УДК 004.4'24

ЖУРБА М. А.,
СТЕЦЕНКО І. В.

ПОЛІПШЕННЯ ПІДХОДУ СТВОРЕННЯ НОВИХ ЦІЛЕЙ ПРОЄКТУ XCODE З ВИКОРИСТАННЯМ СКРИПТІВ ДЛЯ КОДОГЕНЕРАЦІЇ

У даному науковому дослідженні проаналізовано класичний метод створення нових цілей у проєкті Xcode, запропоновано поліпшений підхід із створення нової цілі шляхом використання скрипту для генерації проєктного файлу та скрипту для генерації конфігураційного файлу, розроблено скрипт для оновлення конфігураційного файлу проєкту, розроблено застосунок для ілюстрації запропонованого підходу.
XCODE ЦІЛЬ, XCODE ПРОЄКТ, ПРОДУКТ, ЗАСТОСУНОК, СКРИПТ, ГЕНЕРАЦІЯ ФАЙЛУ, АВТОМАТИЗАЦІЯ

This research analyzes the classic method of creating new targets in Xcode project, proposes an improved approach for new targets creation by using a script to generate a project file and a script to generate a configuration file, developed a script to update the project configuration file, developed an application to illustrate the proposed approach.
XCODE TARGET, XCODE PROJECT, PRODUCT, APPLICATION, SCRIPT, FILE GENERATION, AUTOMATISATION

1. Вступ

Кожен бізнес прагне ефективно витрачати кошти на кожному етапі життєвого циклу продукту або сервісу. У випадку створення мобільних застосунків у бізнесі досить часто виникає потреба у схожих між собою застосунках з тих чи інших причин. Такими причинами можуть бути, наприклад, необхідність у окремих застосунках для різних регіонів, необхідність у окремих застосунках для дочірніх брендів, прагнення зайняти домінуючу позицію у певній ніші шляхом створення різних застосунків під окремими брендами, використання різних стратегій просування для різних застосунків. У будь-якому випадку створення нових застосунків з нуля та паралельна підтримка усієї множини застосунків не є ефективним рішенням. Не тільки тому, що вона вимагатиме великих затрат фінансів та часу, але ще й тому, що збільшується ризик виникнення помилок. Може трапитись ситуація, при якій новий функціонал був реалізований тільки у одному із застосунків усієї множини через виникнення людських похибок, таких як неуважність або забування.

Саме тому для таких розробок використовують підходи із перевикористанням коду і побудовою кінцевих продуктів на одній кодовій базі. У контексті iOS розробки таких підходів декілька: винесення спільного коду у бібліотеки і використання їх у окремих Xcode проєктах, використання окремих цілей для кожного з кінцевих продуктів у рамках одного Xcode проєкту [1], використання окремих схем для кожного з кінцевих продуктів у рамках одного Xcode проєкту [2].

У порівнянні з іншими, підхід із використанням окремих цілей є найбільш гнучким та таким, що мінімізує дублювання коду. Тим не менш, у нього є суттєвий недолік. Підхід вимагає додавання вручну нової цілі та її конфігурації, водночас для кожного файлу проєкту необхідно вказувати цілі, до яких він належить.

2. Огляд класичного підходу із використанням декількох цілей

Для розробки під iOS використовується середовище розробки Xcode. Після створення проєкту Xcode, автоматично створюються базові вихідні файли, ресурсні каталоги, а також ціль для продукту. У цілі є список вихідних файлів для компіляції та список ресурсів (картинок, кольорів, шрифтів та інше). Ці файли можуть належати одразу декільком цілям, а можуть просто знаходитись у проєкті і не належати жодній. Також у кожної цілі є ідентифікатор, за допомогою якого в подальшому App Store і iPhone відрізняє застосунки між собою. Схема визначає

колекцію цілей для побудови та конфігурацію, яку слід використовувати під час побудови.

