Google™ 10 I/O

A Beginner's Guide to Android

Reto Meier
May 19, 2010

@retomeier

What is Android?

- An open source, open platform for mobile development
- All the SDK, API, and platform source is available
- No licensing, no app review
- Replace any system app with your own

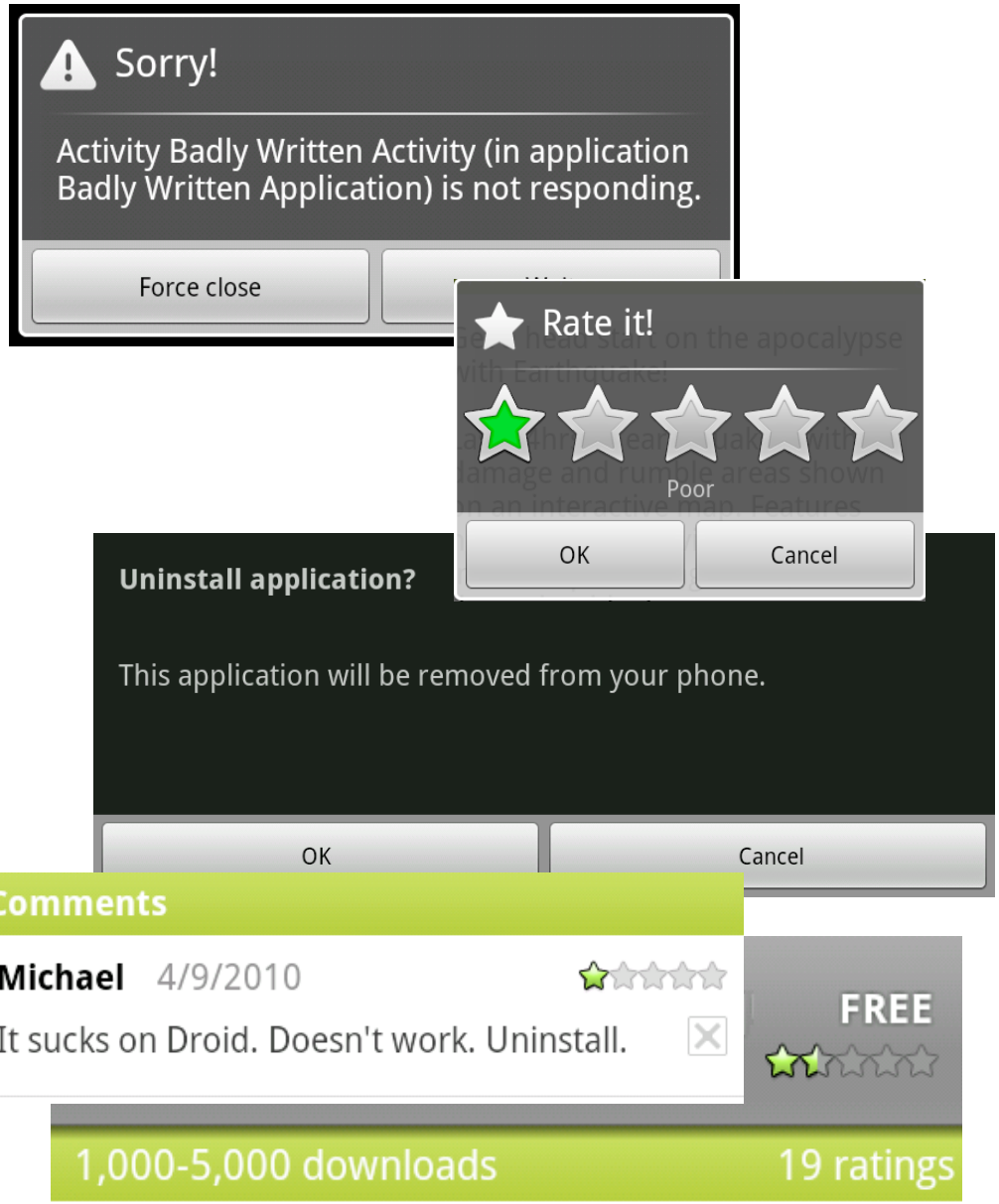
developer.android.com

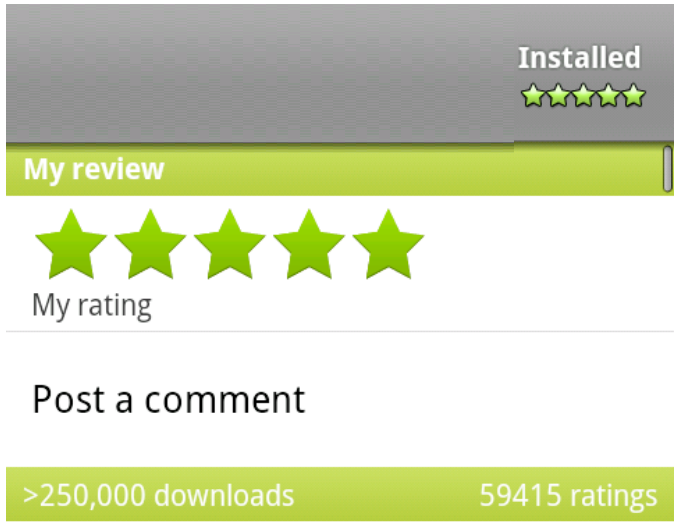
Android Best Practices for Beginners

Reto Meier
May 19, 2010

Your Choice

Your Consequences





Google I/O 10

Agenda

- The Five Deadly Sins
- The Five Glorious Virtues
- Two Practical Examples

The Five Deadly Sins

The Five Deadly Sins

SLOTH

Be Fast. Be Responsive.

The Golden Rules of Performance

- Don't do work that you don't need to do
- Don't allocate memory if you can avoid it

