Introduction (Apps and the Android platform)

Spyros Samothrakis
Research Fellow, IADS
Univerisity of Essex

January 11, 2016

Course Structure

- 10 weeks
- Each week:
  - 2-hour lecture (including group discussion and software demos)
  - 2-hour lab: practice writing and debugging apps
- Assessment:
  - 2 assignments
    - App prototype (20%, wk 19)
    - Final app (70%, wk 25)
- 1 progress test (10%, wk 20)
  - Multi-choice test under exam conditions

Mobile and Social Application Programming

- Course focus: the software design and implementation of mobile applications
- Exciting platforms to develop on
- Many facilities:
  - Powerful processors, reasonable memory
  - Hi-Res touch-screen
  - Connected: Internet (3G, WiFi), Bluetooth, Telephony, SMS, Near Field?, 4G?
  - GPS, location services, maps
  - Access to multi media play and capture
  - Motion sensors

Wide Range of Apps (1)

- Games
  - Casual e.g., reaction games, card games, board games, Tetris, physics-based
  - Arcade e.g., Asteroids
  - 3D Console Style e.g. Grand Theft Auto
  - Social e.g. Quiz/ QuizUp
- Social
  - Facebook, Twitter

Wide Range of Apps (2)

- Sports
  - e.g. trackers like Endomondo, MapMyRide
- Productivity
  - Email, note taking, shopping
- Information (flights, weather, traffic,. . .)
- Transactional (e.g. Shopping: Amazon, eBay)
- Health, Education

Why Android?

- Open platform
- Large market share:
  - Diverse range of devices (some beautiful!)
  - Extensive monetization possibilities
  - Play store and other markets
- Powerful mobile operating system
  - Worth studying in its own right
- Good support tools and easy deployment
- Main language: Java
  - Well known, great IDEs (Intellij, Eclipse), easy to learn and use
- For more on Market Share see:
  - http://www.theguardian.com/technology/2014/jan/09/market-share-smartphones-iphone-android-windows

There are alternatives to Java!

- This course is java-centric
- Not always the case, android development is done in other platforms as well
- From Python (Kivy) to Unity, there are alternatives to Java
- Java is however considered the default android language

Android App Development

- Take an idea through to implementation and publication
  - Idea -> Draft Requirements
  - Requirements may change opportunistically
  - Underlying logic / model
  - File or Network I/O
  - Sensors
  - GUI Design and event handling
    - Glue logic: ensuring all components talk to each properly
  - Testing, Debugging, Redesign, Testing, Debugging, ...
  - Design of Launcher Icon

Note on Opportunistic Development

- For your assignment, and for App development in general I recommend taking an agile development approach
- Start with a rough spec, implement a prototype, then redesign as necessary
- Don't bother trying to get all the details fully specified before implementing anything

Commercial Vs Research

- Let's have a look at possible projects for the course
- Research Project?
  - Different style, different audience
- Commercial focus?
  - What makes a good up sell?

Use a Good IDE (e.g., IntelliJ or Eclipse)

- Auto-generate and check project structure
- Refactoring support
  - Change method names
  - Move methods between classes
  - Pull methods up from classes to interfaces
- Auto-check lots of tedious errors
- Navigate from usage to definition and vice versa
- Auto-generate UML Class Diagrams
  - Useful for high-level view
  - And inclusion in reports
- Drag and Drop GUI Designer

Mining the play store

- Discussion Question
  - As an app developer, what useful market research data is freely available from the Play Store PRIOR to publishing an app?
- And a follow-up:
  - What data is available after publishing?

From Java to Android

- Suppose you are a competent or even expert Java programmer
  - What more knowledge / skills do you need to become and Android Developer?
- App lifecycles
- Android API (e.g. the GUI classes are completely different)
  - Fortunately the many standard Java packages are all included
- XML Descriptor Files
  - Can design GUI using layout editor (which constructs XML), XML editor (text view), or write directly in Java

Good Android apps need to be well engineered

- Some standard ways of doing things
- And some important restrictions you need to learn
  - Seemingly innocent actions such as updating a view with the wrong thread can cause an app to stop
- Architecture such as Model View Controller (MVC)
  - Encapsulation of state (good practice anyway, but essential for easy restoration after a restart)
  - Attention to lifecycle
  - Bundling data
  - Activities, Intents, Fragments
  - Highly modular

