Engineering of Mobile Systems

Romain Robbes
rrobbes@unibz.it

Outline

- Course presentation
- Why Mobile?
- Why React Native?
- The React Native Paradigm

Course Presentation

Course Format

- Lectures
  - Tuesday 08:00 - 10:00
  - Thursday 08:00 - 10:00
- Labs
  - Tuesday 14:00 - 16:00
- Office hours
  - By email appointment (rrobbes@unibz.it)

In this course, you will:

- Learn the key concepts of mobile application development and the internet of things.
- Gain practical experience using state of the art technologies to develop mobile applications.
- Acquire expertise in writing mobile applications that use advanced mobile APIs and connect to outside web services
- Be aware of the various tradeoffs in the development of mobile applications.

Prerequisites

- Basic web and programming background
- The basic principles of Software engineering
- Being curious to learn and get your hand dirty

Main Topics

- Javascript (the sane parts), Typescript, and Functional Programming
- React, React Native, and Expo
- Essential React Native and Expo components
- More advanced APIs such as access to device sensors
- Background activity and connecting to web services
- State Management and persistence
- Limited ressources: energy, performance, memory
- Testing, debugging, and deploying

Version 2

- This is version 2 of the course
- If you attended last year, there are changes:
  - Typescript instead of Javascript
  - Functional components instead of class components
  - More emphasis on Expo
- This is to **simplify** the course overall

Lectures and Labs

- Lectures:
  - Presenting concepts
  - Extensive examples
- Labs:
  - Practice of concepts
  - Q&A, help on projects

Evaluation

- Written exam: 50%
- Project: 50%

Teaching material

- Git repo: **https://github.com/rrobbes/EngineeringOfMobileSystemsV2**
- OLE: https://ole.unibz.it/course/view.php?id=8613
- React: https://reactjs.org
- Native: https://reactnative.dev
- Expo: https://expo.io
- References will be added to the course web site

The Git Repo

- This is the **last time** I'll be using slides!
- Git repo: **https://github.com/rrobbes/EngineeringOfMobileSystemsV2**
- The Git repo contains all the notes for the class, links to additional resources and code examples
- To download a copy: `git clone https://github.com/rrobbes/EngineeringOfMobileSystemsV2.git`
- To update the notes, go inside the directory, then do: `git pull`
- You can also submit corrections via "pull requests"

Assignments

- Only practical experience ensures you master the concepts seen in class
- Two assignments will be handed out
  - Javascript and Typescript
  - React Native basics
- They will **not** be graded, but can contribute “bonus points” for the exam
- They will be a source of **feedback**

Project

- A final graded project will be handed. It will be larger than the assignments
- Start early! Work regularly to avoid last-minute issues
- You can’t pass the class without approving the projects

Contacting me

- rrobbes@unibz.it
- (Piazza Domenicani 3 - Office 1.16)
  (First floor to the right)

Why Mobile?

What does 2009 represent for Mobiles?

- For the first time …
- more people accessed the Internet from a mobile phone rather than a PC

Mobile devices are **everywhere**

- (Still) Growing popularity
- Used for day to day activities
- Affordable
- Convenient Size

Lots of hardware possibilities

- High-definition touch screens
- Camera(s)
- Media player
- Internet connectivity
- GPS
- Accelerometers, barometers
- Augmented reality, on-device machine learning ….

Constraints in Mobile

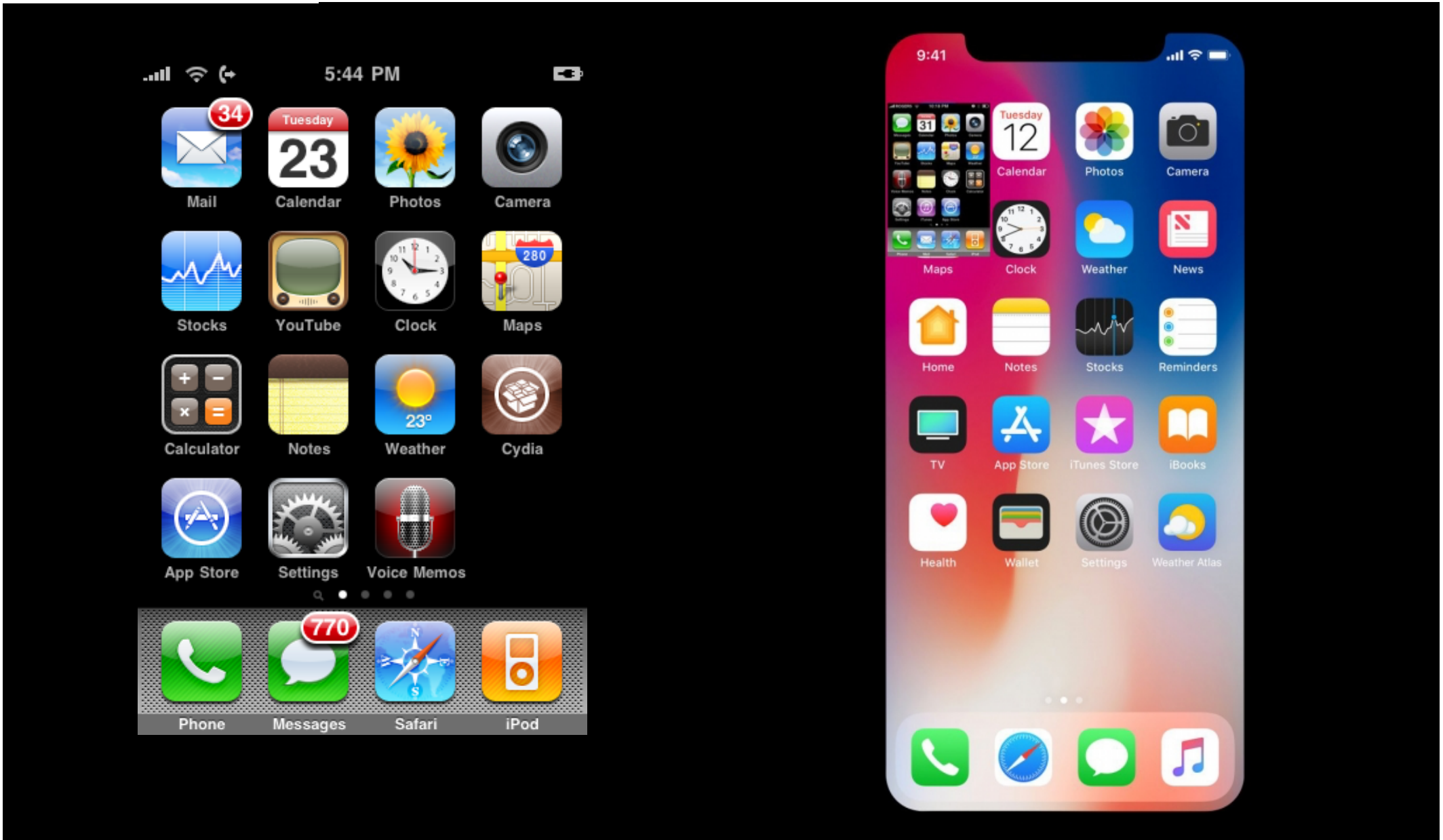
Mobile hardware imposes constraints

- Low processing power
- Limited RAM
- Limited permanent storage capacity
- Small screens with low resolution
- High costs associated with data transfer
- Intermittent connectivity, slow data transfer rates, and high latency
- Unreliable data connections
- Limited battery life

But things change

- Low processing power
- Limited RAM
- Limited permanent storage capacity
- Small screens with low resolution
- High costs associated with data transfer
- Intermittent connectivity, slow data transfer rates, and high latency
- Unreliable data connections
- Limited battery life

10 years later



Things change?

- Low processing power
- Limited RAM
- Limited permanent storage capacity
- Small screens with low resolution
- ….

Limitations lead to design considerations

- Be efficient (speed, battery, etc)
- Expect limited capacity
- Design for different screens
- Expect low network speed and high latency
- Keep costs in mind

Why React Native?

Historically

- Mobile development required
  - Coding in low-level C or C++
  - Understand the specific hardware (typically a single device or possibly a range of devices from a single manufacturer)
- Platforms such as Symbian were created to provide a wider target audience

Then came symbian

- Symbian (1998-2014) provided developers with a wider target audience
- Hundreds of millions of smartphones running Symbian were sold

However

- They required developers to write complex C/C++
- Use of proprietary APIs led to fragmentation
- Limited access to hardware capabilities
- Give different weight to native a third party apps

Java ME (Micro Edition)

- Introduced the Mobile Information Device Profile (MIDP) specification
- Java MIDlets follow this specification
  - abstracting the underlying hardware
  - create applications that run on the wide variety of devices that support the Java run time
- However
  - restricted access to the device hardware
  - sandboxed execution
  - For very limited devices (160-512KB memory), e.g. IoT nowadays

A Modern mobile OS is designed for powerful hardware

- Android, iOS, (and Windows Mobile)
  - Provide a rich and simplified development environment for mobile applications
  - Hardware access is available to all applications
- But …

These are different platforms

- Developing for Android or iOS is a lot of work
- Developing for both is … more work!
- Teaching both in 60 hours is … unwise

Cross-platform solutions

- There are several solutions for cross-platform development
  - PhoneGap
  - Ionic
  - Xamarin
  - React Native
  - Flutter

The problem with cross-platform solutions

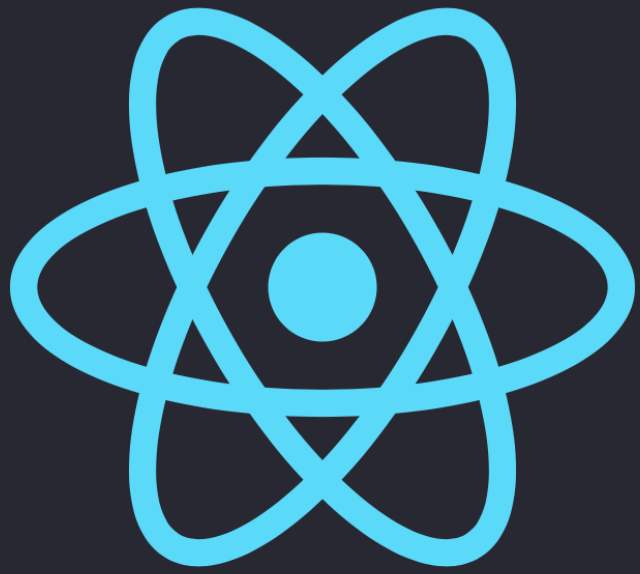
- Most cross-platform solution are “non-native”
- They feel “out of place” for users, even if they can’t really say why
- They may not have access to all functionality

React Native

Learn once, write anywhere.