Таким чином, суть підходу полягає у створенні декількох цілей і встановленні різних ідентифікаторів для кожної з них. Для того, щоб створити нову ціль, необхідно виконати такі дії:

- перейти у Xcode проєкті до розділу з його налаштуваннями;
- у розділі цілей вибрати одну з існуючих цілей;
- обрати «дублювати», при цьому автоматично створиться нова ціль та нова схема для неї, новий файл .plist з базовими конфігураціями цілі. Всі вихідні та ресурсні файли, які належали до початкової цілі, тепер також належатимуть і до новоствореної;
- перейменувати всі новостворені цілі, схеми та файли відповідно до потреб проєкту (за замочуванням назви створюються шляхом додаванням *Copy* до первісних назв);
- у розділі цілей обрати нову ціль, перейти до вкладки *General* та змінити поле *Display Name;*
- у розділі *Build Settings* підрозділі *Packaging* оновити поле *Info.plist* та *Product Bundle Identifier*.

Кожен із цих пунктів алгоритму розробник може пропустити або зробити його неправильно. До того ж, описаний підхід вимагає знань і розуміння того, як влаштоване середовище розробки Xcode, тобто доступний тільки досвідченому розробнику.

Навіть після налаштування цілей, їх потрібно підтримувати у належному стані, а саме, для кожного новоствореного файлу обирати, до яких цілей він належить. Цю дію легко пропустити, адже вона виконується вибором необхідних прапорців, які можна не помітити.

Отже, класичний підхід із використанням декількох цілей вимагає від розробника виконання ряду рутинних дій для створення нової цілі, які потенційно можуть бути не виконані або виконані неправильно. Список вихідних файлів та ресурсних каталогів створюваного застосунку визначається вказуванням відповідних прапорців цілей, що також є незручним. Тому необхідно знайти шляхи вирішення зазначених проблемних моментів.

3. Генерація проєктного файлу за допомогою утиліти для кодогенерації

Перша проблема, яка буде вирішуватися – це проблема встановлення посилань на вихідні файли та ресурсні каталоги для кожної нової цілі. Як вже було зазначено, у проєкті Xcode відповідність файлів та каталогів цілі встановлюється шляхом вказування відповідного прапорця. Але що саме відбувається, коли розробник вмикає або вимикає прапорець цілі для певного файлу? При створенні Xcode проєкту створюється файл з розширенням .xcodeproj. Насправді, це директорія, яка зберігає дані про проєкт. У ній знаходиться файл .pbxproj (Project Builder Xcode Project). Саме у цьому файлі знаходяться всі дані про конфігурації проєкту, його цілі, схеми. Коли розробник корегує прапорець, Xcode оновлює дані у .pbxproj. Для розробника це незручно, адже файл .pbxproj має складний для сприйняття синтаксис.

Саме тому ідея полягає у використанні утиліти XcodeGen[3], яка генерує директорію .xcodeproj на основі заданого конфігураційного файлу project.yaml. Цей підхід має наступні переваги:

- `.yaml` файл більш читабельний у порівнянні із класичним `.pbxproj`;
- для кожної цілі можна вказувати шляхи до директорій із вихідними файлами та ресурсними каталогами;
- зменшення випадків при роботі із системою контроля версій із конфліктами злиття у великих командах;
- створення нової цілі вимагатиме лише копіювання частини коду у файлі.

Як саме треба заповнювати конфігураційний файл проєкту описано у репозиторії XcodeGen. Але загалом результатом дій буде файл project.yaml, який описує налаштування проєкту та його цілей, де для кожної цілі вказуються шляхи до директорій з файлами, які до неї належать.

В решті решт, потрібно лише використати консольну команду xcodegen generate, яка створить .xcodeproj.

Отже, за допомогою XcodeGen і конфігураційного файлу project.yaml можна вирішити проблему рутинного додавання файлів до цілей, натомість можна лише вказати директорії з ними у полі sources. Також, за допомогою конфігураційного файлу, створення нової цілі відбувається простіше, швидше і знижує можливість виникнення

помилок через зменшення кількості операцій, які потрібно виконати розробнику. Проте створення нової цілі відбувається все ще вручну. Далі буде описано, як можна автоматизувати цей процес.