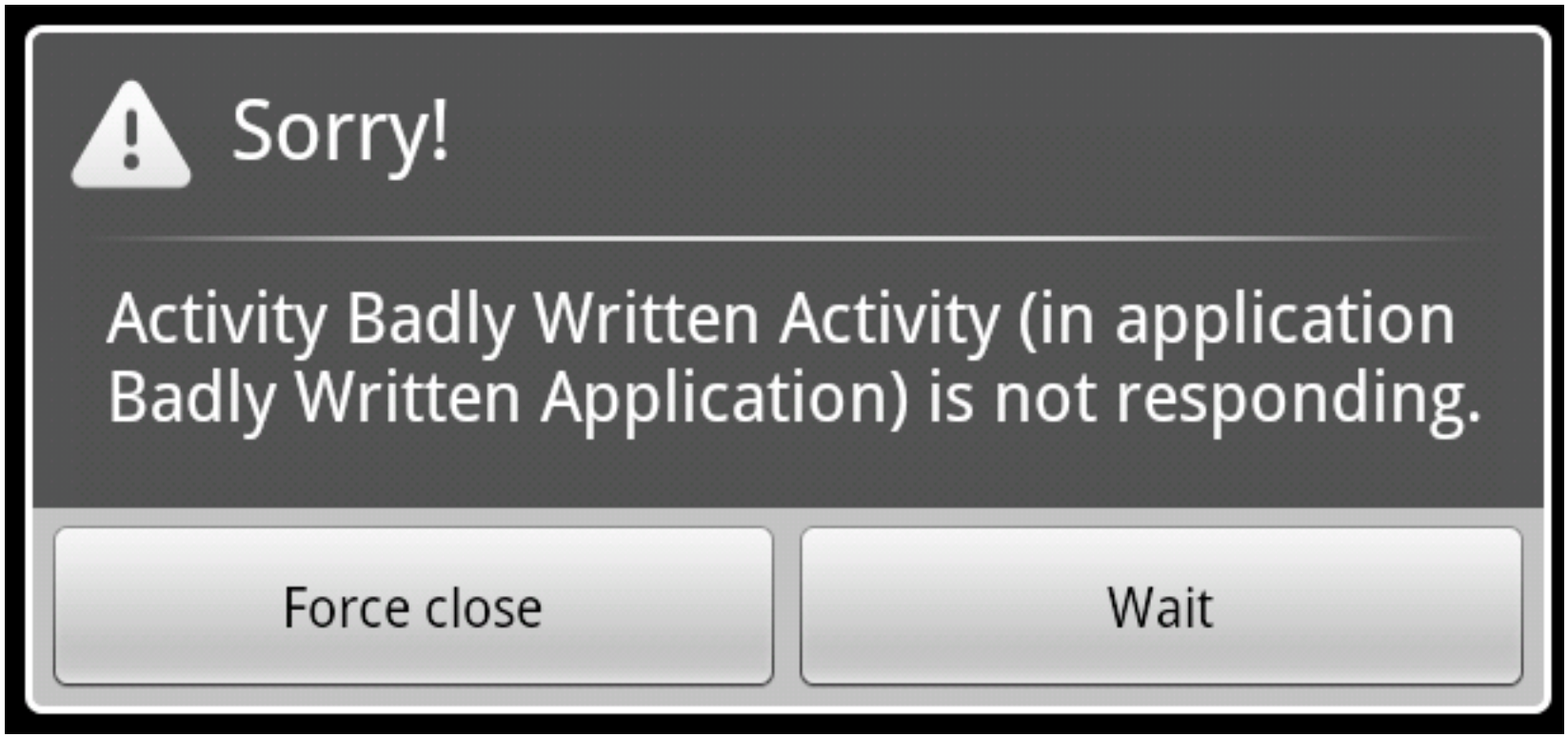
Performance Pointers

- Optimize judiciously
- **Avoid creating objects**
- Use native methods
- Prefer Virtual over Interface
- Prefer Static over Virtual
- Avoid internal setters and getters
- Declare constants final
- Avoid float and enums
- Use package scope with inner classes

Responsiveness

- Avoid modal Dialogues and Activities
  - Always update the user on progress (ProgressBar and ProgressDialog)
  - Render the main view and fill in data as it arrives
- "Application Not Responding"
  - Respond to user input within **5 seconds**
  - Broadcast Receiver must complete in **10 seconds**
- Users perceive a lag longer than **100 to 200ms**
- Use Threads and AsyncTasks within Services