Get started

Learn basics ›



Why React Native?

- React Native is cross-platform, BUT uses native components
- It can be mixed with fully native views if needed (e.g., to use platform-specific native functionality)
- Its programming paradigm promotes good practices (functional programming)
- It also has very good tools, including “live programming” support

In addition

- The React UI paradigm (which RN uses) is not limited to React only.

A web counter in React

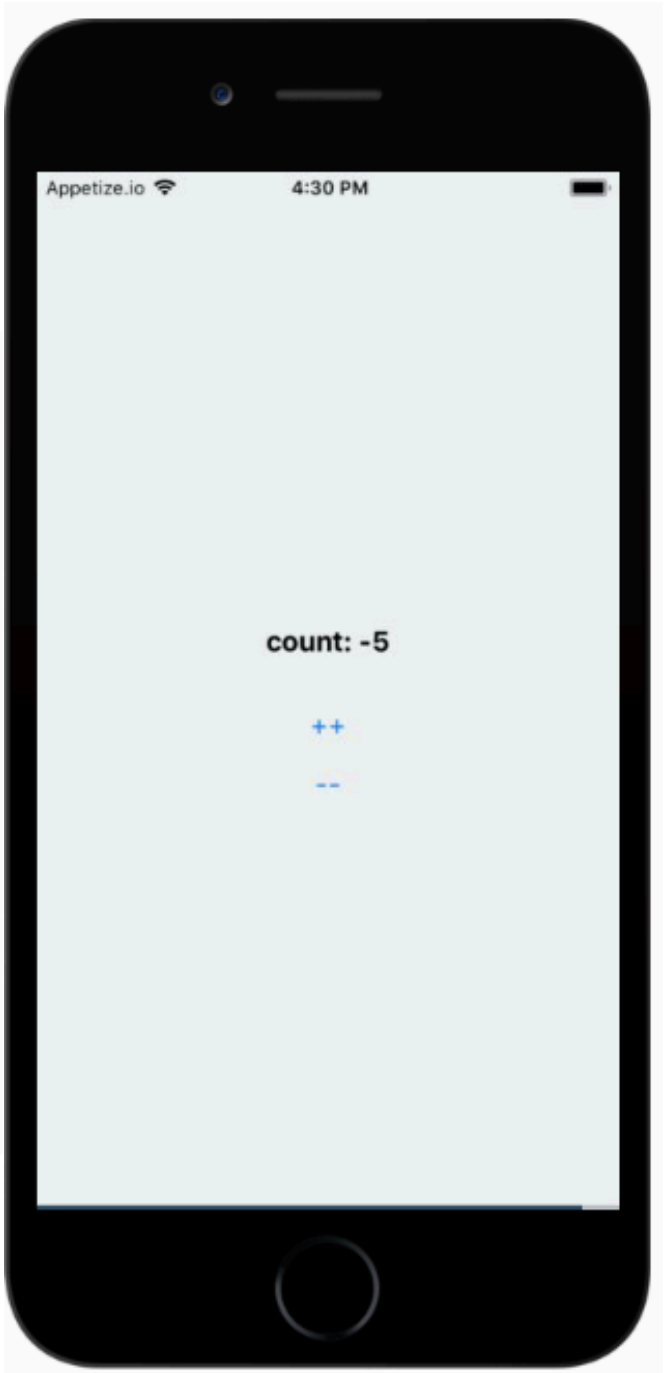
```
const Counter = () => {
    const [count, setCount] = useState(0)
    const increment = () => setCount(count + 1)
    const decrement = () => setCount(count - 1)

    return (
      <div>
        <h2>count: {count}</h2>
        <button onClick={increment}>++</button>
        <button onClick={decrement}>--</button>
      </div>
    )
}
```

RN counter

```
const Counter = () => {
    const [count, setCount] = useState(0)
    const increment = () => setCount(count + 1)
    const decrement = () => setCount(count - 1)

    return (
      <View>
        <Text>count: {count}</Text>
        <Button title="++" onPress={increment} />
        <Button title="--" onPress={decrement} />
      </View>
    )
}
```



Jetpack Compose (Android)

```
@Composable
fun AppBody() {
    val count = +state { 0 }

    Center {
        Column {
            Text("count: ${count.value}")
            Button("Increment", onClick = { count.value++ })
            Button("Decrement", onClick = { count.value-- })
        }
    }
}
```

SwiftUI (iOS)

```
1  struct AppBody : View {
2      @State var count = 0
3
4      var body: some View {
5          VStack {
6              Text("count: \(count)")
7              Button("Increment") { self.count += 1 }
8              Button("Decrement") { self.count -= 1 }
9          }
10     }
11 }
```

https://microsoft.github.io/react-native-windows/
https://github.com/Microsoft/react-native-macos
https://github.com/necolas/react-native-web

The React Paradigm

(This is a very quick tour—we'll see all of this in much more details over several weeks)

Programming is hard!

[…] our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed.

[We should do] our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program […] and the process […] as trivial as possible. —Edsger W. Dijkstra

Programming is hard!

Data changing over time is the **root of all evil.**

Functional Programming is “easy”

Functional Programming is the process of building software by composing **pure functions.**

A **pure function** is a function, which, given the same inputs, always returns the same outputs, and has **no side effects**.

Side effects may include: mutation, relying on shared state, I/O, asynchronous code, etc.

To make our job easier, we should limit side effects.

Characteristics of FP

Pure functions are **predictable** and **reproducible**

Separation: try to make one function do one thing only

Composition: write functions whose outputs are the input of others

Immutability: mutating data erases its previous values

Memoization: predictable functions are easier to cache

Parallelism: predictable functions are easy to parallelize

Imperative vs declarative

Declarative code describes what it does

```
1  function ReactComponent({counter}) {
2    return <span>{counter}</span>
3  }
```

Imperative code describes how it does it

```
1  function UpdateCounter({counter}) {
2    document.getElementById('counter').innerHTML(
3      `<span>${counter}<span>`
4    );
5  }
```

When executed, all code is imperative, but we can hide that

React is …

- Composable
- Reusable
- Scalable

} Components, Props, State

- Declarative

} Rendering, JSX

React: the general idea

- Declare a tree of UI components
- Render it
- Process events via callbacks
- If the data changes, re-render the tree (smartly)

Components

- React components are trees of sub-components
- Components can be **pure functions**

```
const Greeting = () => {
    return (
      <View>
        <Text>
            Hello world!
        </Text>
      </View>
      )
}
```

Components are composable

```
Const App = () => {
    return (
      <View>
         <Header/>
         <Greeting/>
         <Footer/>
     </View>
       );
}
```

JSX

- Is syntactic sugar to declare components easily

```
const element = (
    <Text style={{fontFamily: “bold”}}>
          Hello world!
    </Text>
);
```

```
const element = React.createElement(
    "Text",
    {style: {fontFamily: “bold”}},
    "Hello world!”
);
```

Embedding JS in JSX

- Just put code inside curly braces `{this.way()}`

```
const name = "Romain";

const Greeting = () => {
    return (
        <div>
            <span>
                Hello {name}!
            </span>
        </div>
    );
}
```

Mixing UI and logic?

- Traditional approaches use templates
- Templates separate technologies, not concerns
- React tries to keep each component self-contained
- The component has high cohesion
- And low coupling with other components

JSX gotchas

- Always return one node from render (correctly nesting sub-nodes)
- May need self-closing `<tags/>`
- Keepings track of `()'s, {}'s …`

Props

- Short for “properties”
- Read-only values received from parent components
- Unidirectional data-flow from top to bottom
- Like parameters for a function
- Changes to props will re-render the component

Passing props

```
const Greet = (props) => {
    return (
        <Text>
            Hello {props.name}!
        </Text>
      );
}

const App = () => {
    return <Greet name="you"/>
}
```

State

- Values that may change over time (yuck!)
- Components can have state
- Changes to state will re-render component

```
const Counter = () => {
    const [count, setCount] = useState(0)
    // …
```

Changing the state

- Via the setX function (in this case setCount)

```
const Counter = () => {
    const [count, setCount] = useState(0)

    const increment = () => setCount(count + 1)
    const decrement = () => setCount(count - 1)

    // …
```

Changing state

- **Do not directly mutate the state**
- Use setX (eg setCount) to change it
- Set the state to a clean copy of the state with the changes
- E.g., do not insert an element in a list, return a new list with the element added
- Javascript's spread operators (see next week) can help with that

Callbacks

- Callbacks are functions handling interaction
- JS functions are objects that can be passed around, stored in variables, and called later
- They are attached to event handlers, usually as props
- They are called in response to user interaction, when a task finishes, etc
- If they cause changes the UI will re-renders

Using callbacks with buttons

```
const Counter = () => {
    const [count, setCount] = useState(0)

    const increment = () => setCount(count + 1)
    const decrement = () => setCount(count - 1)

    return (
      <View>
        <Text>count is: {count}</Text>
        <Button title="++" onPress={increment}/>
        <Button title="--" onPress={decrement}/>
      </View>
      )}
```

Re-rendering the tree …

- React re-renders on every change
- Like old websites
- You don’t have to worry about it

Re-rendering on every changes simplifies everything

- All places where data is shown are up-to-date
- No magical data binding
- No model dirty checking
- No explicit tree manipulations

Wait, isn’t this slow?

