УДК 621.9.048.7

ИСПОЛЬЗОВАНИЕ МОДУЛЬНОЙ АРХИТЕКТУРЫ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ ПОД УПРАВЛЕНИЕМ ОПЕРАЦИОННОЙ СИСТЕМЫ IOS

[1]Рудикова-Фронхёфер Л. В., [2]Романчук С. А., [3]Трус Ю. П.
[1]УО «Гродненский государственный университет имени Янки Купалы», Гродно, Беларусь, sergcom1998@gmail.com
[2]УО «Гродненский государственный университет имени Янки Купалы», Гродно, Беларусь, lada.rudikowa@gmail.com
[3]УО «Гродненский государственный университет имени Янки Купалы», Гродно, Беларусь, yurik851@gmail.com

В статье изложены общие принципы формирования и использования модульной архитектуры при разработке программного обеспечения для мобильных устройств под управлением операционной системы iOS. Демонстрируется пример непосредственного исследования, результаты которого базируются на описываемых принципах и подходах.

Разработка мобильных приложений на базе модульной архитектуры в iOS-проекте. Одно из последних в отрасли решений для оптимизации больших мобильных приложений – разделение проекта на внутренние модули. Модульная архитектура приложений решает некоторые серьезные проблемы при разработке мобильных приложений, например, следующие.

1. Максимальное переиспользование кода.

При разработке приложения в командах необходимо, чтобы каждая команда могла отвечать за конкретный модуль и были видны границы этой ответственности, было понятно, какими готовыми решениями можно пользоваться при разработке. Разработка внутри каждого модуля должна минимально влиять на остальные части проекта.

2. Поддержка слабой связности между модулями.

Когда разработка ведется в рамках одного проекта без разделения на модули, легко нарушать принцип инверсии зависимостей – с модулями это сделать намного сложнее.

3. Возможное переиспользование модуля в других приложениях.

Очевидно, например, что модуль UI-компонентов или модуль авторизации можно переиспользовать повторно в других приложениях клиента. То же самое можно делать и с расширениями.

4. Уменьшение времени сборки.

Сборка с пустым кэшем на старте составляет около трёх минут. А вот компиляция из кэша составляет около полторы минуты. Чем быстрее, тем лучше.

5. Отследить, насколько может увеличиться или уменьшиться время запуска приложения.

Если разделение на модули выполнено через динамические фреймворки, то неизбежно растет время запуска. Важно, чтобы это время оставалось в разумных пределах и в случае чего можно было отследить «проблемные» модули.

Разделение на модули. Благодаря использованию модулей можно устанавливать прямую зависимость с конкретным модулем, а не со всем приложением.

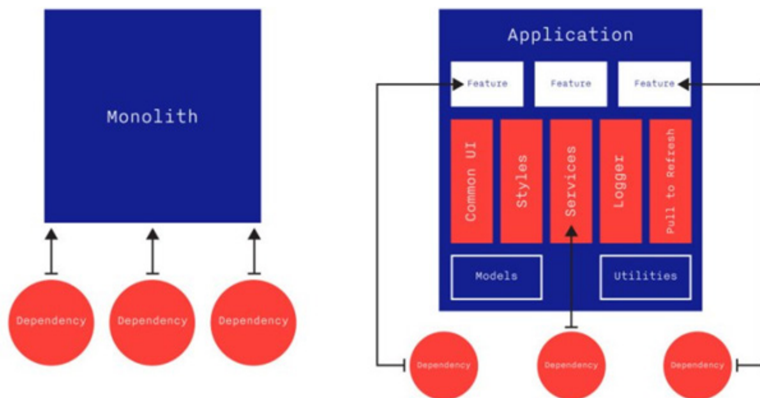

Рисунок 1 – Разница монолитной и модульной архитектур

Рекомендуется начинать с простых вещей, таких как стили (шрифты, цвета), других библиотек, которые есть в проекте. Например, папка с исходным кодом в приложении (загрузчики, логгеры, аналитика). После этого извлекаются модели. Далее необходимо извлечь код, который имеет дело с базой данных, общими элементами пользовательского интерфейса и функциональными модулями (рисунок 1).

Вид модульной архитектуры. Граф зависимостей гораздо понятнее. чем в монолитной архитектуре. Получается, что целое приложение делится на функции, которые должны использовать протоколы для связи между модулями (рисунок 2).

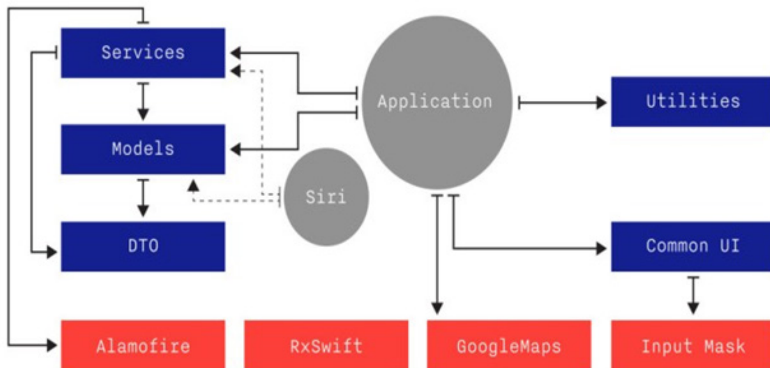

Рисунок 2 – Модульная архитектура приложения

Структура модульного проекта. Каждый отдельный реализуемый функционал будет представлять собой совокупность кода для решения задачи и интегрироваться в проект как фреймворк (рис. 3).

Распределение обязанностей в проекте. Работа с модулями очень удобна в команде. Каждый член команды может отвечать за свой отдельный модуль или использовать уже готовый модуль, который был ранее реализован в другом проекте, просто изменяя его свойства.

