ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ

УДК 004.42:004.92

О. В. ГОРБОВА[1*], О. А. СИРОТА[2*]

[1*]Каф. «Комп'ютерні інформаційні технології», Український державний університет науки і технологій, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (056) 373 15 35, ел. пошта alexandra.gorbova@gmail.com, ORCID 0000-0002-5612-2715
[2*]Каф. «Комп'ютерні інформаційні технології», Український державний університет науки і технологій, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (056) 373 15 35, ел. пошта sirotaalexandr30@gmail.com, ORCID 0000-0001-7391-2471

Дослідження наслідків використання патернів і загальноприйнятих підходів у побудові архітектури кросплатформних додатків

Мета. У наш час, коли важко уявити людину, яка не користувалась би смартфоном, основним напрямком при створенні багатьох додатків є саме такі операційні системи як iOS та Android. Одним із невід'ємних етапів життєвого циклу кросплатформних додатків для iOS та Android є побудова архітектури. Важливо не тільки вміти будувати архітектуру, використовуючи відомі «інструменти», але й розуміти, у якій мірі це повинно бути реалізовано і який вплив це матиме на програмний продукт у подальшому. Основна мета роботи полягає в аналізі підходів до розробки з надмірним або недостатнім використанням принципів і шаблонів проєктування, а також аналіз результатів кінцевого продукту, кросплатформного програмного забезпечення для операційних систем iOS та Android. **Методика.** Для поліпшення наявних програмних засобів спроєктовано та реалізовано статичний аналізатор, який орієнтовано на опрацювання архітектури в програмних засобах різного розміру та типу. Для створення програмного продукту було використано лише такі шаблони й підходи проєктування, які дозволили реалізувати необхідний функціонал, не ускладнюючи систему, та гарантувати легку підтримку, тестування й розширення функціоналу в разі необхідності. **Результати.** Під час проведення експериментів було виявлено, що в ході проєктування кросплатформного програмного забезпечення для операційних систем iOS та Android спостерігається не лише нехтування патернами та загальноприйнятими підходами проєктування, але й надмірне їх використання. Це ускладнює розробку, розширення, підтримку та тестування програмних засобів. **Наукова новизна.** Визначено необхідну міру та наслідки використання шаблонів проєктування, з'ясовано їхню користь та наведено приклади використання патернів і підходів у проєктуванні кросплатформних додатків. Уперше було проведено аналіз необхідної міри використання шаблонів проєктування у різних за розміром та призначенням мобільних додатках. **Практична значимість.** Результати роботи дозволять програмісту краще розуміти, як проєктувати додатки для операційних систем iOS та Android, за яких умов застосувати відомі шаблони проєктування. Отримана інформація може бути використана викладачами закладів вищої освіти для наведення практичних прикладів та демонстрації здобувачам під час виконання практичних робіт, а також програмістами в реальних комерційних проєктах.

Ключові слова: патерн; кросплатформний додаток; операційні системи iOS та Android; проєктування програмного забезпечення; аналіз програмного коду

Вступ

Загальноприйнятими підходами в побудові програмних додатків слід вважати принципи та рекомендації побудови архітектури, які багаторазово перевірені на практиці та зарекомендували себе як спосіб поліпшення програмного забезпечення на різних етапах життєвого циклу. У галузі розробки мобільних додатків або інших продуктів патерном, або шаблоном проєктування (Design Pattern), називають уже

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

знайдений розв'язок задачі, що досить часто зустрічається на практиці. У програмній інженерії шаблон проєктування є загальним повторюваним рішенням поширеної проблеми в розробці програмного забезпечення. Шаблон дизайну – це не готовий дизайн, який можна перетворити безпосередньо в код, а опис, або шаблон, для вирішення проблеми, який можна використовувати в різних ситуаціях [8].

Основна ідея використання патернів полягає у створенні ієрархій класів певного виду під час розв'язання певної задачі. Також потрібно сказати, що застосування патернів або загальноприйнятих підходів не «прив'язується» до конкретної мови програмування, яку використовують, а і підтримується всіма об'єктно-орієнтованими мовами. Основними причинами широкого використання різних шаблонів є такі:

– з'являється можливість для повторного використання коду (code reuse) у цьому ж або іншому програмному додатку;

– є змога легко супроводжувати продукт і додавати, змінювати або видаляти різні частини незалежно одна від одної з мінімальними витратами в часі, кількості розробників та з мінімальною складністю;

– полегшення комунікації розробників завдяки абстракціям, які вносять патерни [8];

– підтримання можливості легко та швидко редагувати всю систему в цілому.

Фабричний метод (Factory Method) – шаблон, який надає можливість дочірнім класам створювати екземпляри певного класу за допомогою абстрактного інтерфейсу, що дозволяє використовувати абстракцію замість конкретної реалізації деяких сутностей.

Принцип SOLID – це список шаблонів або принципів дизайну програмного забезпечення в об'єктно-орієнтованому програмуванні:

– S – принцип єдиної відповідальності (англ. single responsibility principle) свідчить про те, що кожен модуль повинен мати лише одну відповідальність;

– O – принцип відкритості – закритості (англ. open-closed principle) – програмні сутності повинні бути відкриті для розширення і закриті для модифікації;

– L – принцип підстановки Барбари Лісков (англ. Liskov substitution) – підкласи можуть виступати заміною для суперкласів без порушень роботи програми;

– I – принцип розподілу інтерфейсів (англ. interface segregation principle) – декілька вузькоспрямованих інтерфейсів краще, ніж один універсальний;

– D – принцип інверсії залежностей (англ. dependency inversion principle) – залежності між модулями програми повинні будуватися на основі абстракцій та уникати залежностей від конкретної реалізації.

Дотримання цих принципів дозволяє спроєктувати таку систему, яку в подальшому буде легко змінювати, розширювати та підтримувати [3, 7].