- React optimizes it
- It uses a tree of virtual UI components
- Compares the tree to the old one
- Computes the minimal set of changes
- And imperatively executes the changes on the real tree
- So, it’s fast! :-)

Putting it all together

- Let’s code!

To Recap

- React makes it easy to declare and compose components
- Components are cohesive, including their UI
- Components can be pure functions, or have state
- React manages a tree of components
- When they change, they are re-rendered
- React makes it fast

Why develop for android?

- Powerful, and open SDK
- No licensing fees (25$ to access Google Play Store)
- Excellent documentation
- A thriving developer community

It is easy to get started

- The barrier to entry for new Android developers is minimal:
  - No certification is required to become an Android developer.
  - There is no approval process for application distribution.
  - Developers have total control over their brands.

Android is very open

- Third party and native applications use the same APIs and the same runtime
- Users can replace any application by a 3rd party (even the home screen)

And of course, there's market share …

What is Android?

"*The first truly **open** and comprehensive platform for mobile devices. It includes an operating system, user-interface and applications — all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation.*" — Andy Rubin

Android is an ecosystem with three components

- A free, open-source operating system for embedded devices
- An open-source development platform for creating applications
- Devices, particularly mobile phones, that run the Android operating system and the applications created for it

More specifically

- A **Compatibility Definition Document** (CDD) and Compatibility **Test Suite** (CTS) that describe the capabilities required for a device to support the software stack.
- A **Linux kernel** that provides a low-level interface with the hardware, memory management, and process control, all optimized for mobile and embedded devices.
- A **run time** used to execute and host Android applications, including the Dalvik Virtual Machine (VM) and the **core libraries** that provide Android-specific functionality.
- **Open-source libraries** for application development, including SQLite, WebKit, OpenGL, and a media manager.

More specifically

- An **application framework** that exposes system services to the application layer, including the window manager and location manager, databases, telephony, and sensors.
- A **user interface framework** used to host and launch applications.
- A **software development kit** (SDK) used to create applications, including the related tools, plug-ins, and documentation.
- A set of core pre-installed **applications**.

Core applications include:

- An e-mail client
- An SMS management application
- A full PIM (personal information management) suite, including a calendar and contacts list
- A WebKit-based web browser
- A music player and picture gallery
- A camera and video recording application
- A calculator
- A home screen
- An alarm clock

Android devices may also ship with Google apps

- The Google Play Store for downloading third-party Android applications
- Google Maps, including StreetView, driving directions, and turn-by-turn navigation
- The Gmail email client
- The Google Talk instant-messaging client
- The YouTube video player

More apps

- Different vendors customise their own android phones
  - user interface
  - apps
  - services
- Plus 3,000,000+ apps in the Google Play Store (and other Android stores)

Android SDK features

- GSM, EDGE, 3G, 4G, and LTE networks for telephony or data transfer
- APIs for location-based services such as GPS and network-based location detection
- Full support for applications that integrate map controls as part of their user interfaces
- Wi-Fi hardware access and peer-to-peer connections
- Access to Hardware, Including Camera, GPS, and Sensors

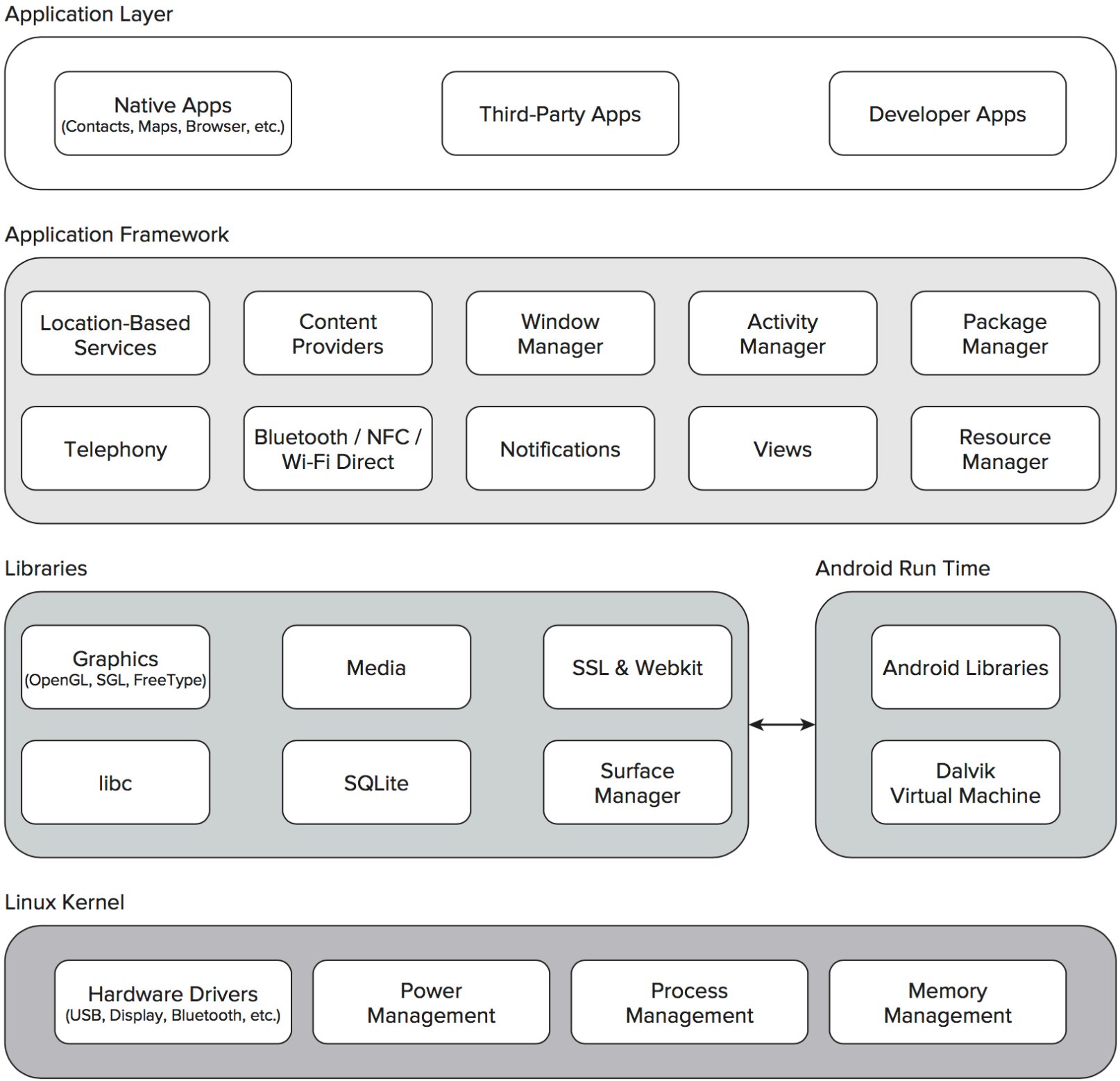
Android SDK features

- Multimedia hardware control, including playback and recording with the camera and microphone
- Background Services, applications, and processes
- An integrated open-source HTML5 WebKit-based browser
- Mobile-optimized, hardware-accelerated graphics, including a path-based 2D graphics library and support for 3D graphics using OpenGL ES 2.0
- Reuse of application components and the replacement of native applications
- Transparent access to different hardware devices

What Comes in the Box

- The Android APIs and full documentation
- Sample code
- Development tools
- The Android Virtual Device Manager and emulator
- Online support:
  - http://developer.android.com/support.html
  - www.stackoverflow.com/questions/tagged/android

Android Software Stack



Android's Application Architecture encourages reuse

- publish and share Activities, services, and data with other applications
- reuse native apps inside your app
- expose your components to other applications

The cornerstones of all Android applications

- **Activity Manager** and **Fragment Manager**: Control the lifecycle of your **Activities** and **Fragments**
- **Views**: construct the user interfaces for your Activities and Fragments
- **Notification Manager**: a consistent, nonintrusive notification mechanism
- **Content Providers**: share data with other applications
- **Resource Manager**: non-code resources management
- **Intents**: transferring data between applications and their components
- **Broadcast receivers**: be informed of system events