Application Not Responding



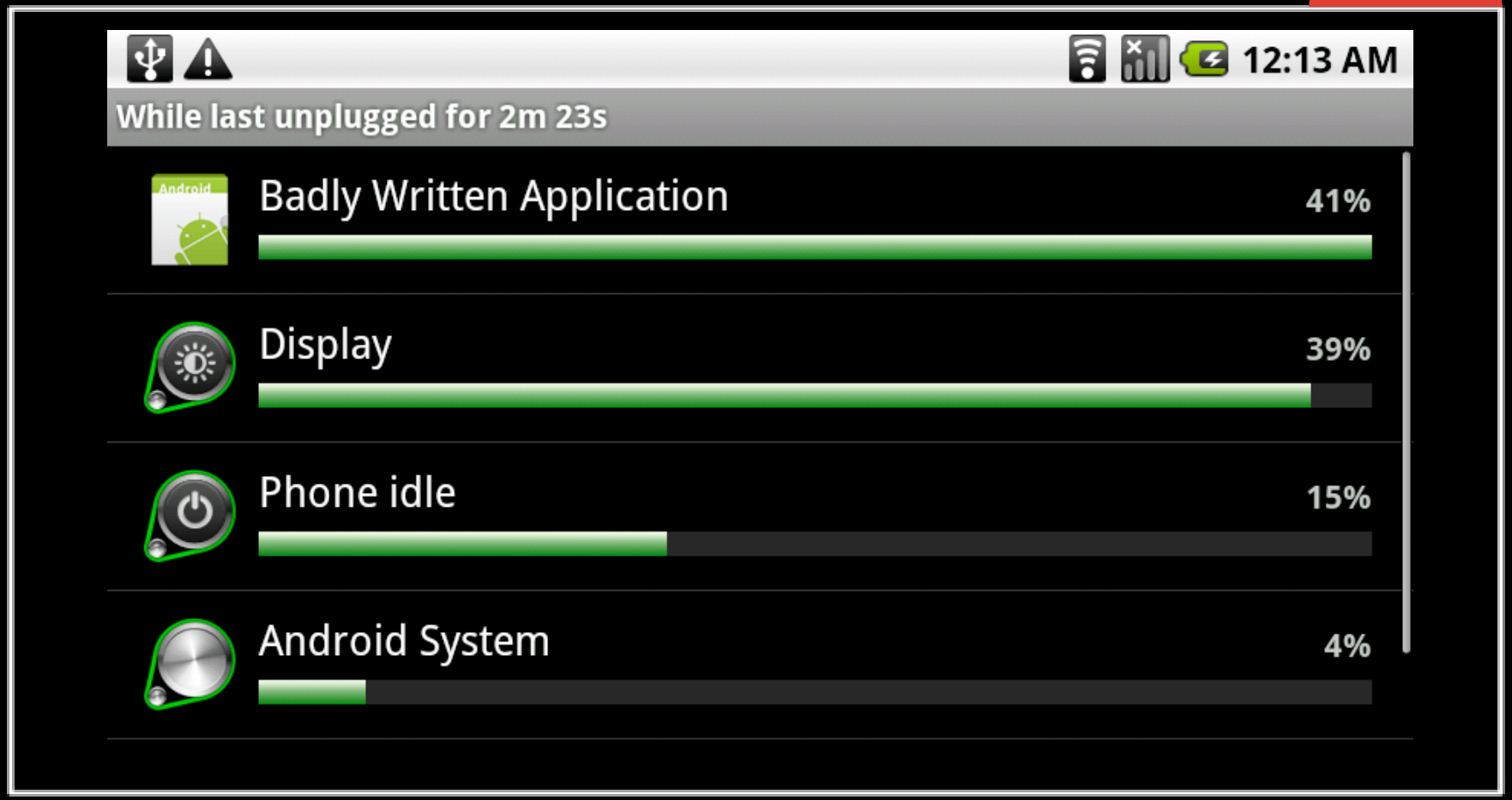
Responsiveness

- Avoid modal Dialogues and Activities
  - Always update the user on progress
  - Render the main view and fill in data as it arrives
- "Application Not Responding"
  - Respond to user input within **5 seconds**
  - Broadcast Receiver must complete in **10 seconds**
- Users perceive a lag longer than **100 to 200ms**
- Use Threads and AsyncTasks within Services

AsyncTask

```
protected void doInBackground(Void... arg0) {
   // Do time consuming processing
   publishProgress();
   return null;
}

protected void onProgressUpdate(Void... arg0) {
}

protected void onPostExecute(Void result) {
}
```

The Five Deadly Sins



GLUTTONY

Use system resources responsibly

Gluttony

Don'ts

- **DON'T** over use WakeLocks
- **DON'T** update Widgets too frequently
- **DON'T** update your location unnecessarily
- **DON'T** use Services to try to override users or the system

Dos

- **DO** share data to minimize duplication
- **DO** use Receivers and Alarms not Services and Threads
- **DO** let users manage updates
- **DO** minimize resource contention

What is a WakeLock?

- Force the CPU to keep running
- Force the screen to stay on (or stay bright)
- Drains your battery quickly *and* efficiently

```
PowerManager pm =
  (PowerManager)getSystemService(Context.POWER_SERVICE);

PowerManager.WakeLock wl =
  pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK,
                 "My Wakelock");

wl.acquire(10000);
  // Screen and power stays on
wl.release();
```

Using WakeLocks

- Do you really need to use one?
- Use the minimum level possible
  - PARTIAL_WAKE_LOCK
  - SCREEN_DIM_WAKE_LOCK