KISS-принцип (KISS (Keep it short and simple) principle) – принцип проєктування, який свідчить про те, що програмна система буде працювати краще, якщо буде залишатися максимально простою, а не ускладнюватись [10].

Medium – це швидка та проста в обігу блог-платформа від творців Blogger і Twitter. Місце, де своїми думками діляться користувачі (як у Twitter) та сам Medium (подібно до BuzzFeed). На тему розробки мобільних додатків під платформи iOS та Android написано безліч статей, бо досить велика частина аудиторії – це розробники програмного забезпечення зі знанням різних технологій та різним рівнем досвіду. Інформація, яку пропонують автори цієї платформи, покриває широкий спектр запитань, із якими інший розробник потрапляє на неї. В основному це новини зі світу розробки мобільних додатків, опис нового функціоналу, який був реалізований у новій версії фреймворку, стислий переказ офіційної документації, приклади використання тієї чи іншої бібліотеки, приклади реалізації різного функціоналу, і звичайно, статті на тему конструювання архітектури, огляд патернів та загальноприйнятих підходів під час конструювання крос-платформних мобільних додатків [2].

Stack Overflow – система запитань і відповідей про програмування, розроблена Джоелем Спольскі та Джеффом Етвудом (англ.) у 2008 році. Перевагою джерела є наявність у базі знань інформації не тільки про специфічні помилки, що призводять до збою в роботі на певній платформі, наприклад Android, а й пояс-

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

нення патернів, приклади та ситуації необхідності їх використання, наслідки. І навіть якщо щось не вдалося знайти, то можна ввести запитання й отримати на нього відповідь із поясненням та посиланнями на інші літературні джерела.

Habr – вебсайт у форматі системи тематичних колективних блогів (іменованих хабами) з елементами новинного сайту, створений для публікації новин, аналітичних статей, думок, пов'язаних з інформаційними технологіями, бізнесом та інтернетом [11]. Більшість статей на тему розробки програмного забезпечення досить об'ємні та містять багато інформації, яка базується на інших літературних джерелах, таких як книги, офіційна документація, інші статті та власний досвід.

Крім цього, існує багато офіційних джерел з офіційною документацією, одним із них є сервіс від компанії Microsoft.

Бібліотека MSDN (англ. MSDN Library) – бібліотека офіційної технічної документації для розробників під ОС Microsoft Windows. Бібліотека також містить інформацію про патерни проєктування. Викладення інформації високого рівня доповнено схемами, прикладами коду, детальним поясненням, за що відповідає той чи інший компонент в реалізації, нотатками з порадами та попередженнями про типові помилки. Також надно детальне пояснення, навіщо взагалі використовувати патерн, про який іде мова в документації, та з яких сутностей буде складатися проєкт у разі його реалізації.

Під час ознайомлювання з пропонованою інформацією можна дізнатися дуже багато про патерни та загальні підходи щодо побудови кросплатформних мобільних додатків, побачити різні приклади їх реалізації тощо. Майже вся інформація – це переказ якоїсь частини книги з накладанням певного досвіду.

Усі літературні ресурси поєднує те, що більшість інформації припадає на пояснення шаблону, його реалізації та прикладів самої реалізації, і досить мала частка припадає на пояснення ситуацій, коли, в якій мірі необхідно використовувати і чи потрібно взагалі використовувати той чи інший патерн/підхід, у проєкті якого розміру чи типу, із яким функціоналом, бюджетом, замовником.

У результаті оцінки програмних продуктів, орієнтованих на аналіз архітектури, наявності реалізації патернів, якості цієї реалізації та використання загальноприйнятих підходів у певному проєкті стає зрозуміло, що більшість програмних аналогів займаються пошуком програмних помилок, і лише декілька надають функціонал для перевірки архітектурної складової та є платними. Досить часто аналіз архітектури не автоматизовано.

Досить важко з упевненістю сказати, що аналізатор будуть використовувати в невеликих компаніях і в невеликих проєктах, бо з фінансової точки зору це дорогий процес, а безкоштовні версії являють собою недостатній набір функцій та обмежені в часі використання. Крім того, усі виявлені проблеми є можливими, а не точними. Це свідчить про те, що цей інструмент не гарантує точно, чи є ця помилка в коді насправді проблемою, тому часто результат аналізу коду є помилковим.

Перевагами статичного аналізу коду є відсутність потреби в астрономічних прогонах програм за різних умов функціонування та можливість домогтися більшої міри автоматизації перевірок на наявність дефектів програм за їх конструктивними ознаками [1].

Мета

Основною метою цієї статті є створення методики, за допомогою якої можна виконати аналіз підходів до розробки з надмірним або недостатнім використанням принципів і шаблонів проєктування, а також проаналізувати результати кінцевого продукту, кросплатформного програмного забезпечення для операційних систем iOS та Android, їх вплив на складність розуміння реалізації, існування, функціонування, підтримку та розширення можливостей програмного продукту з точки зору розробника.

Дослідження проведено в рамках магістерської роботи [5].

Методика

Задачею експериментального методу дослідження є доведення або спростування гіпотези про те, що архітектура кросплатформного до-

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

датку, яка розроблена з надмірним застосуванням патернів і загальноприйнятих підходів проєктування, має не менш негативні наслідки щодо підтримки, розширення, повторного використання окремих модулів, вартості мобільного додатку, ніж архітектура, розроблена з нехтуванням необхідних патернів і принципів проєктування.

Під час розробки алгоритму використовують тенденції, методики, методи та підходи в сукупності. Виходячи з того, що для доведення чи спростування гіпотези необхідно розробити алгоритм для аналізу архітектури програмного забезпечення, слід приділити увагу наявним методам і підходам.