Types of Android Applications

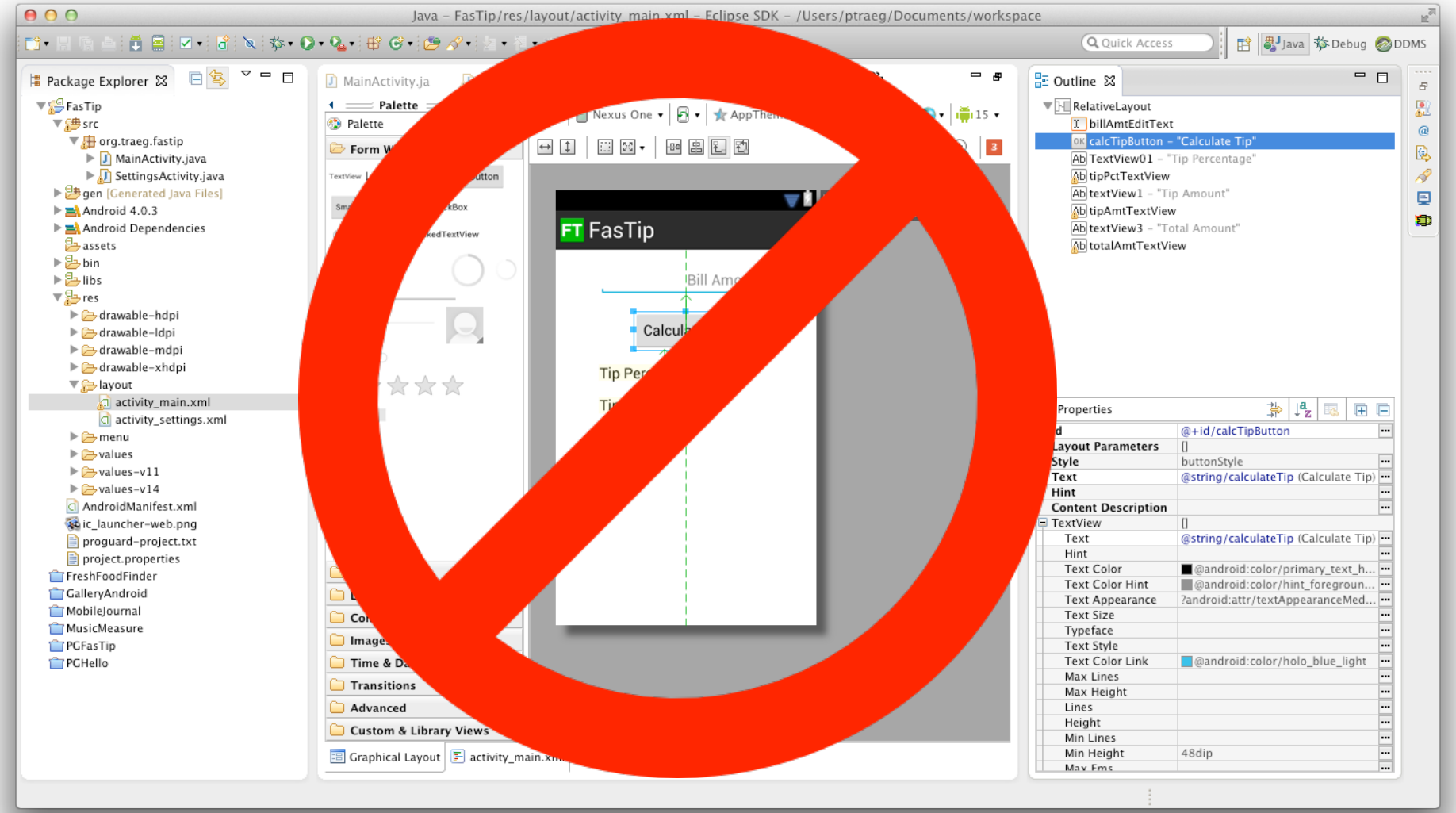
- **Foreground**: An application that is useful only when in the foreground and suspended when not visible. E.g., a game, a web browser.

- **Background**: An application with limited interaction that, apart from when being configured, spends most of its lifetime hidden, such as an alarm clock.

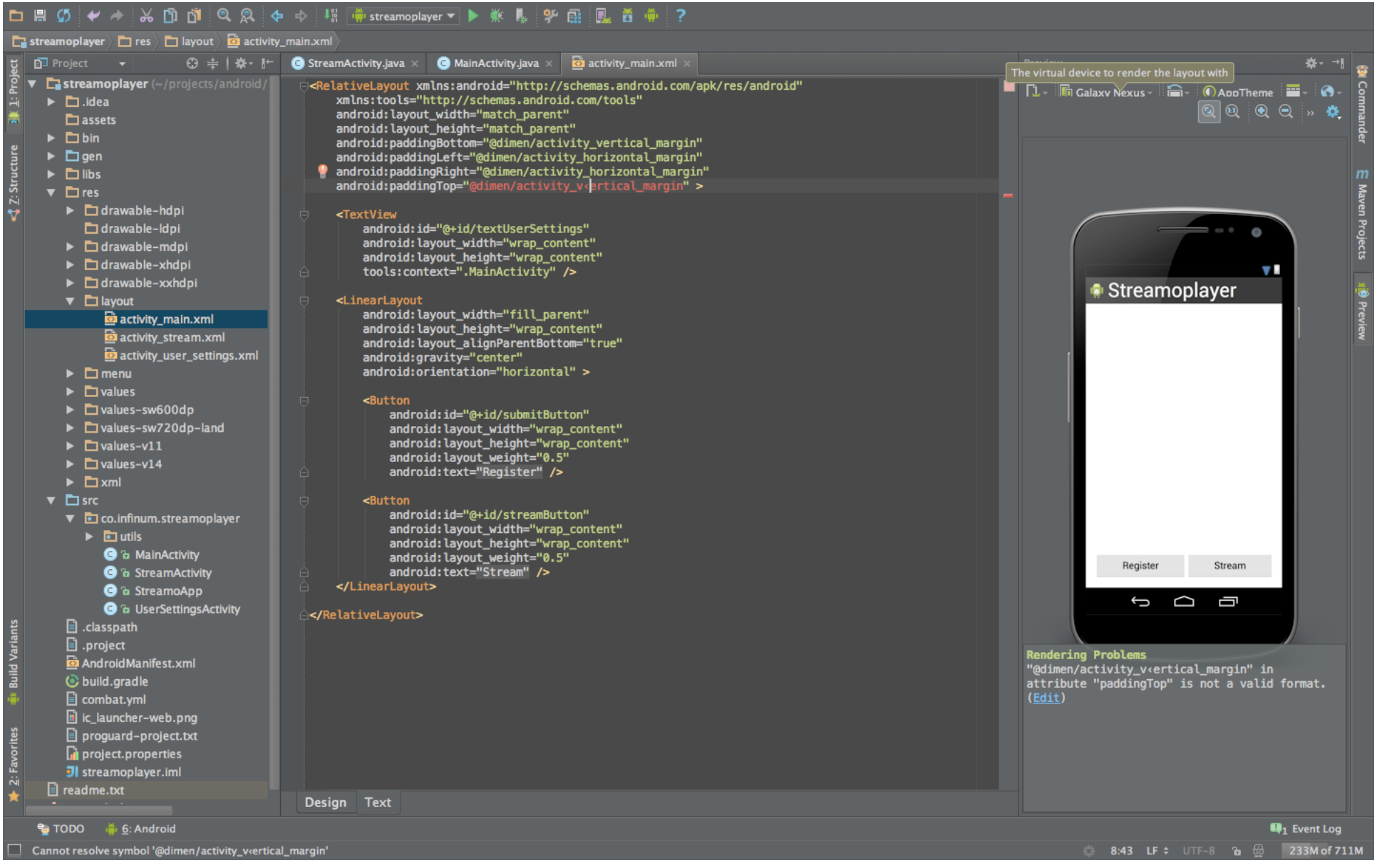
- **Intermittent**: applications that expect limited interactivity but do some work in the background. This includes media players, email clients, chat applications, etc.

- **Widgets and Live Wallpapers**: represented only as a home-screen Widget or as a Live Wallpaper