  - SCREEN_BRIGHT_WAKE_LOCK
  - FULL_WAKE_LOCK
- Release as soon as you can
- Specify a timeout
- Don't use them in Activities

Window Managed WakeLocks

- No need for permissions
- No accidently leaving the screen from the background

```
getWindow().addFlags(
  WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

The Five Deadly Sins

HOSTILITY

Don't fight your users

Hostility

- User experience should be your top priority

Hostility

- User experience should be your top priority
- Respect user expectations for navigating your app

Doing what the user expects with respect to navigation flow is absurdly important for overall user satisfaction.

Respect User Expectations for Navigation

- The back button should **always** navigate back through previously seen screens
- Always support trackball navigation
- Understand your navigation flow when entry point is a notification or widget
- Navigating between application elements should be easy and intuitive

Hostility

- User experience should be your top priority
- Respect user expectations for navigating your app
- Don't hijack the native experience

Don't Hijack the Native Experience

- Don't hide the status bar
- Back button should always navigate through previous screens
- Use native icons consistently
- Don't override the menu button
- Put menu options behind the menu button

Hostility

- User experience should be your top priority
- Respect user expectations for navigating your app
- Don't hijack the native experience
- Respect user preferences

Respect User Preferences

- Use only enabled location-based services
- Ask permission before transmitting location data
- Only transfer data in the background if user enabled