Аналіз архітектури додатків – критично важлива задача, оскільки дозволяє скоротити витрати на виправлення помилок, якомога раніше виявити та виправити можливі проблеми. Аналіз архітектури виконують часто після завершення основних етапів проєкту та у відповідь на істотні зміни в архітектурі.

Хоча архітектура програмного забезпечення є досить поширеною темою дослідження в останні роки, недостатньо уваги приділено методам оцінки архітектури. Оцінити архітектуру складно з двох основних причин:

1) не існує спільної мови для опису різних архітектур;

2) немає чіткого способу розуміння архітектури щодо проблем якості програмного забезпечення, таких як ремонтопридатність, переносимість, модульність, можливість повторного використання тощо.

Основна задача аналізу архітектури – підтвердження відповідності базової архітектури та її можливих варіантів, а також перевірка відповідності запропонованих технічних рішень функціональним вимогам та параметрам якості. Крім того, аналіз допомагає виявити проблеми й області, які потребують виправлень або доопрацювань [1].

Оцінка на основі сценаріїв – це потужний метод аналізу дизайну архітектури. За такого оцінювання основну увагу спрямовано на найважливіші з точки зору бізнесу сценарії, а також на ті, що мають великий вплив на архітектуру. Розглянемо типові методи [1].

Метод аналізу архітектури програмного забезпечення SAAM (Software Architecture Analysis Method) використовують для визначення того, як були досягнуті конкретні атрибути якості програми та як можливі зміни в майбутньому вплинуть на атрибути якості на основі гіпотетичних ситуацій. Загальні атрибути якості, які можна використовувати за допомогою цього методу, включають можливість модифікації, надійність, переносимість і розширюваність [1, 4].

Метод аналізу архітектури програмного забезпечення SAAM можна застосувати для двох різних аналізів: для порівняння двох або більше варіантів дизайну, щоб зрозуміти, який із них відповідає вимогам до якості краще; для оцінки дизайну, щоб виявити місця, де архітектура не відповідає вимогам до якості [4].

Атрибут «Можливість модифікації програми» (англ. Application Modifiability) вказує на те, наскільки легко можна змінити систему в майбутньому. Для того щоб SAAM можна було належним чином застосувати для перевірки цього атрибута, необхідно створити конкретні гіпотетичні приклади. Такими є заміна модуля для оплати продуктів або додавання додаткового виду платежу в мобільному додатку, що надає змогу проводити покупки онлайн. Суть цього атрибуту якості має спільні риси з принципами SOLID [4].

Атрибут якості «Надійність застосування» (англ. Application Robustness) стосується до того, як програма справляється з несподіваними процесами.

Атрибут якості «Переносимість програми» (англ. Application Portability) стосується до простоти перенесення програми для запуску в новій операційній системі або на новому пристрої. Наприклад, набагато простіше змінити вебсайт ASP.net, щоб він був доступним для Windows, Mac, IPhone, Android порівняно з настільною програмою, написаною VB.net, тому що знадобилося б провести набагато більше роботи, щоб довести програму до стану, коли її можна буде використовувати на різних платформах. Цей атрибут якості повинен бути оцінений на основі кожного нового середовища, для якого проведено гіпотетичне дослідження.

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

Особливої уваги потребують:
– питання портативності;
– апаратні залежності;
– залежності операційної системи;
– залежність від джерела даних;
– залежність від доступу до мережі [1].

Атрибут якості «Розширюваність програми» (англ. Application Extensibility) стосується простоти додавання нових функцій до наявної програми без впливу на наявну функціональність. Аспекти, що впливають на цей атрибут якості, – це наявність жорстко закодованих змінних, наявність документації програмного продукту, використання принципів SOLID [1].

Метод аналізу архітектурних компромісів ATAM (англ. Architecture Tradeoff Analysis) – це доопрацьована та поліпшена версія SAAM, яка дозволяє переглядати архітектурні рішення щодо вимог параметрів якості та того, наскільки добре ці рішення відповідають конкретним цільовим показникам якості. ATAM є найбільш корисним, коли використовується на ранніх етапах життєвого циклу програмного забезпечення, коли вартість змін в архітектурі є мінімальною. Схематичне зображення цього методу показано на рис. 1 [1, 15].

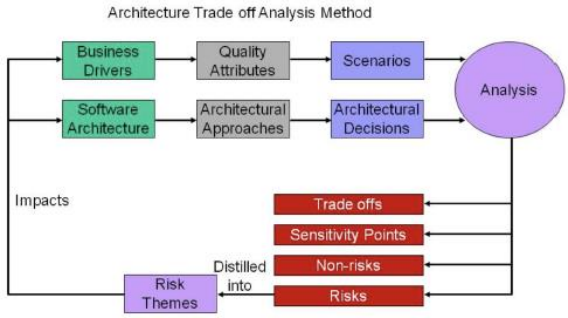

Рис. 1. Схема методу аналізу архітектурних компромісів

Fig. 1. Scheme of the method of architectural compromise analysis

Переваги використання методу аналізу архітектурних компромісів:
– виявлення ризиків на початку життєвого циклу;
– наявність конкретних вимог до атрибутів якості;
– поліпшена архітектурна документація;
– документована основа архітектурних рішень.

Процес ATAM складається зі збору зацікавлених сторін для аналізу бізнес-драйверів (функціональність системи, цілей, обмежень, бажаних нефункціональних властивостей) і вилучення з цих драйверів атрибутів якості, які використовують для створення сценаріїв. Ці сценарії потім використовують у поєднанні з архітектурними підходами та архітектурними рішеннями для створення аналізу компромісів, точок чутливості та ризиків (або не-ризиків). Цей аналіз можна перетворити на теми ризику та їх вплив, після чого процес можна повторити. Із кожним циклом процес аналізу переходить від більш загального до більш конкретного, вивчаючи питання, які були виявлені в попередньому циклі, до тих пір, поки архітектура не буде точно налаштована і не будуть розглянуті теми ризику [12].