4. Модифікація проєктного файлу за допомогою самописної утиліти

У класичному підході з використанням декількох цілей, для створення нової розробнику спочатку потрібно дублювати існуючу, потім змінити деякі поля і назви у новостворені цілі. Тобто глобально, знаючи назву нової цілі, а також назву та параметри цілі, від якої буде зроблено дублювання, можна автоматизувати процес створення за допомогою написаного власноруч скрипта. Ця ідея і була закладена при розробці утиліти командного рядку igen.

Команда `igen` приймає два параметри: шлях до файлу з конфігураціями цілей та шлях до `project.yaml` проєкту.

Перший файл має два поля:

- project – назва проєкту;
- targets – словник, в якому ключ – це назва цілі, значення – це параметри цілі; параметри можуть мати поля steals – список існуючих папок, які потрібно скопіювати для нової цілі, та inherit – назва цілі, від якої відбувається дублювання.

Утиліта працює за наступним алгоритмом:

- виконує парсинг файлу із конфігураціями цілей;
- виконує парсинг `project.yaml`, звідки дістаються дані про вже існуючі цілі;
- виконує створення моделей для нових цілей на основі вхідних даних;
- виконує кодогенерацію оновленого блоку для цілей файлу project.yaml;
- створює директорії COMMON і <назва_цілі> за допомогою виклику консольної команди `mkdir`;
- якщо у якоїсь цілі було вказано steals, виконує копіювання директорій за допомогою виклику консольної команди `cp -R`;
- оновлює файл project.yaml (оновлюється тільки блок цілей, ключі будуть відсортовані за алфавітним порядком);
- оновлює файл з конфігураціями цілей (прибираються усі параметри цілей, якщо такі є, залишаються тільки назви цілей).

Повний код розробленого алгоритму доступний у GitHub репозиторії за посиланням [4].

5. Ілюстрація розробленого підходу

Для ілюстрації був розроблений невеликий iOS застосунок iNote для створення та редагування нотаток. Він має дві цілі, у кожної з них абсолютно різні ресурсні файли, тому зовнішній вигляд застосунків відрізняється. Файли, що містять код зі спільною логікою для обох застосунків належать до обох цілей, а зі специфічною – тільки до одної з них (рис.1). Файл `project.yaml` містить дані про конфігурацію проєкту (рис.2).

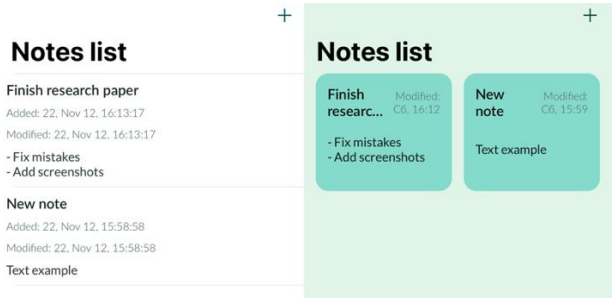

Рис. 1. Головний екран з нотатками кожного з застосунків

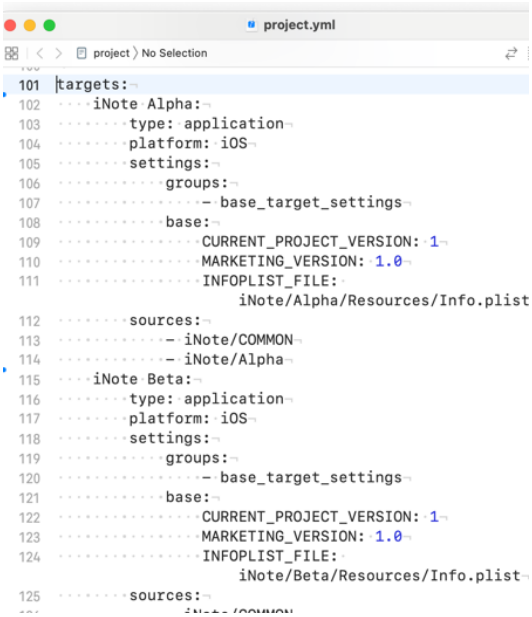

Рис. 2. Список цілей у файлі `project.yaml` для проєкту iNote

У разі необхідності створення третьої варіації застосунку, більше непотрібно вручну створювати ціль, налаштовувати її та розбиратися із файлами, і навіть непотрібно самостійно редагувати project.yaml. Достатньо створити конфігураційний файл цілей для