```
ConnectivityManager cm = (ConnectivityManager)
  getSystemService(Context.CONNECTIVITY_SERVICE);

boolean backgroundEnabled =
  cm.getBackgroundDataSetting();
```

The Five Deadly Sins

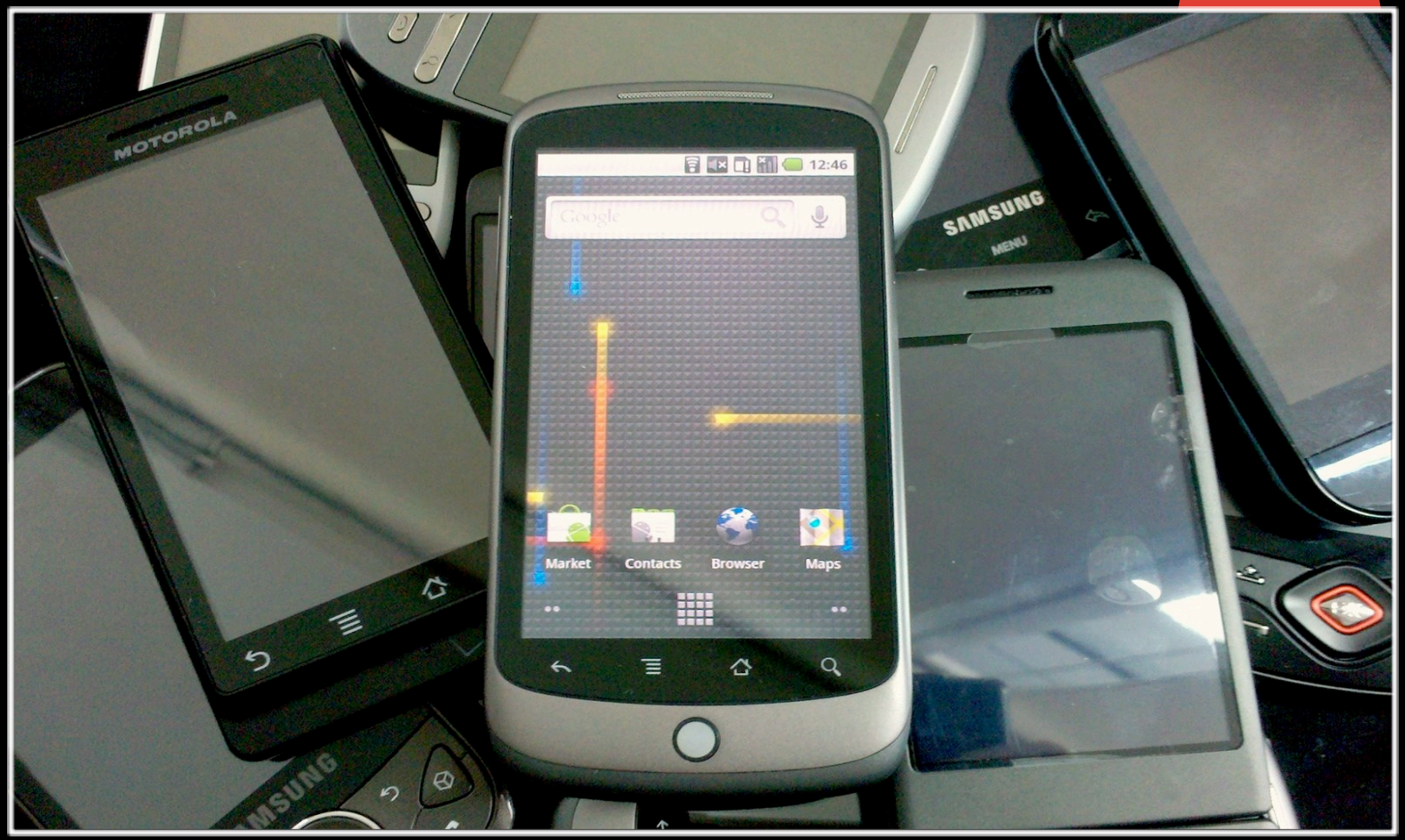
ARROGANCE

Don't fight the system