Метод аналізу рентабельності CBAM (англ. Cost Benefit Analysis Method) основну увагу приділяє аналізу витрат, вигод та планування наслідків архітектурних рішень.

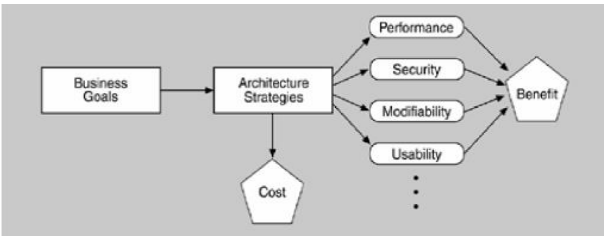

Рис. 2. Складові методу аналізу рентабельності

Fig. 2. Components of the method of profitability analysis

Метод оцінки сімейства архітектур FAAM (англ. Family Architecture Assessment Method) оцінює архітектури сімейства інформаційних систем щодо можливості взаємодії та розширюваності [1].

Аналіз модифікованості лише на рівні архітектури ALMA (англ. Architecture Level Modifiability Analysis) оцінює модифікованість архітектури для систем бізнес-аналітики (англ. business information systems, BIS) [1].

Для формалізації задачі була розроблена діаграма варіантів використання, на якій відображено можливі варіанти використання системи актором, що представляє користувача. Варіанти використання системи показані на рис. 3.

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

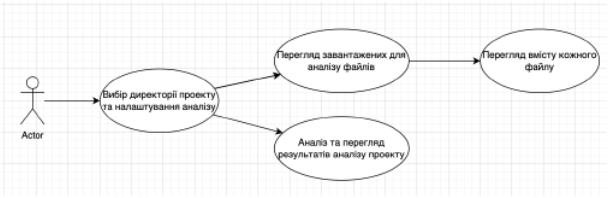

Рис. 3. Схема варіантів використання

Fig. 3. Scheme of variants of use

Під час проєктування інструментального забезпечення було використано шаблон MVVM – «Модель – вигляд – стосується Модель вигляду» (англ. Model-View-ViewModel), які використовують для розподілу архітектури програмного забезпечення «Static architecture analyzer» на три рівні. Схему розподілу зображено на рис. 4.

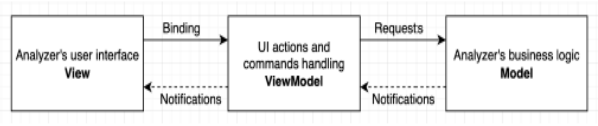

Рис. 4. Схема архітектури «Static architecture analyzer» за патерном MVVM

Fig. 4. Static architecture analyzer scheme according to the MVVM pattern

Розподіл архітектури на три рівні дозволяє визначити обов'язки кожного модуля, надає можливість паралельної розробки модулів незалежно один від одного, а також полегшує процес тестування, бо кожен із модулів може бути протестовано окремо один від одного. У свою чергу модулі патерну MVVM містять функціональність за типом «Модель» (Model) Модулі за таким типом представляють бізнес-логіку проєкту та моделі даних, які використовує бізнес-логіка проєкту. Ці моделі даних можуть бути збережені в базі даних, використовуватися для обміну даними між сервісами або можуть бути надані рівню ViewModel як результат якоїсь операції. Бізнес логіка на рівні – «Модель» поділяється на сервіси, які є підмодулями з реалізацією певного функціоналу. Прикладом може бути сервіс із реалізації CRUD (Create-Read-Update-Delete) – операцій із замітками користувача, результатом виконання яких буде модель «Нотатка». У шаблоні MVVM залежність рівнів між собою однонаправлена, тобто влаштована так, що «Модель вигляду» має залежність від «Моделі», а «Модель», у свою чергу, немає залежності від жодного з рівнів. На рівні «Модель» недопустимо описувати логіку інтерфейсу користувача або логіку взаємодії з ним [11].

«Вигляд» (View) описує реалізацію інтерфейсу користувача (англ. User Interface). Це може бути сторінка мобільного, десктопного або вебдодатку.

Рівень інтерфейсу користувача часто поділяється на підмодулі, залежить від рівня «Модель вигляду», але не повинен реалізовувати бізнес-логіку чи логіку щодо роботи з даними [12].

«Модель вигляду» (англ. ViewModel) займається обробкою команд від інтерфейсу користувача та запитами до рівня «Модель», щоб потім отримані дані були відображені на рівні «Вигляд». Цей компонент є мостом між рівнем представлення даних та роботою з ними. Передача інформації до інтерфейсу користувача реалізується завдяки механізму зв'язування (англ. Binding). Цей рівень не повинен містити реалізації роботи з даними [10].

Під час розробки статичного аналізатора було застосовано об'єктно-орієнтовану мову програмування C# та мову розмітки для додатків XAML. XAML дозволяє розробникам визначати інтерфейси користувача в програмах Xamarin.Forms, використовуючи розмітку, а не код. Додатки Xamarin.Forms можуть бути розроблені і без застосування XAML, але вона часто є більш стислою і візуально узгодженою, ніж еквівалентний код. Ця мова добре підходить для використання з популярним архітектурним патерном MVVM (Model-View-ViewModel): за допомогою XAML реалізується View-рівень, який пов'язаний із кодом рівня ViewModel через прив'язки даних.

Проєктування системи відбувалось за допомогою створення діаграми класів. Взаємодія різних рівнів показана на рис. 5.