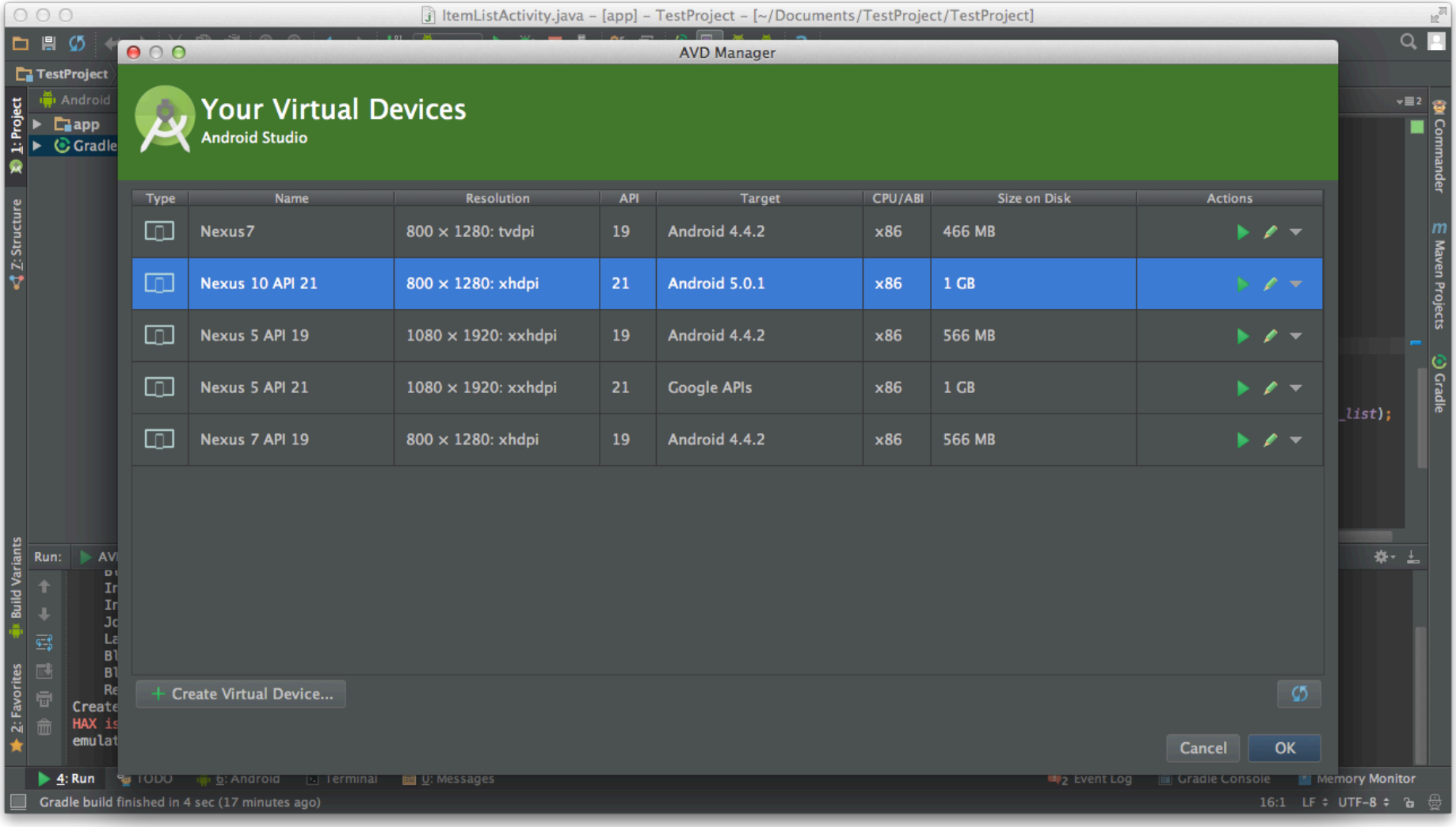
The Eclipse ADT plugin



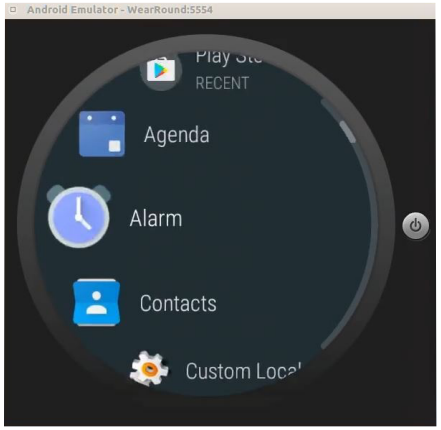
Android Studio

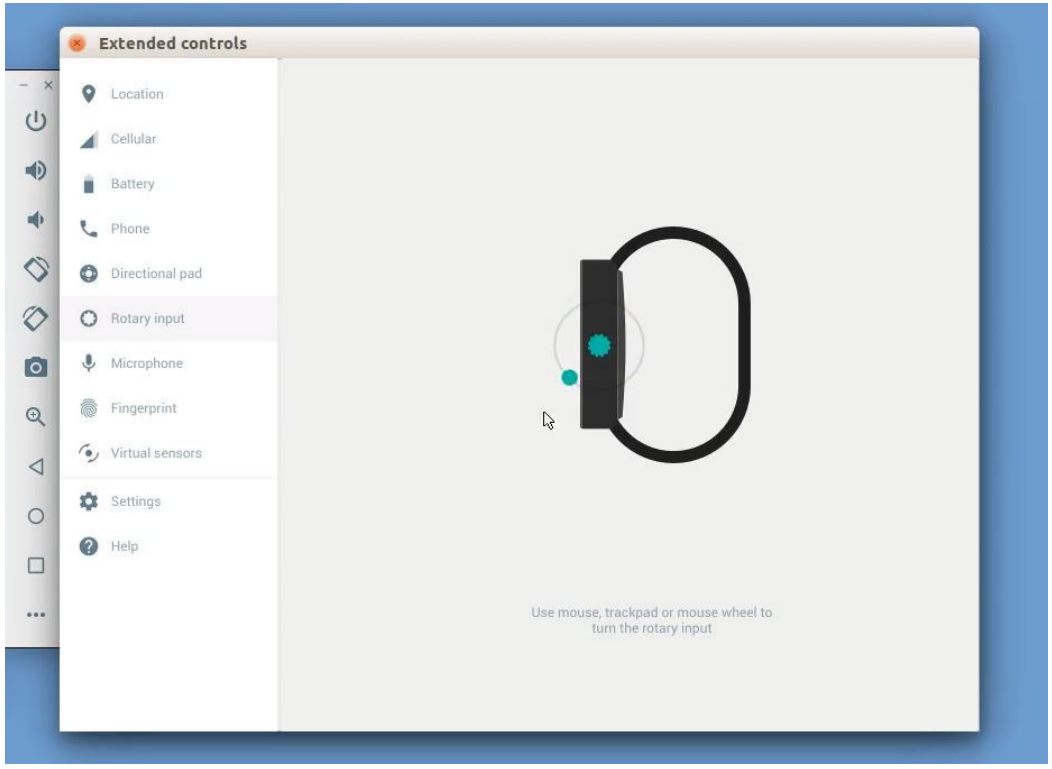


The Android Virtual Device and SDK Manager



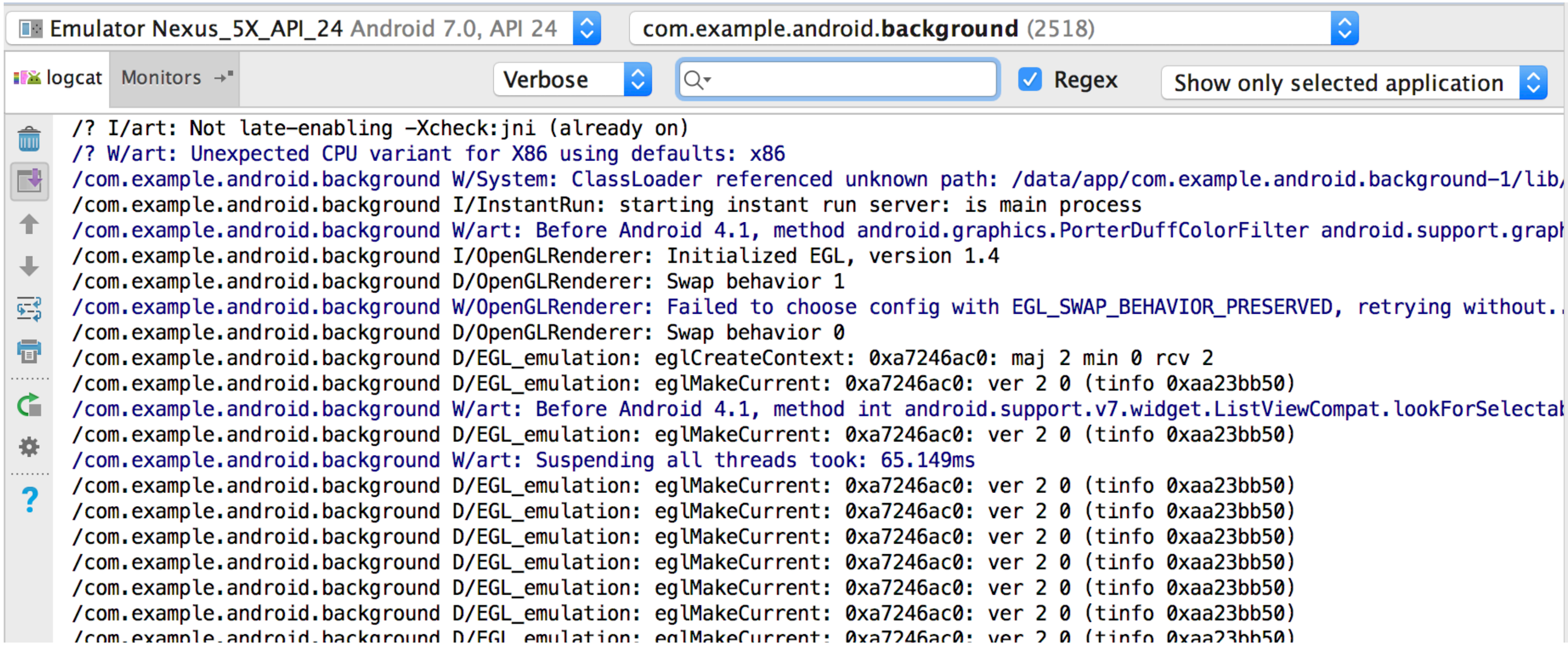
The Android Emulator





Logcat

- Logcat allows you to view and filter the output of the Android logging system



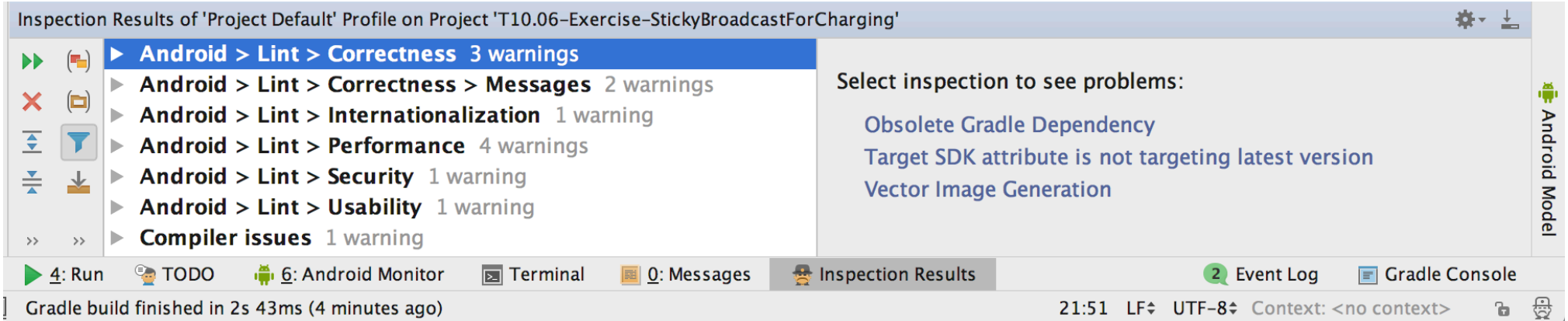
Android Debug Bridge

provides a link to virtual and physical devices. It lets you copy files, install compiled application packages (.apk), and run shell commands.