Arrogance

- Don't use undocumented APIs
- Seriously. Don't use undocumented APIs
- Make your app behave consistently with the system
- Respect the application lifecycle model

The Five Deadly Sins

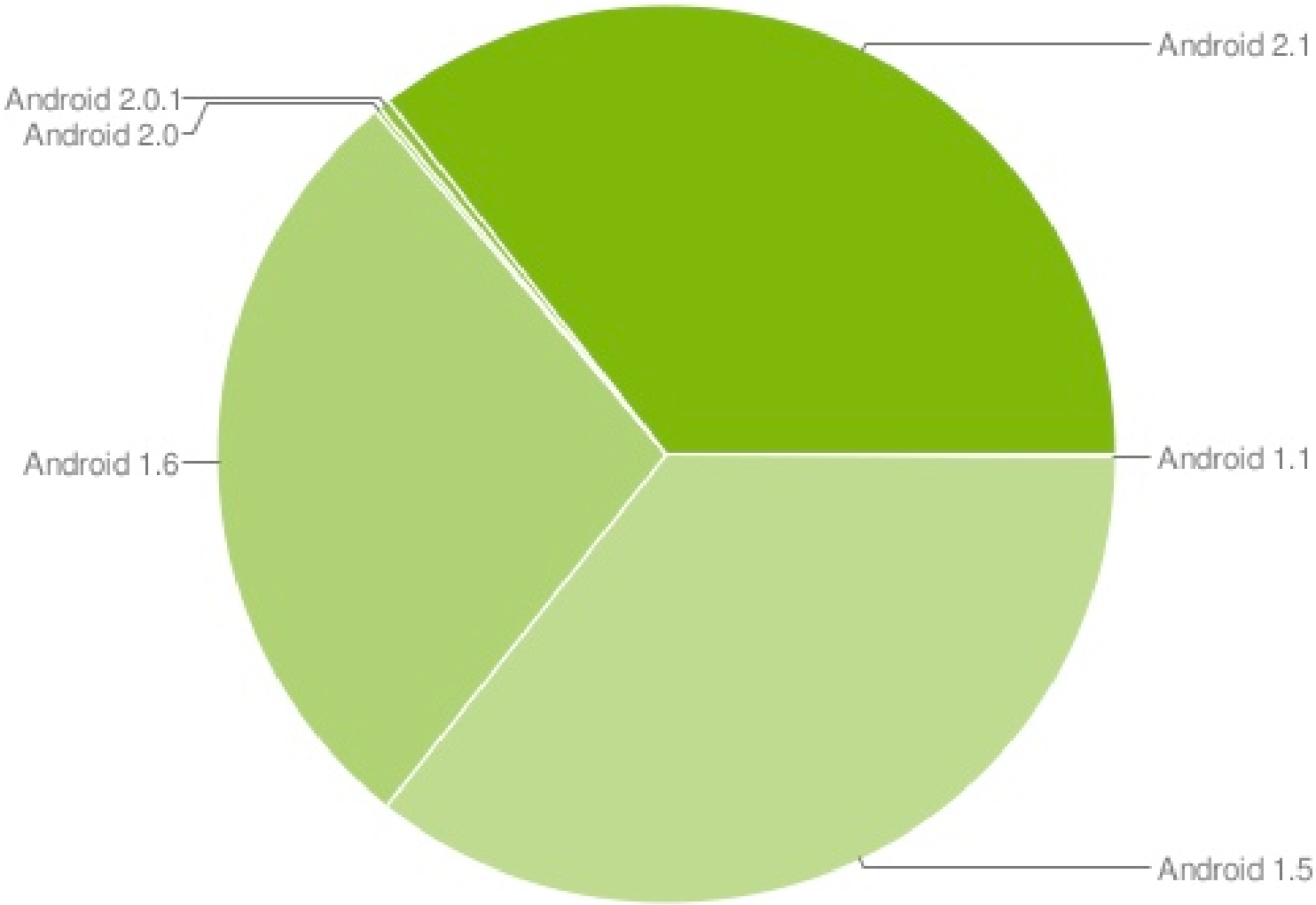


DISCRIMINATION