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

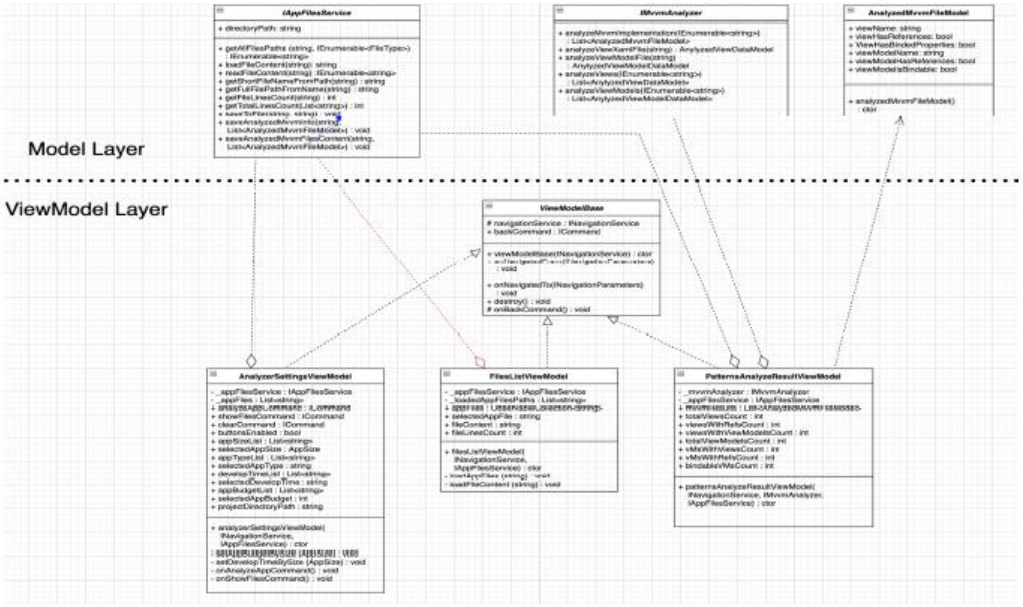

Рис. 5. Взаємодія між рівнями ViewModel та View

Fig. 5. Interaction between the levels ViewModel and View

Відкритості–закритості (англ. Open-closed) – функціонал сутностей програми може розширювати завдяки поліморфізму, але початкові сутності залишаються незмінними. Цьому сприяє такий ООП-принцип, як поліморфізм.

Підстановка Барбари Лісков (англ. Liskov substitution) – у спроєктованій архітектурі спадкоємці базової моделі можуть бути використані замість базової, і правильність виконання програми при цьому не постраждає. Принцип простоти (KISS) – розроблені модулі та сам програмний додаток максимально просто реалізовані, не перевантажені непотрібними архітектурними рішеннями та функціоналом.

Абстракція та інкапсуляція – принципи, які мають іти в парі один з одним, бо лише в парі вони гарантують відділення деталей реалізації від інтерфейсу певного модуля. Завдяки цим принципам програмні інтерфейси сервісів не несуть ніяких деталей про те, як вони реалізовані на тому рівні, на якому ці інтерфейси використовуються. Інтерфейс кожного із сервісів лише свідчить про те, що він робить, і нічого про те, як.

Поліморфізм – завдяки йому одні моделі програми розширюють функціонал інших моделей, тим самим створюючи декілька реалізацій одного інтерфейсу батьківського класу, та дозволяють під час опису залежностей між модулями абстрагуватися завдяки використанню базової моделі – суперкласу, замість використання конкретної [9].

На базі розглянутих методів було спроєктовано та реалізовано статичний аналізатор, який орієнтовано на опрацювання архітектури в програмних засобах різного розміру та типу. Для створення програмного продукту було використано лише необхідні шаблони та підходи проєктування, що дозволило реалізувати необхідний функціонал, не ускладнити систему та гарантувати легку підтримку, тестування й розширення функціоналу за необхідності.

Результати

Невід’ємною частиною будь-якого експерименту є етап підготовки даних, які використовують відповідно до предметної галузі дослідження. У нашій роботі вхідними даними є набір проєктів реалізації деяких кросплатформних програмних продуктів для операційних систем Android та iOS. Приклад структури одного з проєктів наведено на рис. 6. Фактичний розмір *S* кожного проекту на етапі аналізу оцінено згідно з моделлю COCOMO в рядках коду KLOC.

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

Розмір проєктів, що будуть використані як вхідні дані для експериментів:

1) KLOC першого проєкту дорівнює 4 270 рядків;

2) KLOC другого становить 3 710 рядків;

3) KLOC третього – 3 674 рядки;

4) KLOC четвертого складає 2 425;

5) KLOC четвертого – 4 303.

Усі вони реалізовані об’єктно-орієнтованою мовою програмування C# та з використанням фреймворку Xamarin.Forms, що має необхідність реалізувати патерн MVVM.

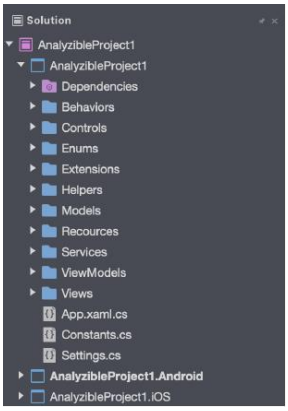

Рис. 6. Типова структура проєкту, що використовується як вхідні дані

Fig. 6. Typical project structure used as input data

Суть проведення експерименту полягає в аналізі архітектури різних кросплатформних проєктів за допомогою статичного аналізатора коду з використанням підготовлених вхідних даних та подальшим аналізом результатів цього експерименту. Проведення експерименту складається з таких дій:

1) запуск статичного аналізатора (рис. 7);

2) вказання директорії проєкту, що потрібно проаналізувати;

3) проведення додаткового налаштування аналізатора;

4) перегляд результатів аналізу архітектури проєкту (рис. 8).

Оскільки результатами аналізу є файли з основними й додатковими даними, а також візуальне їх представлення, то результати першого та всіх наступних експериментів буде представлено на рисунках та у вигляді двох таблиць з основними результатами аналізу й додатковими.