Lint

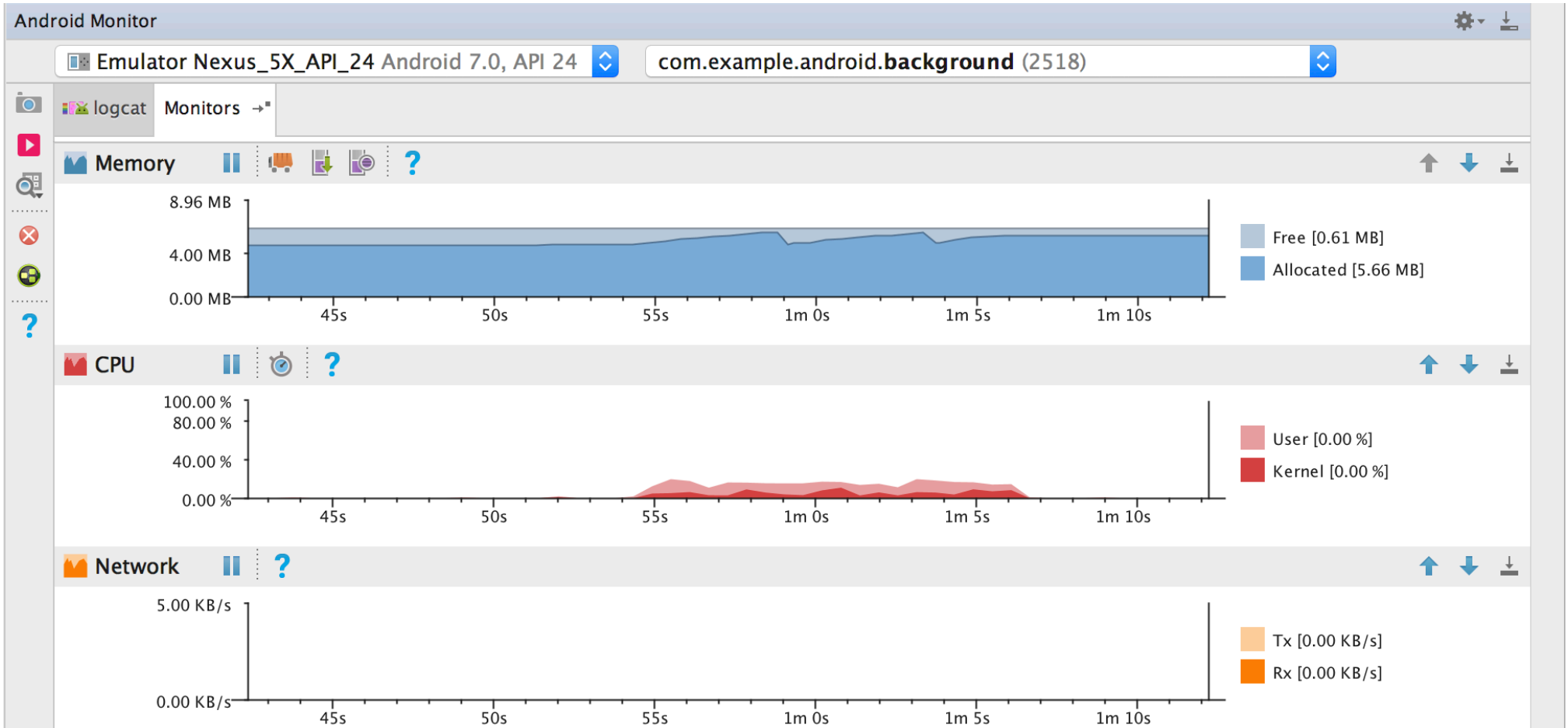
Lint analyzes your application and its resources to:

- Find possible bugs
- Suggest improvements and optimizations



Android Monitor

- Graphical analysis tools for viewing the trace logs from your Android application



Other support tools

- Monkeyrunner: an API to script user events, to automatise the testing of your application.
- ProGuard: shrinks and obfuscates your code by replacing class, variable, and method names with meaningless alternatives. This makes your code harder to reverse engineer.

The Android API

- Android provides APIs that applications can use to interact with the underlying Android system
  - A core set of **packages** and **classes**
  - A set of **XML elements** and attributes for declaring a **manifest** file
  - A set of **XML elements** and attributes for declaring and accessing **resources**
  - A set of **Intents**
  - A set of **permissions** that applications can request, as well as permission enforcements included in the system

Packages coming from Java

- **java.lang**—Core Java language classes
- **java.io**—Input/output capabilities
- **java.net**—Network connections
- **java.text**—Text-handling utilities
- **java.math**—Math and number-manipulation classes
- **javax.net**—Network classes
- **javax.security**—Security-related classes
- **javax.xml**—DOM-based XML classes
- **org.apache.***—HTTP-related classes
- **org.xml**—SAX-based XML classes

Android Core Packages

- **android.app**—Android application model access
- **android.bluetooth**—Android's Bluetooth functionality
- **android.content**—Accessing and publishing data in Android
- **android.net**—Contains the Uri class, used for accessing content
- **android.gesture**—Creating, recognising, loading, and saving gestures
- **android.graphics**—Graphics primitives
- **android.location**—Location-based services (such as GPS)
- **android.opengl**—OpenGL classes

Android Core Packages

- **android.os**—System-level access to the Android environment
- **android.provider**—ContentProvider-related classes
- **android.telephony**—Telephony capability access, including support for both Code Division Multiple Access (CDMA) and Global System for Mobile communication (GSM) devices
- **android.text**—Text layout
- **android.util**—Collection of utilities for logging and text manipulation, including XML
- **android.view**—UI elements
- **android.webkit**—Browser functionality
- **android.widget**—More UI elements

Changes to the API

- Each new version of the platform can include **updates** to the Android application framework API
- The framework API are designed to be **compatible** with earlier versions
  - **most changes** in the API **are additive** and introduce new or replacement functionality
  - the older **replaced parts** are deprecated but **are not removed**
  - In **a very small number** of cases, parts of the API **may be modified or removed**

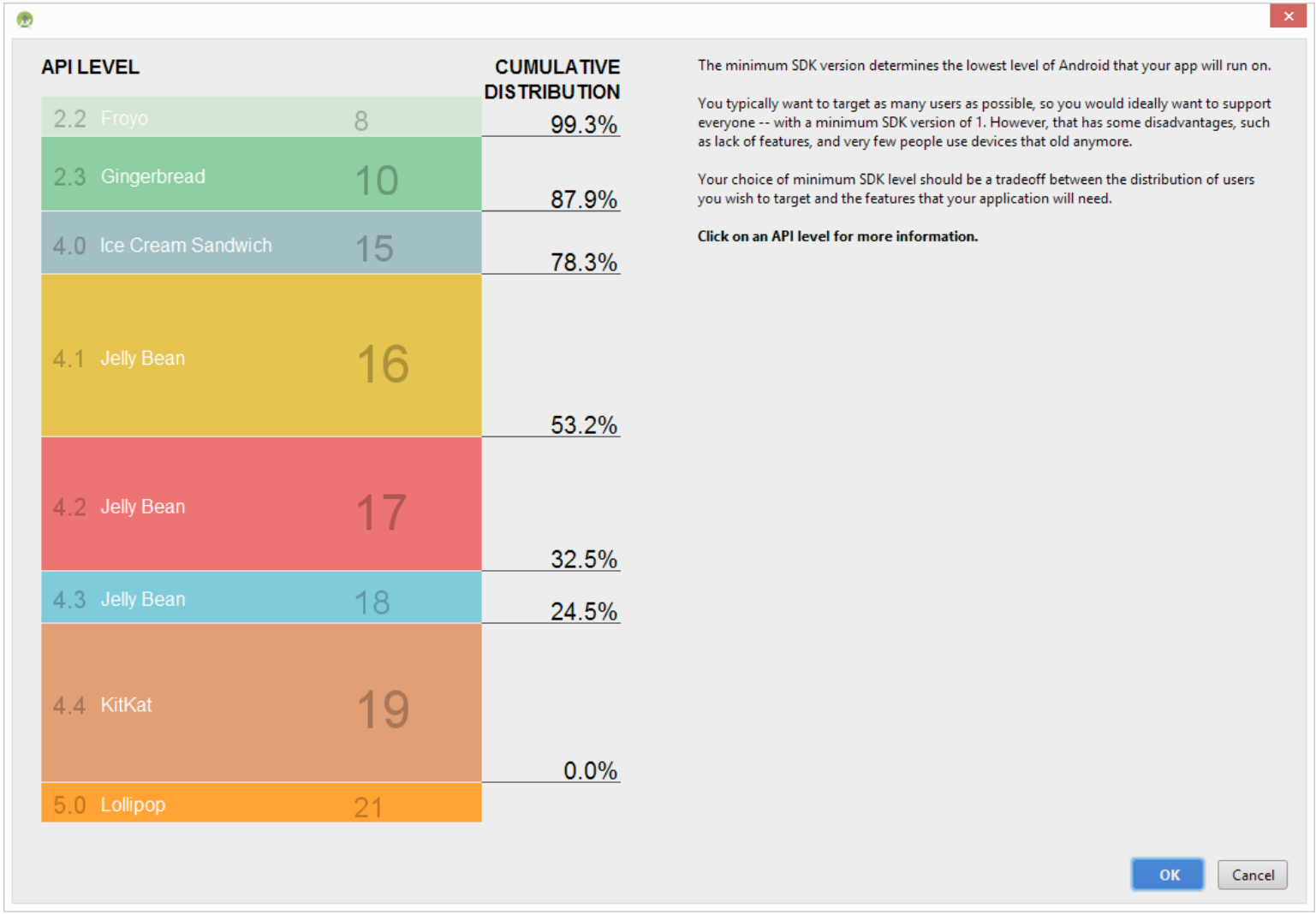
API Levels

- Each Android platform version supports exactly **one API Level**
  - support for all earlier API Levels is implicit
- API Level is an Integer value that uniquely identifies the framework API revision

Using API Levels

- Ensures the best possible choices for application developers and users:
  - the Android platform describe the maximum framework API revision that it supports
  - applications specify the framework API revision that they require
  - the system negotiates the installation of applications on the user's device, such that version incompatible applications are not installed.