Learning and Discovery

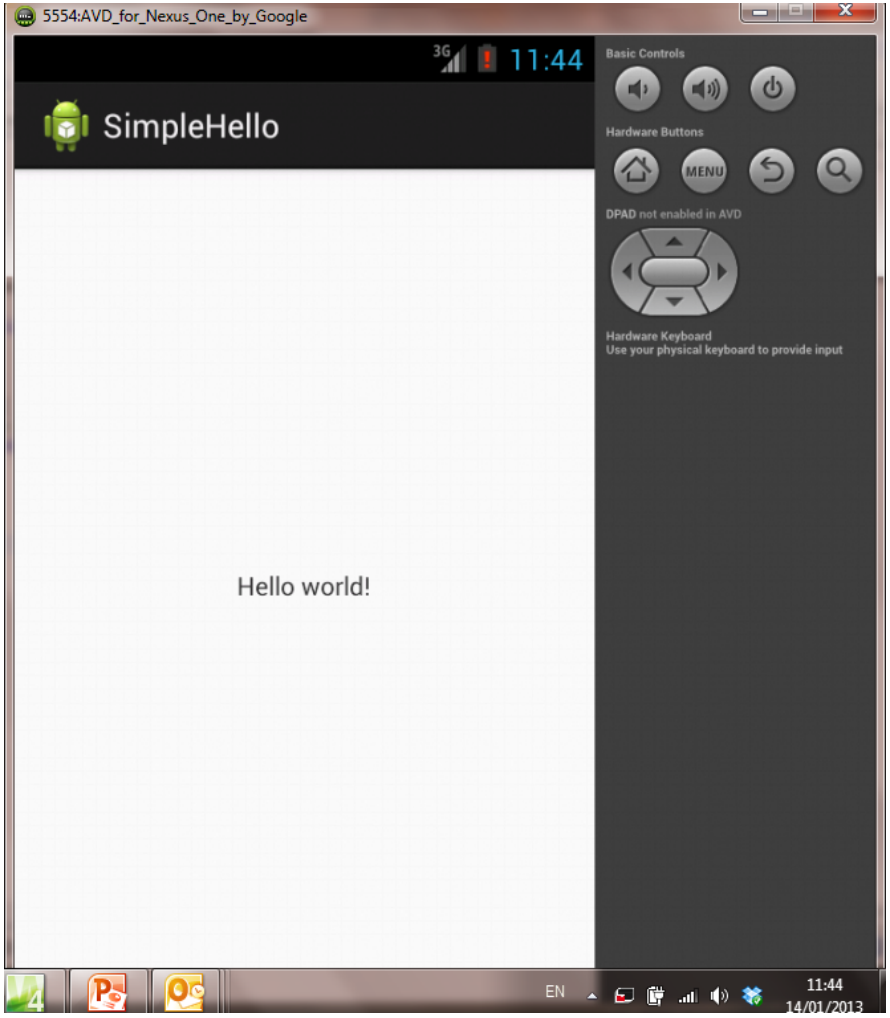
- This is a taught project-style course
- Lectures and labs will cover a good deal of useful material
- BUT: the Android platform is extensive, we won't cover it all
- You will need to discover / research many aspects for yourself
- Ask me and each other
- StackOverflow, developer.android.com and other resources
- Just Googling for a problem often finds the solution

When things go wrong

- Use IDE to find static edit / compile time errors
- For Runtime errors learn to use the Logcat
- All System.out is directed there
- Use Tags to filter most relevant messages
- Learn to use debugger
  - DDMS (Dalvik Debug Monitor Server)
  - Find problems with running code
  - (Dalvik is the name for the Android Java Virtual Machine)
- Google for solutions to other problems (e.g. deployment errors)

Hello World

- This is just one possible first app
- The one that gets auto-created by Intellij or Eclipse when selecting a Blank Activity
- (each IDE may have minor differences in the default HelloWorld app)



Hello World Code

```java
package com.example.simplehello;

import android.os.Bundle;

public class SimpleHelloActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple_hello);
    }
}
```