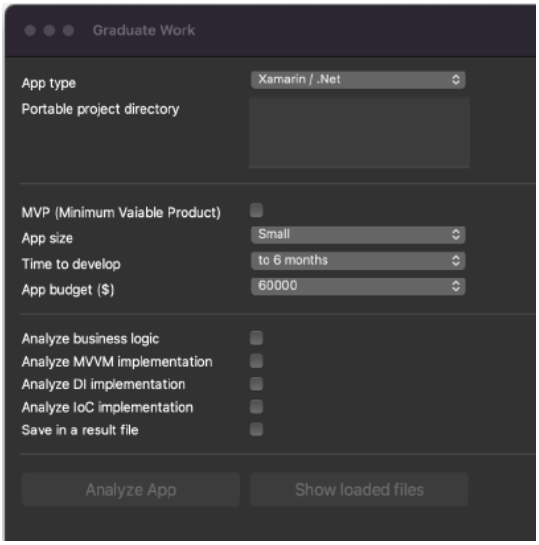

Рис. 7. Екранна форма статичного аналізатора

Fig. 7. Screen form of the static analyzer

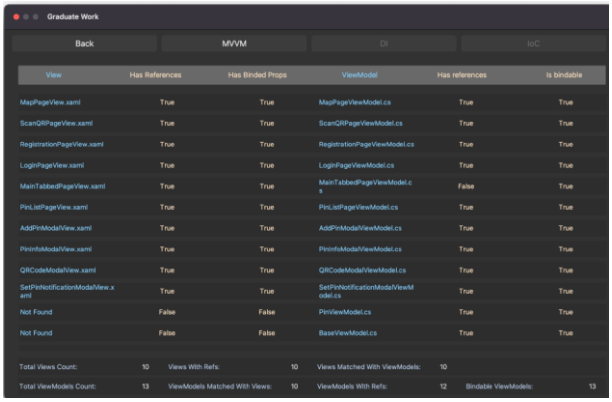

Рис. 8. Перегляд результатів аналізу архітектури

Fig. 8. Review of the architecture analysis results

У представлений проєктах, із застосуванням розробленого додатку, отримано результати дослідження наслідків використання патернів і загальноприйнятих підходів у побудові архітектури кросплатформних додатків для Android і iOS. Ураховуючи той факт, що патернів проєктування існує велика кількість, аналіз результатів ми провели, базуючись на результатах реалізації патерна MVVM, який є одним із найпоширеніших патернів під час побудови мобільних додатків. У ході аналізу ми знехтували файлами з базовою реалізацією логіки інтерфейсу користувача або базовою реалізацією цього інтерфейсу, бо ці файли в цілому не можуть залежати один від одного.

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

Із результатів експерименту, де як вхідні дані виступав найменший проєкт із KLOCK, що дорівнює 2 425 рядків коду, можна зробити висновки, що за наявності трьох файлів інтерфейсу користувача та трьох файлів логіки інтерфейсу користувача проєкт реалізує патерн MVVM на 100 %, а файлів, що не використовуються, узагалі немає. Відзначимо що складність підтримки або розширення цього проєкту низька, а отже, і вартість процесу теж не перевищує розумні показники. Проте реалізація патерна займає *t* год, що залежать від кваліфікації розробника, а у разі нехтування його реалізації час на розробку проєкту зменшується на ті ж самі *t* год, що певною мірою знижує його вартість. Та насправді ціна проєкту лише зросте через те, що процес підтримки стане набагато складнішим із плином часу, якщо програмний продукт планують розвивати в майбутньому. Станеться це тому, що зі зростанням кількості сутностей програмний код буде ускладнюватися, у результаті модулі буде майже неможливо протестувати або розширити незалежно один від одного. Зростання кількості помилок заважатиме реалізації нового функціоналу, а вартість збільшуватиметься з кожною новою задачею.

За результатами експериментів, для яких були використані проєкти середнього розміру з KLOCK, що дорівнює рівним 3 710 та 3 674 рядки видно що в другому проєкті патерн реалізовано приблизно на 80–90 %, бо деякі з файлів інтерфейсу та логіки інтерфейсу користувача не зв'язані між собою за механізмом прив'язки. Третій проєкт реалізує шаблон MVVM на 100 %. Як і в попередньому випадку, такий відсоток дає гарантію того, що складність підтримки продукту та його вартість буде невеликою навіть у разі росту проєкту.

З аналізу результатів першого та п'ятого експериментів, де вхідними даними були проєкти найбільшого розміру з показниками KLOCK 4 720 та 4 303 рядки вихідного коду, видно, що для першого проєкту патерн MVVM реалізовано приблизно на 0 %, тобто не реалізовано взагалі. Із результатів попередніх експериментів відразу стає зрозуміло, що складність розширення та підтримки цього програмного продукту буде досить швидко зростати, збільшуючи вартість розробки. Якщо не провести рефакторинг, то це може завдати великих збитків або навіть спричинити провал ідеї та роботи компанії. У випадку з п'ятим проєктом патерн реалізовано на 100 %, що на перший погляд спрощує етапи тестування, підтримки та розширення функціоналу програмного додатку. Але з розмірів проєкту та можливого його росту в майбутньому неважко зрозуміти, що і кількість сутностей зростатиме. Цей факт буде ускладнювати процес розробки, адже сутності в проєкті утворюють граф залежностей одна від одної, і чим він більший, тим складніше розробнику тримати його в пам'яті під час роботи над проєктом. Щонайменше це призведе до більших часових витрат. Також це може стати основою для майбутніх помилок на етапі розробки програмного засобу.

