості модульного тестування у сучасній розробці програмних інформаційних систем. Метою мутаційного тестування є оцінка здатності тестових наборів виявляти помилки шляхом створення штучних дефектів, тобто мутацій, у коді програми. Це дозволяє визначити повноту тестових сценаріїв та підвищити надійність, якість і стабільність програмного продукту. Важливість цієї теми зумовлена зростанням складності сучасних програмних продуктів, необхідністю підвищення їхньої якості та зменшення витрат на виправлення дефектів на пізніх етапах розробки. Ефективний аналіз результатів мутаційного тестування дозволяє оптимізувати процеси тестування, підвищити їх результативність та запроваджує більш високі стандарти якості програмного забезпечення.

Генерування модульних тестів є одним з ключових напрямків застосування великих мовних моделей (LLM) у сфері розробки програмного забезпечення [1]. Використання LLM у такому контексті дає змогу істотно скоротити тривалість написання тестів для програмного забезпечення та ефективніше розподілити ресурси команди розробників. Однак, попри ці переваги, згенеровані тести часто характеризуються недостатньою якістю, що зумовлено специфічними похибками («галюцинаціями») у самих мовних моделях [2]. Для подолання цієї проблеми доцільно проводити додатковий аналіз результатів, отриманих після генерації тестів. Авторами у [3, 4] запропоновано автоматизовану систему генерації модульних тестів. LLM генерує модульні тести, які підлягають контролю мутаційним тестуванням. Отримані результати вцілілих мутацій забезпечують уточнення та покращення початково згенерованих тестових випадків шляхом ітеративних звернень до LLM. Фактична реалізація зворотного зв'язку у системі тестування є очевидною ознакою її належності до автоматичних систем. Оскільки ефективність та якість згенерованих модульних тестів суттєво залежать від мутаційного тестування, актуальним завданням є дослідження рівня його впливу на кінцеву якість програмного продукту.