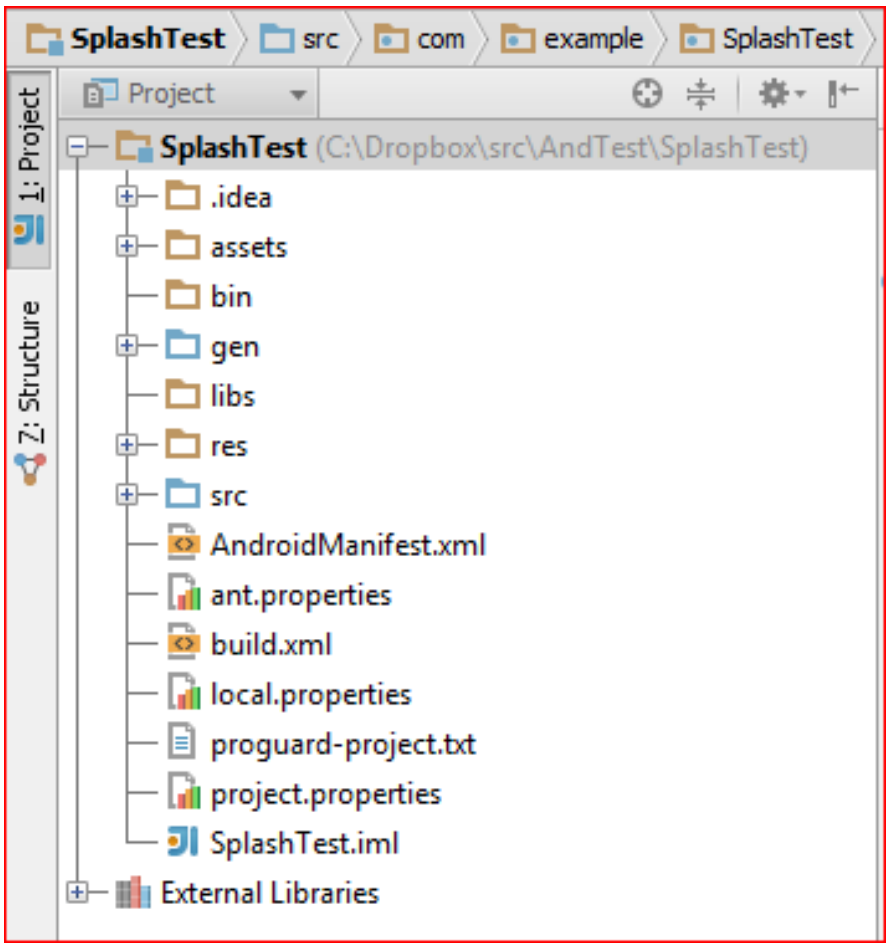
Notes on Hello World

- Extends Activity: this is the most common class to sub-class when making an app
- onCreate is the method called when the app is first launched
- Bundle is the set of data passed to onCreate that allows an App to re-create the previous state where the user left off
- Well behaved Apps normally do something to explicitly manage state
- Either using the Bundle, or by storing data in a file
- The file-based approach gives longer persistence

Anatomy of an Android App

http://developer.android.com/tools/projects/index.html

- assets: files you provide at compile-time for your app
- bin: the final .apk file for deployment on Google Play gets built here
- gen: auto-generated resources go here
  - generated from the XML files in the res folder



Anatomy Continued

- libs
  - Put library .jar files (e.g. we'll be using gson.jar to save and load data with minimal effort)
- res
  - XML files go here that specify GUI features of the project including the arrangement of component views
- src
  - Java files go here
- They should be properly package qualified
- e.g., for a developer account:
  - com.ssamotapps. . . . (important when publishing on Play)

Android Manifest File

SplashScreen.java | StartPage.html | AndroidManifest.xml | main.xml | FirstScreen.java | firstscreen.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.SplashTest"
          android:versionCode="1"
          android:versionName="1.0">
    <uses-sdk android:minSdkVersion="17"/>
    <application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
        <activity android:name="SplashScreen"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name="FirstScreen"
                  android:label="@string/app_name">
        </activity>
    </application>
</manifest>
```

Important Aspects of the Manifest File

- The manifest file is auto-created by the IDE but may involve you specifying some options
- You can also edit these by hand
  - `<uses-sdk android:minSdkVersion="17"/>`
- Choose one as low as you can that supports all the features you need
- The application attributes specify the app name and the app icon
- Note the use of the '@' to refer to resources declared elsewhere:
- ```
  <application android:label="@string/app_name"
               android:icon="@drawable/ic_launcher">
  ```
  ```

The Res Folder

- Four drawable folders containing different resolution versions of the same icon
- A layout folder with an XML file for each activity
- A values file containing a strings.xml file to define commonly used string values
- Note: res folders can contain more than this

An application contains at least one Activity

- The one identified by the MAIN intent is the one called when the App is launched (e.g. by clicking the icon on a device screen)
- The main activity may then launch other activities
- Only one main activity can be defined per application
- But Activities may respond to other Intents

Exploring Manifest Entries

- Tip!
- Use navigation within an IDE to find where things are defined
- E.g.

```
<activity android:name="FirstScreen"
          android:label="@string/first_label">
</activity>
```

- In Intellij using -b will with the cursor in “FirstScreen” will take you to the FirstScreen.java file where the class FirstScreen is defined
- This also works for Strings and other definitions