На практиці великі проєкти реалізують не один патерн або архітектурний підхід, що збільшує складність на всіх етапах розробки. З отриманих результатів випливають такі тенденції:

– зменшення вартості продукту шляхом зменшення необхідного часу на його розробку за рахунок нехтування патернами проєктування – насправді лише ілюзія. У майбутньому складність розробки буде збільшуватися в геометричній прогресії, що, без сумніву вплине на його вартість в гіршу сторону;

– зменшення вартості проєкту за рахунок зменшення необхідного часу на розробку за умови нехтування реалізацією певного патерна можна допускати лише в тому разі, коли проєкт на планують підтримувати та супроводжувати в майбутньому. Прикладом такої ситуації може бути розробка MVP–додатку. Найчастіше це трапляється на проєктах невеликого розміру;

– реалізація шаблонів проєктування може спрощувати процес підтримки програмного засобу та утримувати вартість цього процесу на адекватному рівні в майбутньому;

– збільшення кількості реалізованих патернів та архітектурних підходів у проєкті з часом ускладнює його розуміння, сповільнює розробку та слугує основою для подальших помилок у функціоналі, що також призводить до збільшення його вартості. Бажання покращити архітектуру завдяки імплементації додаткового шаблону може призвести до його погіршення.

doi: https://doi.org/10.15802/stp2021/258109

Наукова новизна та практична значимість

У роботі вперше створено метод та інструменту для визначення наслідків використання шаблонів, з'ясовано їхню користь, наведено приклади використання патернів та підходів у побудові кросплатформних застосунків під час реалізації проєктів різного масштабу та з різною функціональністю. Уперше проведено аналіз необхідної міри використання шаблонів проєктування у різних за розміром та призначенням мобільних додатках.

Висновки

У результаті досліджень було виявлено, що не лише нехтування патернами та загальноприйнятими підходами проєктування, але й надмірне їх використання ускладнює розробку, розширення, підтримку та тестування програмних засобів. Також це накладає додаткове навантаження на розробників, що призводить до помилок у коді, а в сукупності – до погіршення архітектури проєкту та збільшення його вартості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кудрявцев В. В. *Автоматизована система аналізу програмного коду для оцінки ризиків та забезпечення якості програмного забезпечення* : дипломна робота магістра. Хмельниц. нац. ун-т. Хмельницький, 2021. 122 с.
2. Мнушка О. В., Котенко Б. О., Савченко В. М. Аналіз вимог та розроблення прототипу навчального програмного забезпечення для мобільних платформ. *Вісник ХНАДУ*. 2021. Вип. 92, Т. 1. С. 51–59. DOI: https://doi.org/10.30977/bul.2219-5548.2021.92.1.51
3. Мнушка О. В., Савченко В. М., Маций О. Б. *Об'єктно-орієнтоване програмування мовою Python*. Харків : ХНАДУ, 2021. 228 с.
4. *Руководство Microsoft по проектированию архитектуры приложений*. 2009. URL: https://dut.edu.ua/uploads/l_1507_99407341.pdf
5. Сирота О. А. *Дослідження наслідків використання патернів і загальноприйнятих підходів в побудові архітектури крос-платформних додатків під Android і IOS* : дипломна робота на здобуття кваліфікаційного ступеня магістра. Укр. держ. ун-т науки і технологій. Дніпро, 2021. 204 с.
6. Alaoui L., Penta A. Cost-Benefit Analysis in Reasoning. *Journal of Political Economy*. 2022. Vol. 130, № 4. P. 881–925. DOI: https://doi.org/10.1086/718378
7. Architecture tradeoff analysis method (ATAM). *Concise Software*. URL: https://concisesoftware.com/architecture-tradeoff-analysis-method-atam
8. *Architecture tradeoff analysis method*. URL: https://en.wikipedia.org/wiki/Architecture_tradeoff_analysis_method
9. Davami F., Adabi S., Rezaee A., Rahmani A. M. Fog-based architecture for scheduling multiple workflows with high availability requirement. *Computing*. 2022. Vol. 104. Iss. 1. P. 169–208. DOI: https://doi.org/10.1007/s00607-021-00905-1
10. Kadri S., Aouag S., Hedjazi D. MS-QuAAF : A generic evaluation framework for monitoring software architecture quality. *Information and Software Technology*. 2021. Vol. 140. P. 106713. DOI:https://doi.org/10.1016/j.infsof.2021.106713
11. Kumar A., Anand E., Natarajan S., Dandekar A. Architecture Analysis Methods. *Inclose International Symposium.* 2021. Vol. 31. Iss. 1. P. 1377–1392. DOI: https://doi.org/10.1002/j.2334-5837.2021.00907.x
12. Kumar A. G. *SOLID Principles Succinctly*. 2016. URI: https://lib.hpu.edu.vn/handle/123456789/24913
13. Shahbazi Z., Rasoolzadegan A., Purfallah Z., Jafari Horestani S. A new method for detecting various variants of GoF design patterns using conceptual signatures. *Software Qual J*. 2021. DOI: https://doi.org/10.1007/s11219-021-09576-9
14. Software Architecture Analysis Method (SAAM). *Dzone*. URL: https://dzone.com/articles/software-architecture-analysis
15. Wijerathna L., Aleti A., Bi T., Tang A. Mining and relating design contexts and design patterns from Stack Overflow. *Empirical Software Engineering*. 2022. Vol. 27, № 8. DOI: https://doi.org/10.1007/s10664-021-10034-0

doi: https://doi.org/10.15802/stp2021/258109

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 6 (96)

O. V. HORBOVA[1*], O. A. SYROTA[2*]

[1*]Dep. «Computer Information Technologies», Ukrainian State University of Science and Technologies, Lazaryana St., 2, Dnipro, Ukraine, 49010, tel. +38 (056) 373 15 35, e-mail alexandra.gorbova@gmail.com, ORCID 0000-0002-5612-2715
[2*]Dep. «Computer Information Technologies», Ukrainian State University of Science and Technologies, Lazaryana St., 2, Dnipro, Ukraine, 49010, tel. +38 (056) 373 15 35, e-mail sirotaalexandr30@gmail.com, ORCID 0000-0001-7391-2471