Работа строится следующим образом.

1. Разработчик, который реализовывает модуль, должен быть владельцем репозитория.

2. Разработчики, которые снова и снова используют (reuse) модуль в своих проектах, могут лишь создавать свои ветки и работать с модулем отдельно от мастера.

Удобство в том, что всегда есть человек, который может провести проверку кода, ведь он ведает про ценность, которую приносит этот код.

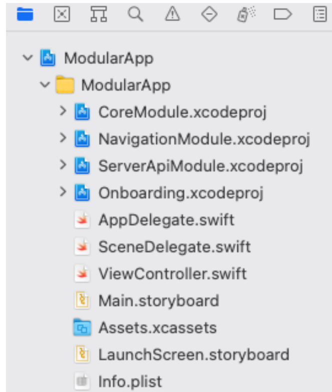

Рисунок 3 – Структура модульного проекта

Создание модуля.

1. Необходимо создать проект с шаблона «framework» (рисунок 4).

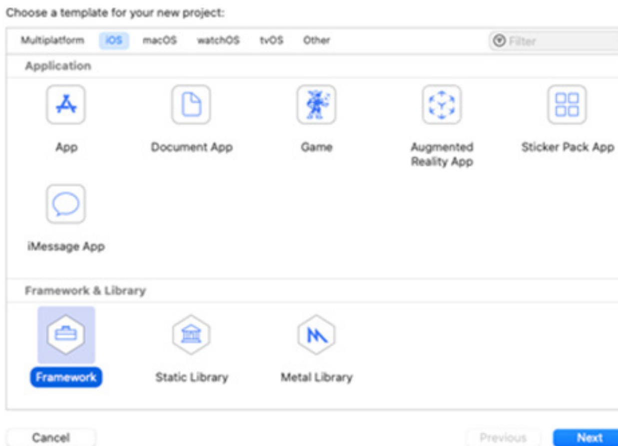

Рисунок 4 – Шаблоны проектов

2. Далее следует развернуть репозиторий, в котором будет находится наш проект. Расставить правильные уровни доступа для нашей команды.

1. Приступить к разработке кода.

2. Для оценки и тестирования кода понадобиться создать тестовый проект, в который будет добавлен framework и в дальнейшем будет доведен до желаемого уровня.

3. После того, как модули будут заполнены кодом, необходимо описать весь функционал в «README.md» файле.

При разработке модуля следует помнить, что реализация может быть сложной или простой внутри, но использование модуля должно быть простым для понимания. Один из плюсов модульности в том, что внутри мы можем использовать любую архитектуру, любые посторонние библиотеки и это не сломает основной проект.

При инициализации проекта уже известны какие модули будут использоваться, знаем как они будут между собой общаться, поэтому контрольной точкой является именно место склейки модуля, а не его внутренность.

Интеграция модуля в проект может проходить посредством стандартного набора: Swift Package Manager, Manual Framework, Carthage, Cocoapods.

При выборе провайдера необходимо учитывать некоторые факторы, а именно:

1. Carthage и Cocoapods — для публичного кода.

2. Manual Framework и SPM позволяют использовать внутренний код;

3. Swift Package Manager не работает с объектами @IBDesignable, @IBInspectable.

Недостатки использования модульной архитектуры.

1. Время на обучение. При переходе к модульному программированию придется набраться терпения и приучить себя описывать код и документацию. Вам нужно развить видение зависимостей, научиться определять и разделять функциональность на модули.

2. Чтение документации. Разработчик в команде не должен отвлекать разработчика модуля. Модуль считается мертвым, если нет документов.

3. Постоянная актуализация. Также необходимо поддерживать модули в актуальном состоянии как в основном проекте, так и в репозитории модулей. Поскольку каждый модуль представляет собой отдельный фреймворк, необходимо определить путь интеграции для SPM, Cocoapods и других.

Положительные аспекты использования модульной архитектуры.

1. Совместимость. Совместимость с различными архитектурами. Кроме того, разработчики время от времени совершают «ротацию». Зоны ответственности на монолитах не явны.

2. Стандартизация кода, привычка писать и читать код правильно. Швы выносятся в отдельные модули и решались проблемы с функциональными задачами изолированно.

3. Оптимизация. Сокращение количества постороннего кода и разграничение зоны ответственности. Каждый новый проект имеет уже готовый фундамент.

4. Скорость. Модули – это также итеративность. Проект создается на очертаниях взаимодействия модулей, а потом накапливаем код внутри каждого отдельного модуля. Итеративность помогает при разработке больших проектов с быстрым изменением вектора, модули взаимозаменяемы, а значит в любой момент мы можем изменить его бизнес ценность.

5. Облегчилась интеграция тестов, внедрение TDD.

Заключение. Хороший объектно-ориентированный дизайн всегда окупается. С такой архитектурой, особенно когда проект постепенно переносится в нее, преимущества хорошего объектно-ориентированного дизайна ощутимы. Переход модуля из монолитной конструкции в модульную может быть очень болезненным. Но если сделать работу правильно и создать слабосвязанные абстракции, вероятность того, что этот процесс будет затруднительным, будет меньше. Также необходимо помнить про инкапсулированную функциональность в открытом пространстве. Определяя четкую границу, нам нужно указать все функции, которые модуль будет использовать извне. Это позволяет записывать и видеть в одном пространстве все функции, которые вызывает модуль. Следствием этого является довольно четкая картина того, как настроен модуль.

СПИСОК ЛИТЕРАТУРЫ

1. Фаулер, М. Шаблоны корпоративных приложений / М. Фаулер. – 2014. – 544 с.

2 Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – 2021. – 352 с.