Android Versions vs levels



Android Versions vs levels

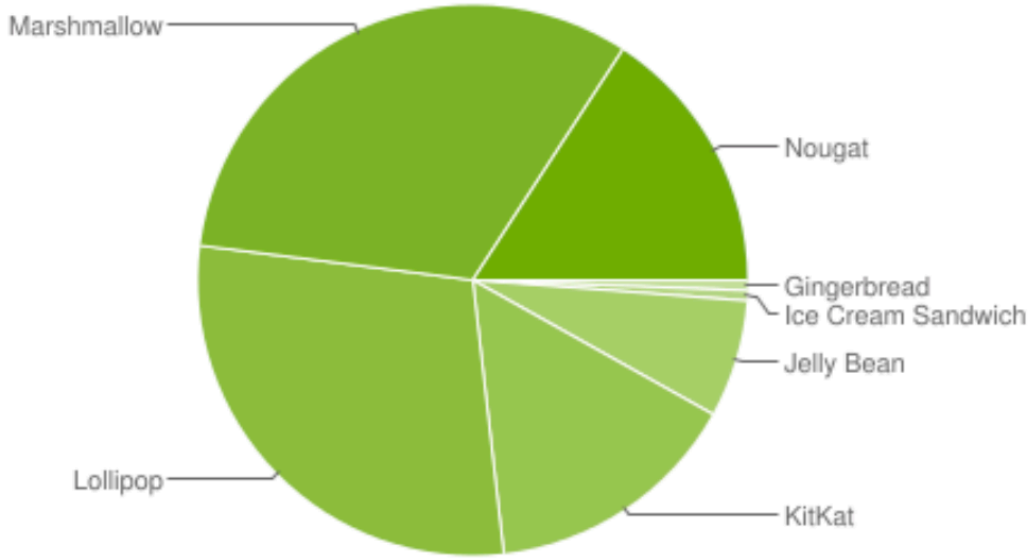
http://socialcompare.com/en/comparison/android-versions-comparison

Version name	Key user features added	Key developer features added	Release date	Android market share	API Level
Android 8.0	Oreo	• PIP: Picture-in-Picture with resizable windows • Android Instant apps • Improved notifications system • Improved system settings • Lock screen redesign		2017 Aug 21		26
Android 7.1.2	Nougat	• Battery usage alerts • Nexus and Pixel specific improvements		2017 Apr 4	1.2 % (7.1 - 7.1.2)	25
Android 7.1.1	Nougat	• Long press on the app icon enable new launch actions • The default keyboard allows now to send GIFs directly • New set of emojis		2016 Dec 5	1.2 % (7.1 - 7.1.2)	25
Android 7.1	Nougat	• Daydream Virtual Reality mode • Night Light • Storage manager improvements • Performance improvements for Touch and Display managements • Option to enable fingerprint swipe down gesture • Seamless system updates	• Shortcut manager APIs • Support Circular app icons • Keyboard image insertion • VR thread scheduling improvements • Enhanced wallpaper metadata • Multi-endpoint call support • Source type support for Visual Voicemail • Carrier config options to manage video telephony	2016 Oct 4	1.2 % (7.1 - 7.1.2)	25
Android 7.0	Nougat	• Unicode 9.0 emoji • Better multitasking • Multi-window mode (PIP, Freeform window) • Seamless system updates (with dual system partition) • Better performance and code size thanks to new JIT Compiler	• Sustained Performance Mode (SPM) API • Vulkan 3D rendering API • Daydream virtual reality platform	2016 Aug 22	12.3 %	24
Android 6.0.1	Marshmallow	New emojis		2015 Dec 7	32.3 % (6 - 6.0.1)	23
Android 6	Marshmallow	• USB Type-C support • Fingerprint Authentication support • Better battery life with "deep sleep" • Permissions dashboard • Android Pay	• Custom Chrome Tabs for better in app browser support • App Permissions management update	2015 Oct 5	32.3 % (6 - 6.0.1)	23

Check the Dashboards

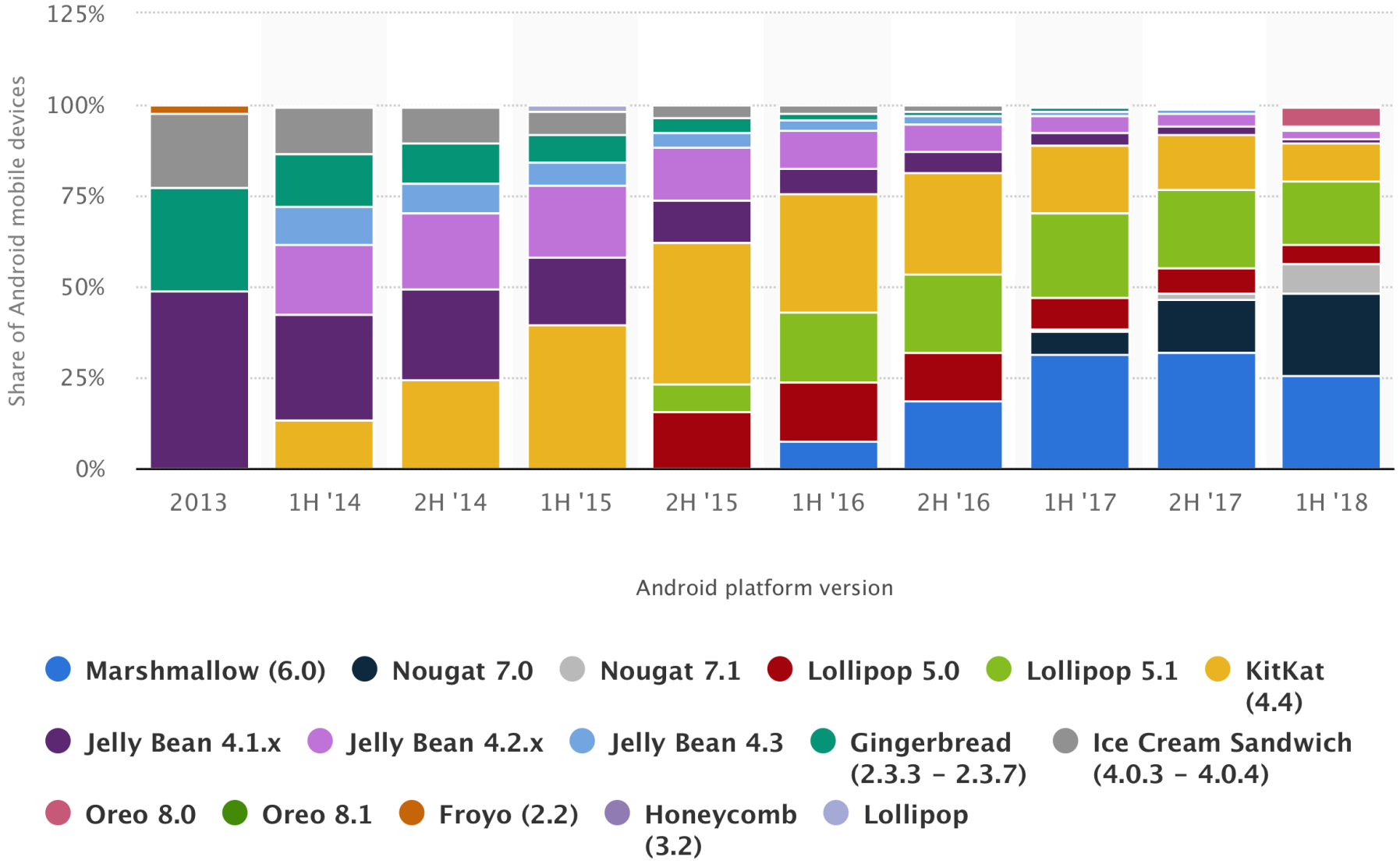
http://developer.android.com/about/dashboards/index.html

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.6%
4.1.x	Jelly Bean	16	2.4%
4.2.x		17	3.5%
4.3		18	1.0%
4.4	KitKat	19	15.1%
5.0	Lollipop	21	7.1%
5.1		22	21.7%
6.0	Marshmallow	23	32.2%
7.0	Nougat	24	14.2%
7.1		25	1.6%



	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	0.7%						0.7%
Normal		1.6%	0.2%	31.3%	36.1%	20.8%	90.0%
Large	0.1%	3.0%	1.6%	0.5%	0.9%	0.1%	6.2%
Xlarge		2.1%		0.5%	0.5%		3.1%
Total	0.8%	6.7%	1.8%	32.3%	37.5%	20.9%	

History of version Distribution



The Android Support Library

- android.support.*
- A set of libraries providing backwards compatibility for new features
- Functionality that is not available degrades gracefully
- 99+% of google play store apps use the library!