Research of the Use Consequences of Patterns and Common Approaches in the Architecture Development of Cross-Platform Applications

Purpose. Nowadays, it is difficult to imagine a person who would not use a smartphone. The main direction in creating many applications are such operating systems as iOS and Android. One of the essential stages of the life cycle of cross-platform applications for iOS and Android is building architecture. It is important not only to be able to build an architecture using well-known «tools», but also to understand to which extent this should be implemented and what impact it will have on the software product in the future. The aim of the work is to analyze approaches to development with excessive or insufficient use of design principles and templates, as well as analysis of the results of the final product, cross-platform software for iOS and Android operating systems. **Methodology.** To improve the existing software, a static analyzer has been designed and implemented, which is focused on the development of architecture in software of different sizes and types. Only the necessary templates and design approaches which allowed implementing the necessary functionality without complicating the system and guarantee easy support, testing and extension of functionality if it is needed, were used to create the software product. **Findings.** During the experiments, it was found that when designing cross-platform software for iOS and Android operating systems, not only neglect of patterns and common design approaches, but also their excessive use complicates the development, expansion, maintenance and testing of software. **Originality.** The extent and consequences of using the design templates, the benefits and examples of using the patterns and approaches in the design of cross-platform applications were determined. For the first time, the necessary degree of use of design templates for mobile applications of different sizes and purposes was analyzed. **Practical value.** The results of the work will allow the programmer to better understand the principles of designing the applications for iOS and Android operating systems, as well as the conditions of use of the known design templates. The information obtained can be used by teachers of higher education institutions as a means of providing practical examples and demonstrations for higher education students in performing practical work and programmers on real commercial projects.

Keywords: pattern; cross-platform application; iOS and Android operating systems; software design; program code analysis

REFERENCES

1. Kudryavtsev, V. V. (2021). *An automated system for analyzing the software code for assessing the risks and security of software security*: master degree work. Khmelnytskyi National University. Khmelnitsky, Ukraine. (in Ukrainian)
2. Mnushka, O. V., Kotenko, B. O., & Savchenko, V. M. (2021). Analyzing requirements and developing prototype of training software for mobile. *Bulletin of Kharkov National Automobile and Highway University*, *92*(1), 51-59. DOI: https://doi.org/10.30977/bul.2219-5548.2021.92.1.51 (in Ukrainian)
3. Mnushka, O. V., Savchenko, V. M., & Matsyy, O. B. (2021). *Obiektno-oriientovane prohramuvannia movoiu Python*. Kharkiv: KhNADU. (in Ukrainian)
4. *Rukovodstvo Microsoft po proektirovaniyu arkhitektury prilozheniy*. (2009). Retrieved from https://dut.edu.ua/uploads/l_1507_99407341.pdf (in Russian)
5. Sirota, O. A. (2021). *Research of the Consequences of using Patterns and Common Approaches in Building the Architecture of Cross-Platform Applications for Android and IOS*: diploma work for obtaining the qualification degree of master. Ukrainian State University of Science and Technologies. Dnipro, Ukraine. (in Ukrainian)
6. Alaoui, L., & Penta A. (2022). Cost-Benefit Analysis in Reasoning. *Journal of Political Economy*, *130*(4). DOI: https://doi.org/10.1086/718378 (in English)
7. Architecture tradeoff analysis method (ATAM). *Concise Software*. Retrieved from https://concisesoftware.com/architecture-tradeoff-analysis-method-atam (in English)

doi: https://doi.org/10.15802/stp2021/258109

8. *Architecture tradeoff analysis method*. Retrieved from https://en.wikipedia.org/wiki/Architecture_tradeoff_analysis_method (in English)
9. Davami, F., Adabi, S., Rezaee, A., & Rahmani, A. M. (2021). Fog-based architecture for scheduling multiple workflows with high availability requirement. *Computing*, *104*(1), 169-208. DOI: https://doi.org/10.1007/s00607-021-00905-1 (in English)
10. Kadri, S., Aouag, S., & Hedjazi, D. (2021). MS-QuAAF: A generic evaluation framework for monitoring software architecture quality. *Information and Software Technology*, *140*, 106713. DOI:https://doi.org/10.1016/j.infsof.2021.106713 (in English)
11. Kumar, A., Anand, E., Natarajan, S., & Dandekar, A. (2021). Architecture Analysis Methods. *Inclose International Symposium*, *31*(1), 1377-1392. DOI: https://doi.org/10.1002/j.2334-5837.2021.00907.x (in English)
12. Kumar, A. G. (2016). *SOLID Principles Succinctly*. Retrieved from https://lib.hpu.edu.vn/handle/123456789/24913 (in English)
13. Shahbazi, Z., Rasoolzadegan, A., Purfallah, Z., & Jafari Horestani, S. (2021). A new method for detecting various variants of GoF design patterns using conceptual signatures. *Software Qual J*. DOI: https://doi.org/10.1007/s11219-021-09576-9 (in English)
14. Software Architecture Analysis Method (SAAM). *Dzone*. Retrieved from https://dzone.com/articles/software-architecture-analysis (in English)
15. Wijerathna, L., Aleti, A., Bi, T., & Tang, A. (2022). Mining and relating design contexts and design patterns from Stack Overflow. *Empirical Software Engineering*, *27*(8). DOI: https://doi.org/10.1007/s10664-021-10034-0 (in English)

Надійшла до редколегії: 13.08.2021
Прийнята до друку: 13.12.2021

doi: https://doi.org/10.15802/stp2021/258109