Slide1

Всем привет.
Меня зовут Александр Додатко. Я работаю в компании EPAM system.

Сейчас я расскажу вам о Continuous integration для iOS приложений.

Slide2

Речь пойдет о работе с проектами в xCode, сборке из командной строки (без запуска GUI), создании "Universal binary" для iOS, а также поставке версий для тестирования (deployment).

работ
Slide3

Также мы рассмотрим unit-testing с помощью GHUnit и настройку hudson build server.
Запуск приложений без запуска xCode будет полезен вашим тестировщикам (QA).

Slide4

Для начала давайте вспомним, каким образом ОБЫЧНО осуществляется повторное использование кода в iOS проектах.
На примере TouchXML мы видим что исходники просто копируются в основной проект.

Slide5

Так делать не следует. В особенности если код будет использоваться в нескольких проектах.

Slide6

Вместо копирования следует использовать статические библиотеки, выделяя их в отдельные *.xcodeproj
---

Slide7
Slide8

Для этого мы добавляем library sub-project.

Slide9

После чего следует добавить зависимости компоновщика и заголовков интерфейсов.
(linker and header dependencies)

Slide10

В своих проектах мы используем следующую структуру директорий.
  app - для продуктов, поставляемых в App Store
  lib, lib-third-party - для библиотек. Своих и сторонних соответственно.
  frameworks - для сторонних *.framework. Также в этот каталог выполняется deployment своих библиотек, оформленных как Universal Binary или framework
  deployment - здесь build server будет искать собранные версии продуктов.
  tests - для Unit test всех видов.
  demo - для пробных проектов-прототипов, не поставляемых в App Store.
  tools - для программ-утилит Mac OS X, используемых при сборке.

  [остальное и так ясно]

Slide11

<No Comments>

Slide12

Особенностью desktop applications является тот факт что сборка и исполнение производится на одной и той же системе.
Поэтому организовать Continuous Integration сравнительно просто. Для таких приложений существует множество CI tools и литературы.

Slide13

Для iOS ситуация немного другая. Программы исполняются либо на реальном устройстве с iOS, либо на симуляторе.
Ситуация дополнительно усложняется системой provisioning profiles, которую навязывает apple.

Посему автоматический запуск программ, сбор результатов тестов (*.xml JUnit reports) будет несколько труднее чем "написать имя программы в shell script" и "открыть файл на чтение".

Slide14

Если с этим всем не разобраться, то вашим QA прийдется доставать исходные коды из системы контроля версий (SVN, GIT, mercurial), КОМПИЛИРОВАТЬ их, настраивать у себя provisioning profiles.
Наши QA одно время этим занимались.

Это неправильно. CI script должен собирать *.ipa файл, который может быть установлен на устройство с помощью iTunes. Хотя xCode organizer удобней с моей точки зрения (согласитесь, organizer -- это гораздо проще и быстрее чем компиляция).

Slide15

Итак, приступим к сборке. Данный слайд иллюстрирует взаимосвязь между xCode command line interface и GUI.
Screenshot был взят из xCode3. Надеюсь, соответствие с xCode4 GUI вы сможете найти самостоятельно. (к моему сожалению, эту информацию разбросали по разным частям IDE)

Если нет вопросов по этому слайду - давайте продолжим.

Slide16

После успешной сборки проекта соберем *.ipa файл для наших любимых QA и клиентов.
Для этого нам нужно 3 вещи :
  * собранный *.app для iOS устройства
  * DeveloperName -- строка с информацией о developer profile. Пример -- в нижней части слайда.
  * Provisioning Profile -- это файл. Да-да. Тот самый файл который вы импортировали в свой Organizer.

Слушаю вопросы по данному слайду.

Slide17

Если больше вопросов не предвидится, то давайте поговорим о Unit тестировании.

Slide18

Выбор unit test framework очень важен. Мы рассмотрели трех основных "игроков" для ObjectiveC.
SenTestingKit, GoogleToolbox, GHUnit.

Мне немного непонятен подход Apple к данному вопросу. Он всем хорош кроме 2х вещей :
1. Отсутствие Debug. (с помощью некоторых ухищрений этого можно добиться, пожертвовав корректной работой Assert)
2. Отстутствие работы с Bundles. (для нас это важно, так как наши приложения активно взаимодействуют с web services)

Google toolbox пытается быть совместимым с ним. Поэтому страдает от тех же проблем.

GHUnit же избрал иной путь. Unit tests оформляются в виде отдельного iOS application. При желании его можно даже отправить в App Store ;)
Таким образом, эти тесты избавлены от описанных недостатков.
Однако не все так радужно. Их гораздо труднее использовать в CI из-за отсутствия интеграции с xCode и их "Application origin".

Slide19

Но и это еще не все. Для использования в рамках CI приложение нужно правильно сконфигурировать.
Эта конфигурация отличается от интерактивной, используемой при development.

На слайде показано как сделать необходимые вещи :
* запускать без повеления пользователя.
* получить результаты теста в формате *.xml
* завершиться после выполнения (и не мешать дальнейшей работе сценария сборки)

** запустить проект и показать действие флагов.

Slide20

Здесь показана команда запуска теста. Данная утилита не входит в состав SDK. Однако вы с легкостью сможете найти ее на github и собрать самостоятельно.
Ее интерфейс до безобразия прост.
Однако следует быть осторожным с передачей пути к приложению. Программа отказывается работать с относительными путями и выдает не вполне внятные сообщения об ошибках.

Slide21

Сбор результатов тестов также достаточно прост. GHUnit записывает их во временную папку.
Вот в эту (** показать **).
Теперь осталось только переместиться в нее и скопировать тесты в нужную папку.

Slide22

Еще один неочевидный момент -- необходимость "убить" процесс симулятора. Если этого не делать, то следующий тест может вообще не запуститься (это bug/feature apple SDK).
Я советую делать это перед и после запуска очередного UnitTestXXX.app
---

Slide23

Вот и дошла очередь до Universal binaries. А что это?
Это специальным образом собранная библиотека, которая содержит символы как для device, так и для simulator.
Пользоваться ей так же просто как и обычными static libraries в вашей любимой Mac OS X/Linux/Windows.
При этом не отдавая ваш драгоценный код кому попало (** о техниках защиты от дизассемблирования речь идти не будет **).

А вот собрать ее куда сложнее.

Slide24

Для этого нужно собрать отдельные версии для device и simulator. А затем специальным образом объединить их.
Вот она - эта "уличная магия".

Slide25

Теперь немного псевдокода нашего сценария сборки.
Полный код можно найти на github.

Slide26

А вот что делает наш build server.
Думаю, эти шаги подойдут и вам.

Slide27

Спасибо за внимание.
Надеюсь, никто не успел заснуть.