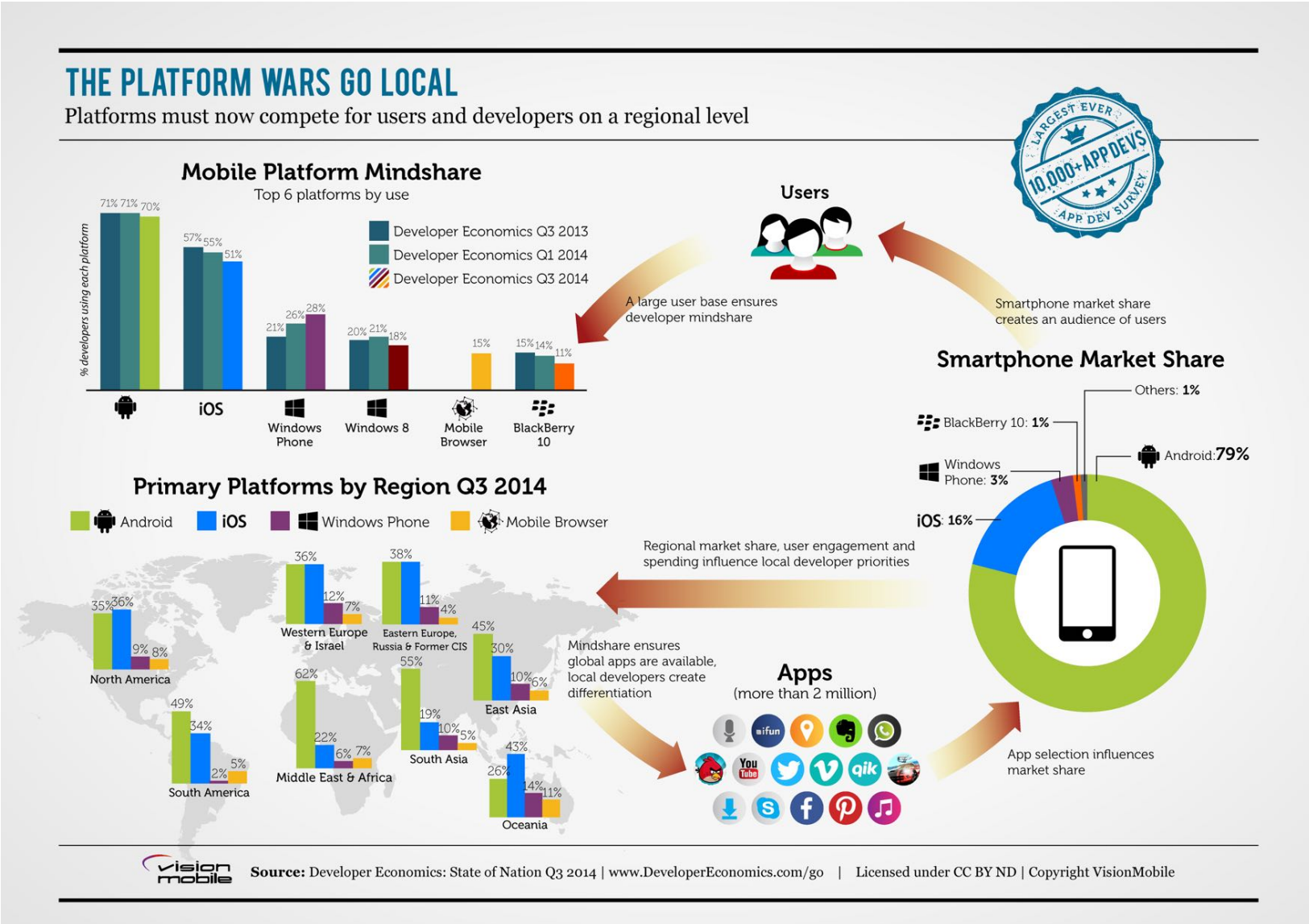
Can you fully explain this line?

- setContentView(R.layout.activity_simple_hello);