цього проєкту, наприклад `iNote.yaml` із таким змістом:

```
project: iNote
targets:
      Alpha:
      Beta:
      Gamma:
            steals:
                  -
iNote/Alpha/Resources/
                        -
iNote/Beta/Modules/
            inherit: Alpha
```

Запуск скрипта здійснюється командою `./igen iNote.yaml project.yaml`. У результаті виконання скрипта відбувається оновлення файлів `iNote.yaml`, `project.yaml`, копіювання ресурсних файлів із цілі Alpha, а вихідних файлів - із цілі Beta. Після запуску `xcode generate` у проєкт буде додано нову ціль, повністю готову для використання (рис.3).

Таким чином, використання скрипту igen та конфігураційного файлу дозволяє розробнику максимально швидко додавати нові цілі до існуючого проєкту.

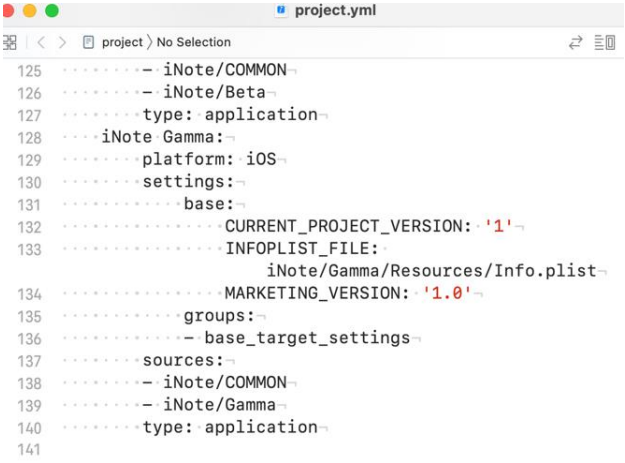

Рис. 3. Список цілей у файлі `project.yaml` після використання igen

Проєкт для ілюстрації розташований у GitHub репозиторії за посиланням [5].

Висновки

Було вдосконалено підхід із використанням декількох цілей в рамках одного Xcode проєкту шляхом використання XcodeGen для генерації проєктного файлу та самописної утиліти igen для модифікації конфігураційного файлу project.yaml.

Удосконалений підхід дозволяє максимально швидко створювати нові цілі на основі конфігураційного файлу, що дозволяє максимізує використання людських ресурсів та грошей, пришвидшує час розробки, а також мінімізує можливість виникнення людських помилок.

Був розроблений проєкт Xcode для ілюстрації описаного підходу і розміщено у відкритому доступі у GitHub репозиторії.

Список літератури

1. iOS : How to create two versions of your app with a single codebase [Online] – Available from: https://anutoshdatta.medium.com/ios-how-to-create-two-versions-of-your-app-with-a-single-codebase-eca0bb75b0fe, last accessed 2022/06/11.
2. 2 iOS Apps, 1 Codebase with XCode Scheme [Online] – Available from: https://testfairy.com/blog/2-ios-apps-1-codebase-with-xcode-scheme/, last accessed: 2022/06/11.
3. XcodeGen [Online] – Available from: https://github.com/yonaskolb/XcodeGen, last accessed: 2022/06/11.
4. igen [Online] – Available from: https://github.com/gloomikon/MultiTargetAutomatisation/tree/main/igen, last accessed: 2022/06/11.
5. MultiTargetAutomatisation Example Project [Online] – Available from: https://github.com/gloomikon/MultiTargetAutomatisation, last accessed: 2022/06/11.