Slide1

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

Сейчас я расскажу вам о нашем опыте Continuous integration для iOS приложений.

Slide2

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

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
Изначально build servers были нацелены на сборку Java и .NET проектов. Поддержка сборок для C и C++ выполнена гораздо хуже.
Однакоко каждый из них может и исполняет свои основные функции :
  * получение исходных кодов из SVN
  * сборка проекта. (для c, c++ это запуск скрипта или makefile)
  * публикация результатов сборки (продуктов и unit tests)

Slide5

Теперь разберемся, что же должен делать build script.
В нашем случае он
  * собирает основной проект
  * собирает и запускает unit tests
  * подготавливает собранные продукты и test reports к обработке с помощью build server.
  * выкладывает сборку в директорию с общим доступом. (наша команда использует dropbox. В зависимости от своих нужд вы можете использовать ftp или другие средства)

Slide6

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.

Slide7

Так делать не следует. Надеюсь, всем понятно почему.
В особенности если код будет использоваться в нескольких проектах.

Slide8

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

Slide9
Slide10

Для этого мы добавляем library sub-project.

Slide11

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

Slide12

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.
Она используется для предоставления библиотеки сторонним разработчикам, не открывая своего драгоценного кода.
В сегодняшнем докладе о техниках защиты от дизассемблирования речь идти не будет.

В MacOS для этих целей также используют frameworks и SDK. Однако Universal binary является их важной составляющей.

Использовать Universal binary так же просто, как и обычные static libraries в вашей любимой Mac OS X/Linux/Windows.
А вот собрать ее куда сложнее.

Slide13

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

Slide14

Теперь поговорим о процедуре deployment под iOS.
Процедура "выкладывания в App Store" хорошо изучена и описана в документации Apple.
Поэтому мы поговорим о поставке версий для тестирования.

Slide15

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, описывающей процесс сборки.

Slide16

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".

Slide17

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.
*.ipa файл содержит в себе приложение, собранное для iPhone, iPod или iPad. А также provisioning profiles, необходимые для установки.
В принципе, profiles и application можно устанавливать на устройства отдельно. Однако с моей точки зрения, *.ipa файлы более удобны.

Slide18

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, эти настройки разбросали по разным частям новой IDE).

Slide19

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.

Slide20

Следующая тема -- модульное тестирование. Мы рассмотрим
  * вопрос выбора testing framework.
  * технику автоматического запуска тестов на iOS simulator.
  * получение результатов тестирования по завершении работы iOS simulator.

Slide21

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

SenTestingKit разрабатывается в Apple и поставляется вместе с SDK.
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)
Если честно, мне немного непонятен подход Apple к данному вопросу.

Google toolbox пытается быть совместимым с ним. Поэтому страдает от тех же проблем.
Хотя он содержит некоторые полезные функции. Такие как возможность сравнения screenshots приложения на предмет соответствия спецификации (при условии что она у вас есть).

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".

Учитывая данную сравнительную таблицу, мы выбрали для себя GHUnit. Поскольку мы не готовы жертвовать возможностями отладки тестов и работы с bundles.

Slide22

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 :
  * GHUNIT_AUTORUN -- запускать без повеления пользователя.
  * WRITE_JUNIT_XML -- получить результаты теста в формате *.xml
  * GHUNIT_AUTOEXIT -- завершиться после выполнения (и не мешать дальнейшей работе сценария сборки)

(** запустить проект и показать действие флагов. **)

Slide23

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.

Slide24

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.

Slide25

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
---

Slide26

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]

Slide27

Спасибо за внимание.
Рабочий пример проекта и сценариев сборки можно найти на нашей странице github. Равно как и эту статью.
Те, кто не уснул, могут задать свои вопросы.