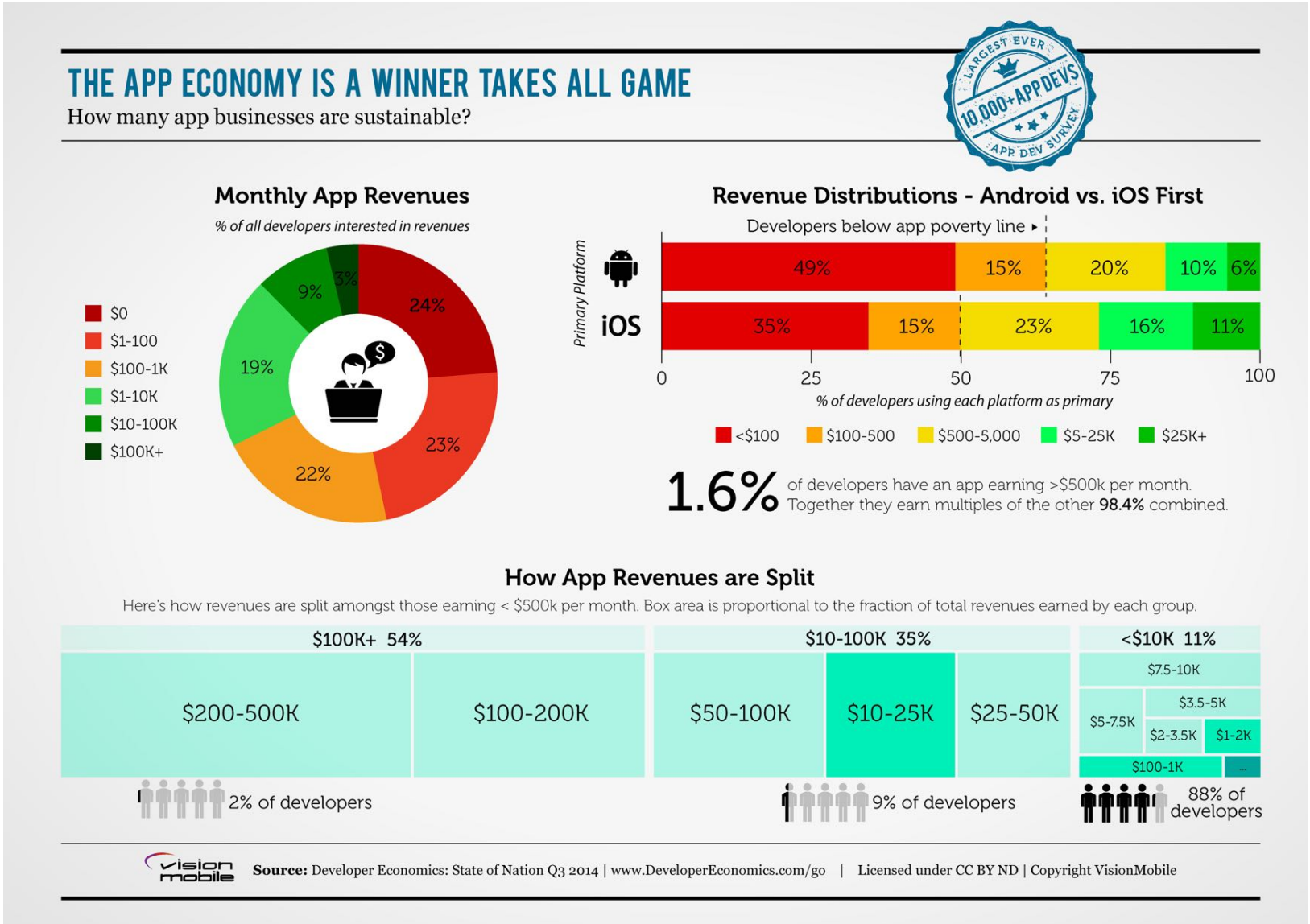
Statistics

- Let's assume you finish the course
- What are your chances of earning money in the wild west?
- Developer Economics, State of the Developer Nation Q3 2014 www.developereconomics.com/go
- Survey on 10K developers

