УПРАВЛЕНИЕ ЗАВИСИМОСТЯМИ ПРИ РАЗРАБОТКЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Полуэктов Н.А.

Институт информационных технологий Белорусского государственного университета информатики и радиоэлектроники
г. Минск, Республика Беларусь

Савенко А.Г. – старший преподаватель, м.т.н.

Аннотация. В данной работе рассмотрены различные менеджеры зависимостей, использующиеся при разработке мобильных приложений и их отличия между собой. Рассмотрено построение графа зависимостей и настройка версий пакетов.

Менеджеры зависимостей упрощают использование чужого кода программы, предоставляя этот код в виде независимых модулей – пакетов. Эти пакеты подключаются к проекту, и мы не знаем и нам не важно, как всё устроено внутри пакетов, но мы знаем, что они делают. Благодаря такой слабосвязанной архитектуре появляется возможность легко обновлять чужой код или заменять один пакет другим со схожей функциональностью.

У каждого менеджера зависимостей есть файл с настройками, в котором нужно указать от каких пакетов зависит приложение, чтобы менеджер их скачал и установил в систему. При этом каждый пакет может зависеть от других пакетов. Он распутывает эту систему зависимостей и устанавливает всё что необходимо для корректной работы.

Существуют следующие разновидности менеджеров зависимостей:

1. Системные менеджеры зависимостей – устанавливают недостающие утилиты в операционную систему.
2. Менеджеры зависимостей языка – собирают исходные файлы, написанные на одном из языков программирования, в конечные исполняемые программы.
3. Менеджеры зависимостей проекта – управляют зависимостями в разрезе конкретного проекта. То есть, в их задачи входит описание зависимостей, скачивание, обновление их исходного кода [1].

Общий алгоритм работы менеджера зависимостей можно представить следующим образом:

Шаг 1. Валидация проекта и среды окружения;

Шаг 2. Построение графа зависимостей;

Шаг 3. Скачивание пакетов;

Шаг 4. Интеграция зависимостей;

Шаг 5. Обновление зависимостей.

Пример файла настроек менеджера зависимостей представлен на рисунке 1.

Валидация включает проверку версий ОС, вспомогательных утилит, которые необходимы менеджеру зависимостей, а также линковку настроек проекта и manifest-файла: начиная от проверки на синтаксис, заканчивая несовместимыми настройками [2].

Возможны следующие предупреждения и ошибки при валидация проекта:

1. Не найдена зависимость.
2. Явно неуказана операционная система или версия.
3. Некорректное имя workspace или проекта.

```
target 'YFC-ios' do
  use_frameworks!

  # UI
  pod 'lottie-ios'
  pod 'SnapKit', '~> 5.0.0'
  pod 'CountryPickerSwift'
  pod 'PanModal'
  pod 'CropViewController'
  pod 'GoogleMaps', '6.1.1'

  # Storage
  pod 'KeychainAccess'

  # Network
  pod 'Moya', '~> 14.0'
  pod 'Alamofire', '~> 5.2'
  pod 'AlamofireNetworkActivityLogger'

end
```

Рисунок 1 - Пример файла настроек менеджера зависимостей Cocoapods для iOS приложения

Все зависимости должны образовывать направленный ацикличный граф. У нужных зависимостей могут быть свои зависимости, а у тех в свою очередь – собственные вложенные зависимости или подзависимости. Результатом построения графа является созданный lock-файл, который полностью описывает отношения между зависимостями. Пример такого графа представлен на рисунке 2.

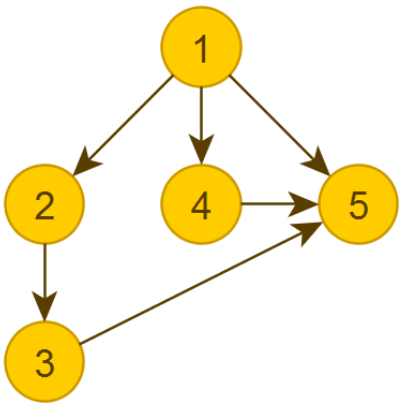

Рисунок 2 – Иллюстрация направленнего ацикличного графа

После успешного построения графа и создания lock-файла, менеджер зависимостей скачивает пакеты и интегрирует в проект таким образом, чтобы их беспрепятственно можно было использовать.

Контролировать исходный код зависимостей в проекте можно с помощью их версий.

В менеджерах зависимостей можно контролировать исходный код зависимостей с помощью:

1. Версии. Можно указать конкретную версию, так и интервал.
2. Ветка. При обновлении ветки мы не можем предсказать, какие изменения произойдут.
3. Коммит. Зависимость ссылается на конкретный коммит или тэг. Если его не изменят, то пакет никогда не будут обновляться.

В заключение можно сказать, что менеджеры зависимостей являются важным инструментом разработчика и без его использования было бы затруднительно добавлять новые модули или обновлять существующие, что приведет к значительному замедлению и повышению стоимости разработки программного продукта.

Список использованных источников:

1. Dependency Management in iOS [Электронный ресурс]. – Режим доступа: https://blog.devgenius.io/dependancy-management-for-ios-27dd681d7ea0, – Дата доступа: 04.02.2022.

2. Менеджеры зависимостей [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/redmadrobot/blog/412945/, – Дата доступа: 14.03.2022.