AndroidX/Jetpack

- androidx.*
- A new set of components (released in 2018) to ease Android development in many ways
- The support library is now part of JetPack
- Developers are encouraged to migrate to it; Android Studio can help

Later on

- Application forward compatibility
  - forward-compatible with new versions of the Android platform
- Application backward compatibility
  - supporting old versions of the Android platform

```
if (Build.VERSION.SDK_INT > Build.VERSION_CODES.GINGERBREAD) {
```

Before creating our first Android app …

- To write high-quality applications, it's important to **understand the components** they consist of and how those components are **bound together** by the Android manifest

Android Components

- Android Applications are made of loosely coupled components bound by the application manifest
  - Activities
  - Services
  - Content Providers
  - Intents
  - Broadcast receivers
  - Widgets
  - Notifications

Activities

- Your application's presentation layer
- Compared to desktop development, Activities are equivalent to Forms or Frames
- The UI of your application is built around one or more extensions of the Activity class

Services

- The “invisible workers” of your application
- They are components which run without a UI
  - updating your data sources and Activities
  - triggering Notifications
  - broadcasting Intents
- Used for long-running tasks, or tasks that don’t need user interactions

Content Providers

- Shareable persistent data storage
- They manage and persist application data and typically interact with SQL databases
- A mean to share data across application boundaries
- Android provide several native Content Providers
  - Media store
  - Contacts
  - Calendar

Intents

- An Inter-application message-passing framework
- They are used (extensively!) to:
  - start and stop Activities and Services
  - broadcast messages system-wide or to an explicit component
  - request an action be performed on a particular piece of data
- Can be of explicit, implicit or broadcast types

Broadcast Receivers

- Intent listeners
- They enable the application to listen for Intents that match the criteria you specify
    - They make the application event-driven

Notifications

- Alert users to application events without stealing focus or interrupting their current Activity
- Preferred way to get user attention without interrupting their tasks
- Example: Gmail applications use Notifications
  - Flashing lights
  - Playing sounds
  - Displaying icons

Widgets

- Visual application components that are typically added to the device home screen

- They enable developers to create dynamic, interactive application components for users to embed on their home screens

What is the Manifest?

- Each Android project includes an xml file in its root folder called ***AndroidManifest.xml***
- It defines the structure and metadata of the application, its components, and its requirements.
- It includes nodes for each of the **Activities**, **Services**, **Content Providers**, and **Broadcast Receivers**
- Using **Intent Filters** and **Permissions**, it determines how components interact with each other and with other applications

The Manifest also specifies metadata

- e.g. application icons, or version number
- unit tests
- platform requirements

The Manifest's root node

```
<manifest
    xmlns:android=”http://schemas.android.com/apk/res/android”
    package=“com.paad.myapp“
    android:versionCode=“1“
    android:versionName=“0.9 Beta“
    installLocation=“preferExternal“>
    [ ... manifest nodes ... ]
</manifest>
```

Manifest sub-nodes

- uses-sdk : This node enables you to define a **minimum and maximum SDK version** (API level) that must be available on a device for your application to function properly

```
<uses-sdk android:minSdkVersion=”6”
android:targetSdkVersion=”15”/>
```

- Using **maximum SDK is strongly discouraged** (newer devices would not see your application!)

Uses-feature sub-nodes

- uses-feature nodes to specify which hardware features your application requires

  <uses-feature android:name=”
  android.hardware.nfc” />

- To ensure compatibility, requiring some permissions **implies** a feature requirement

  - You can override these implied requirements by adding a required attribute and setting it to false

Uses-feature example

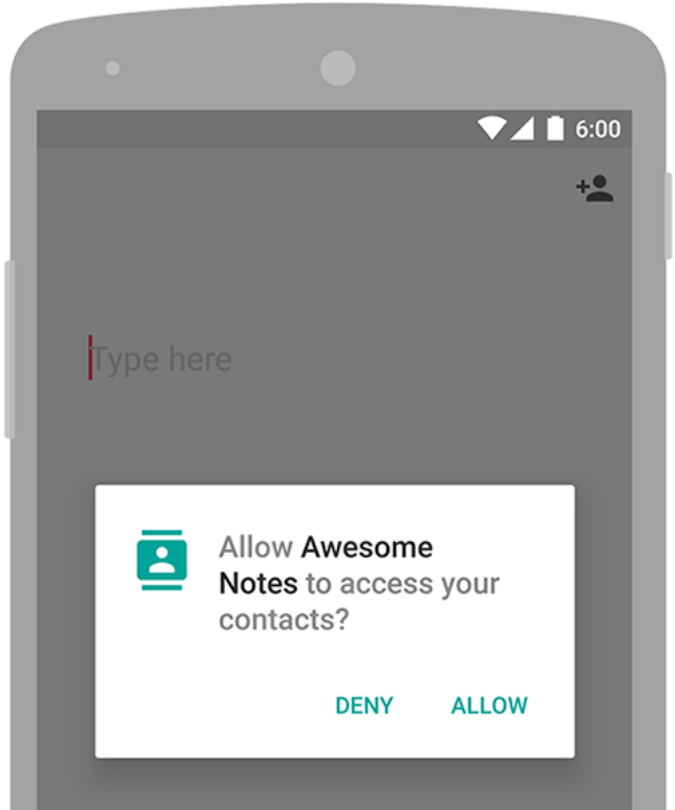
- An application that supports recording an audio note:

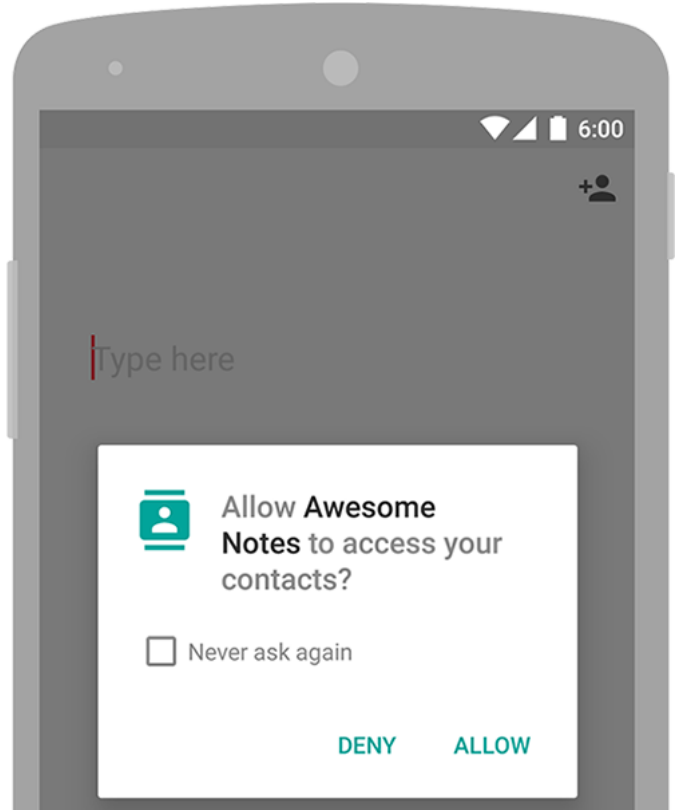
```
<uses-feature android:name=”
android.hardware.microphone”
android:required=”false” />
```

- it means that the application prefers to use the feature if present on the device

Uses-permission sub-nodes

- uses-permission: declare the user permissions your application requires
- Each permission may be presented to the user at runtime, or before the application is installed, (depending on Android version)





Permission example

- <uses-permission android:name=”
  android.permission.ACCESS_FINE_LOCATION”/>
- Each permission **may** be presented to the user
- Permisions for sensitive information (location, contacts) **will** be presented to the user
- Users may **deny** the permission

Uses-permission vs Uses-feature

- <uses-feature> specify hardware requirements and their strength
- If <uses-feature> is not specified, <uses-permission> assumes a strong hardware requirement. This can affect how your application is filtered by Google Play.
- for example requesting permission for CAMERA
  - it assumes that the app requires the underlying hardware feature and filters the application from devices that do not offer it

Permissions can be fine-grained

<uses-feature android:name=”
android.hardware.camera” />

<uses-feature android:name=”
android.hardware.camera.autofocus”
android:required=”false” />

<uses-feature android:name=”
android.hardware.camera.flash” android:required=”
false” />

Application sub-node

- application: A manifest can contain only one application node
- it specifies the metadata for the application
- The application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes

Application node example

```
<application
  android:icon=”@drawable/icon”
  android:logo=”@drawable/logo”
  android:theme=”@android:style/Theme.Light”
  android:name=”.MyApplicationClass”
  android:debuggable=”true”>
  [ ... application nodes ... ]
</application>
```

Activity sub-node

- activity: is required for every Activity within the application
- Use the android:name attribute to specify the Activity class name
- Each Activity node supports intent-filter child tags that define the Intents that can be used to start the Activity

Activity example

```
<activity android:name=".MyActivity" android:label="@string/
app_name">

  <intent-filter>

      <action android:name="android.intent.action.MAIN" />

      <category

          android:name="android.intent.category.LAUNCHER" />

  </intent-filter>

</activity>
```

Further example

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

Service sub-node

- service: add a service tag for each Service class used in your application

  <service android:name=”.MyService”> </service>

- Services should not have Intent filters

Caution: To ensure that your app is secure, always use an explicit intent when starting a **`Service`** and do not declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you can't be certain what service will respond to the intent, and the user can't see which service starts. Beginning with Android 5.0 (API level 21), the system throws an exception if you call **`bindService()`** with an implicit intent.

Provider sub-node

- provider: specifies each of your application's Content Providers. Content Providers are used to manage database access and sharing.

  <provider android:name=".MyContentProvider" android:authorities="com.paad.myapp.MyContentProvider"/>

Receiver sub-node

- To register a Broadcast Receiver
- Broadcast Receivers are like global event listeners that, when registered, will execute whenever a matching Intent is broadcast by the system or an application

```
<receiver android:name=".MyIntentReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED">
    </intent-filter>
</receiver>
```

★ **Note:** If your app targets API level 26 or higher, you cannot use the manifest to declare a receiver for *implicit* broadcasts (broadcasts that do not target your app specifically), except for a few implicit broadcasts that are exempted from that restriction. In most cases, you can use scheduled jobs instead.

Uses-library sub-node

- To specify a library the application needs that is not included by default

  <uses-library android:name=”
  com.google.android.maps” android:required=”
  false”/>

Other Manifest nodes

- uses-configuration : to specify external hardware (e.g. a keyboard)
- support-screens: to specify supported screen sizes.
- Instrumentation: to specify test classes