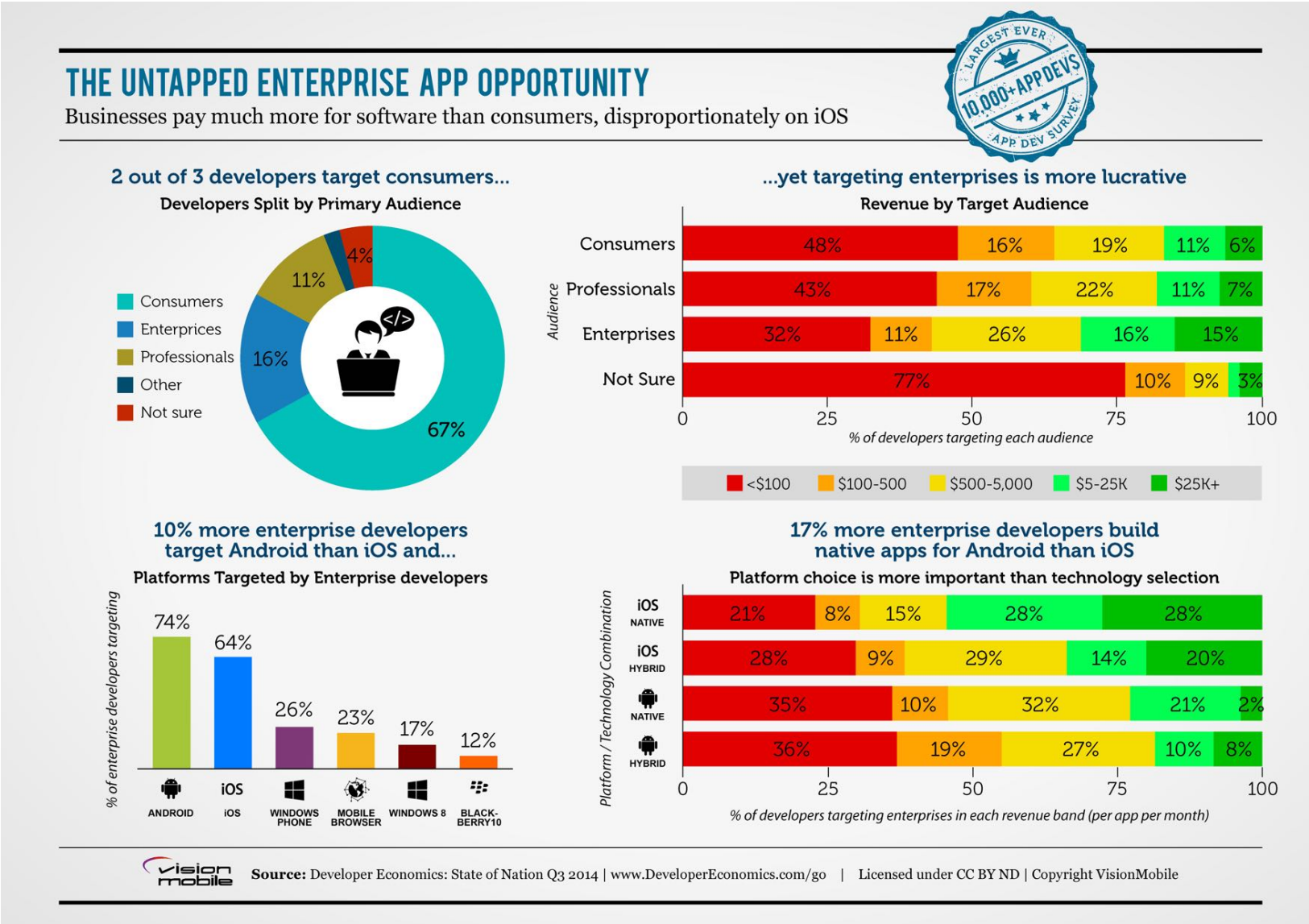
More Statistics



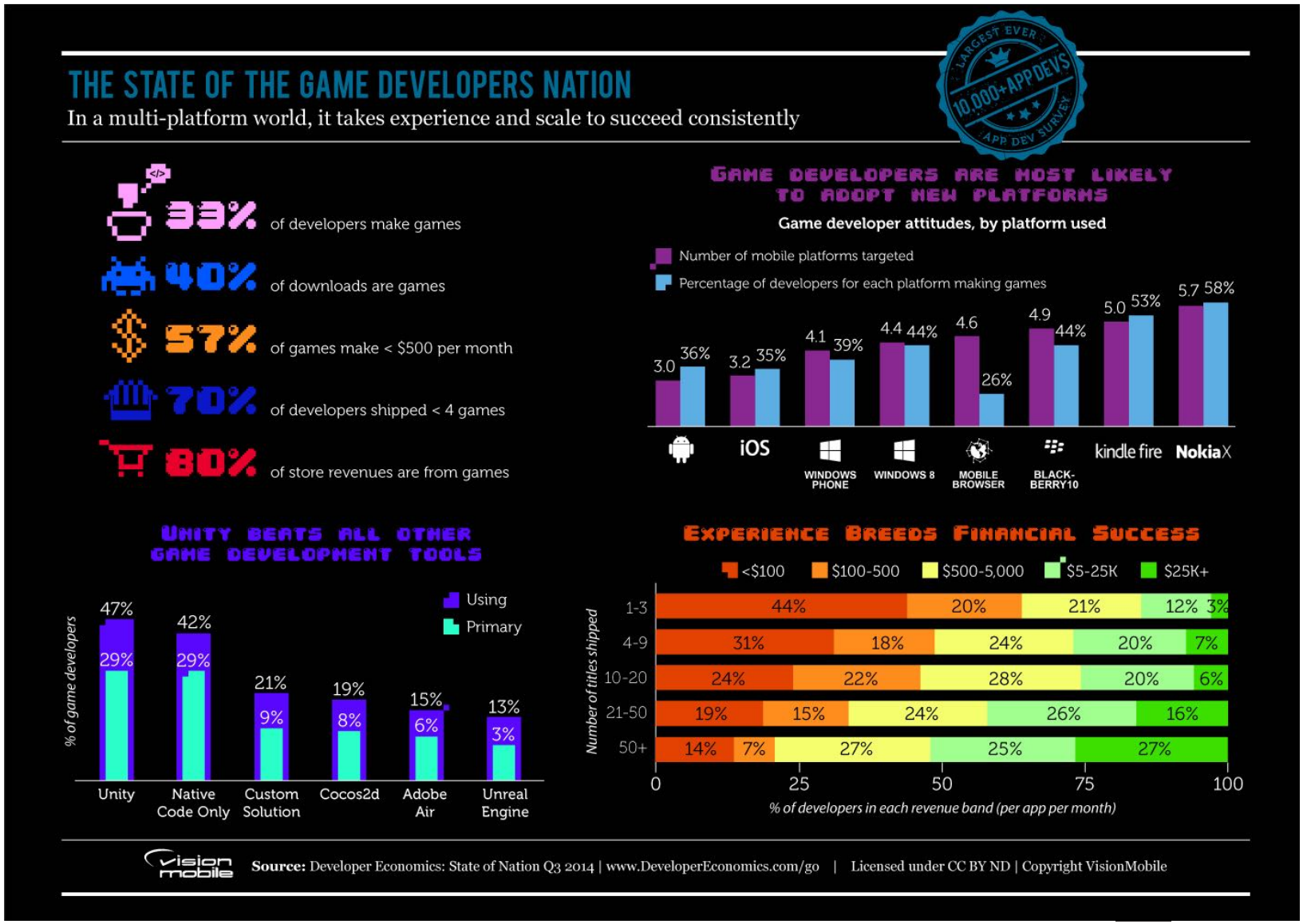
App Economy


THE APP ECONOMY IS A WINNER TAKES ALL GAME
How many app businesses are sustainable?
LARGEST EVER
10,000+ APP DEVS
APP DEV SURVEY
Monthly App Revenues
% of all developers interested in revenues
$0
$1-100
$100-1K
$1-10K
$10-100K
$100K+
24%
23%
22%
19%
9%
3%
Revenue Distributions - Android vs. iOS First
Developers below app poverty line
Primary Platform
iOS
49%
15%
20%
10%
6%
35%
15%
23%
16%
11%
0
25
50
75
100
% of developers using each platform as primary
<$100
$100-500
$500-5,000
$5-25K
$25K+
1.6% of developers have an app earning >$500k per month.
Together they earn multiples of the other 98.4% combined.
How App Revenues are Split
Here's how revenues are split amongst those earning < $500k per month. Box area is proportional to the fraction of total revenues earned by each group.
$100K+ 54%
$200-500K
$100-200K
$10-100K 35%
$50-100K
$10-25K
$25-50K
<$10K 11%
$7.5-10K
$5-7.5K
$3.5-5K
$2-3.5K
$1-2K
$100-1K
2% of developers
9% of developers
88% of developers
vision mobile
Source: Developer Economics: State of Nation Q3 2014 | www.DeveloperEconomics.com/go | Licensed under CC BY ND | Copyright VisionMobile

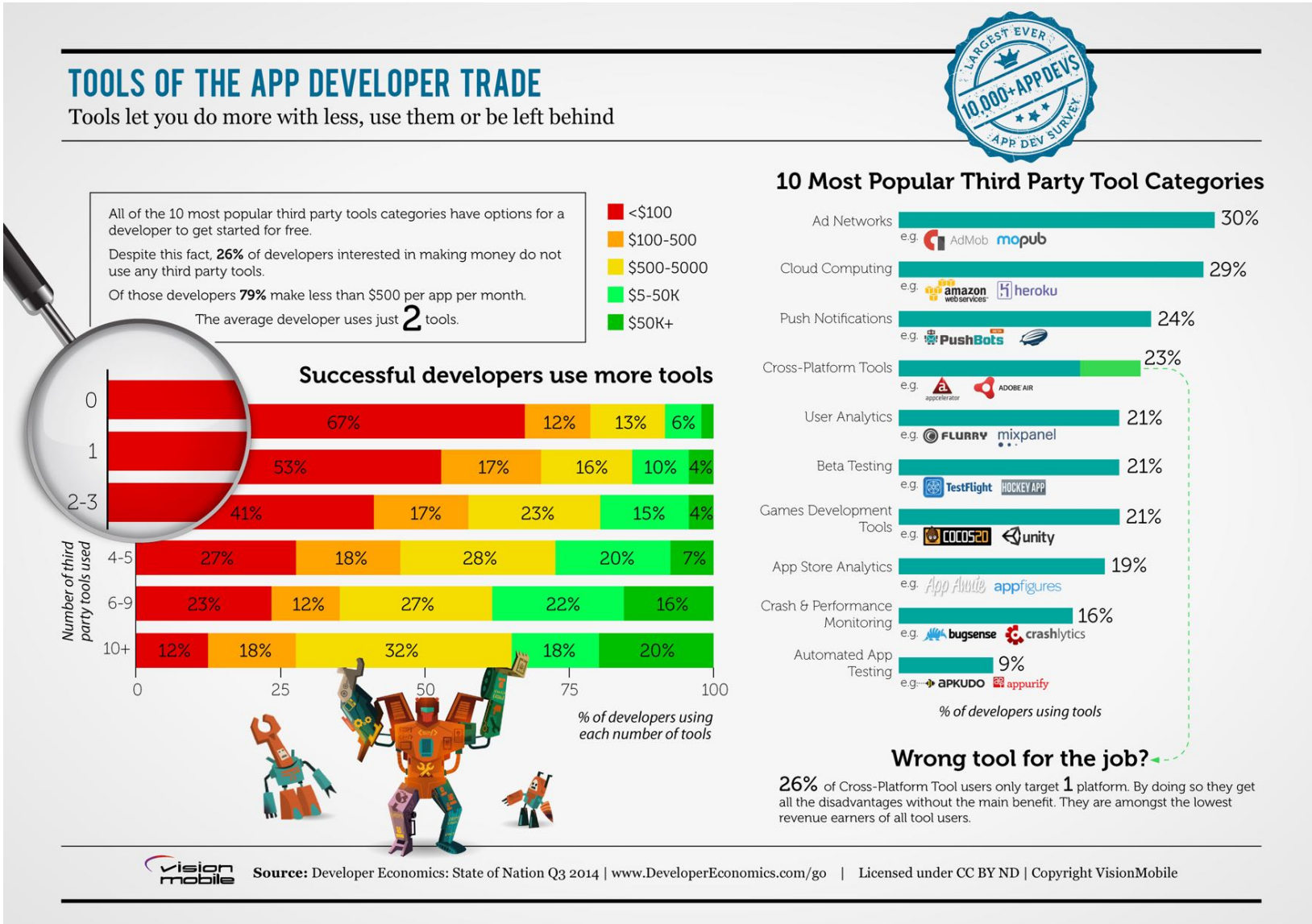
Enterprise Apps



Games


THE STATE OF THE GAME DEVELOPERS NATION
In a multi-platform world, it takes experience and scale to succeed consistently
LARGEST EVER
10,000+ APP DEVS
APP DEV SURVEY
33% of developers make games
40% of downloads are games
57% of games make < $500 per month
70% of developers shipped < 4 games
80% of store revenues are from games
GAME DEVELOPERS ARE MOST LIKELY TO ADOPT NEW PLATFORMS
Game developer attitudes, by platform used
Number of mobile platforms targeted
Percentage of developers for each platform making games
3.0 36%
3.2 35%
4.1 39%
4.4 44%
4.6 26%
4.9 44%
5.0 53%
5.7 58%
iOS
WINDOWS PHONE
WINDOWS 8
MOBILE BROWSER
BLACK-BERRY10
kindle fire
NokiaX
UNITY BEATS ALL OTHER GAME DEVELOPMENT TOOLS
% of game developers
Using
Primary
47% 29%
42% 29%
21% 9%
19% 8%
15% 6%
13% 3%
Unity
Native Code Only
Custom Solution
Cocos2d
Adobe Air
Unreal Engine
EXPERIENCE BREEDS FINANCIAL SUCCESS
<$100
$100-500
$500-5,000
$5-25K
$25K+
Number of titles shipped
1-3 44% 20% 21% 12% 3%
4-9 31% 18% 24% 20% 7%
10-20 24% 22% 28% 20% 6%
21-50 19% 15% 24% 26% 16%
50+ 14% 7% 27% 25% 27%
0
25
50
75
100
% of developers in each revenue band (per app per month)
vision mobile
Source: Developer Economics: State of Nation Q3 2014 | www.DeveloperEconomics.com/go | Licensed under CC BY ND | Copyright VisionMobile

Tools


TOOLS OF THE APP DEVELOPER TRADE
Tools let you do more with less, use them or be left behind
LARGEST EVER
10,000+ APP DEVS
APP DEV SURVEY
All of the 10 most popular third party tools categories have options for a developer to get started for free.
Despite this fact, 26% of developers interested in making money do not use any third party tools.
Of those developers 79% make less than $500 per app per month.
The average developer uses just 2 tools.
<$100
$100-500
$500-5000
$5-50K
$50K+
Successful developers use more tools
Number of third party tools used
0
1
2-3
4-5
6-9
10+
67% 12% 13% 6%
53% 17% 16% 10% 4%
41% 17% 23% 15% 4%
27% 18% 28% 20% 7%
23% 12% 27% 22% 16%
12% 18% 32% 18% 20%
0
25
50
75
100
% of developers using each number of tools
10 Most Popular Third Party Tool Categories
Ad Networks 30%
e.g. AdMob mopub
Cloud Computing 29%
e.g. amazon web services heroku
Push Notifications 24%
e.g. PushBots
Cross-Platform Tools 23%
e.g. appcelerator ADOBE AIR
User Analytics 21%
e.g. FLURRY mixpanel
Beta Testing 21%
e.g. TestFlight HOCKEY APP
Games Development Tools 21%
e.g. COCOS2D unity
App Store Analytics 19%
e.g. App Annie appfigures
Crash & Performance Monitoring 16%
e.g. bugsense crashlytics
Automated App Testing 9%
e.g. APKUDO appurify
% of developers using tools
Wrong tool for the job?
26% of Cross-Platform Tool users only target 1 platform. By doing so they get all the disadvantages without the main benefit. They are amongst the lowest revenue earners of all tool users.
vision mobile
Source: Developer Economics: State of Nation Q3 2014 | www.DeveloperEconomics.com/go | Licensed under CC BY ND | Copyright VisionMobile

Summary

- Android is a rich and powerful platform, with many opportunities for developing and profiting from apps
- Give careful thought to the app you want to develop for this course
- IDEs such as Intellij and Eclipse take a lot of the tedium out of the development process
  - But Android is complex, and there is much to learn
- Massive audience, you still stand a chance to make it big

Recommended reading

Programming Android: Java Programming for the New Generation of Mobile Devices, By Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura, Publisher: O'Reilly Media, July 2011
Android Programming: The Big Nerd Ranch Guide (2nd Edition), by Bill Phillips, Chris Stewart, Brian Hardy and Kristin Marsicano

Credits

- Course outline/structure was based on Simon's Lucas 2014 Course