Design for everyone

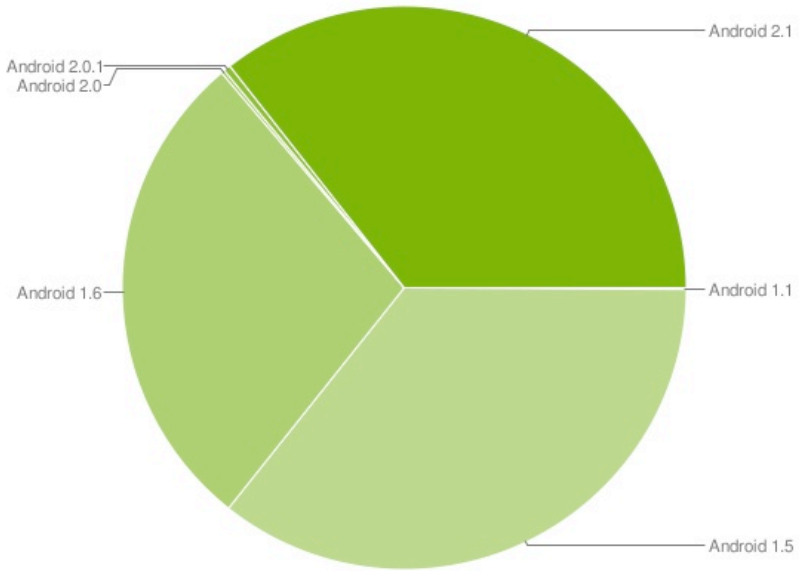
Google 10 I/O

Discrimination



Discrimination

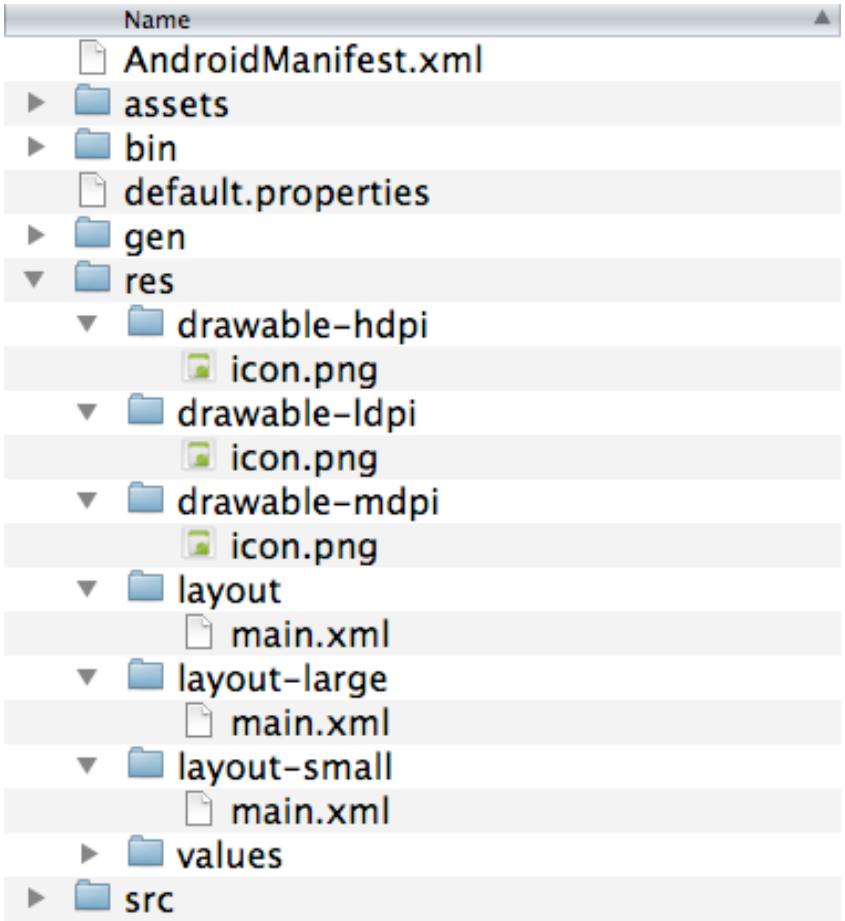
- Don't make assumptions about screen size or resolution
- Never hard-code string values in code (or XML)
- Use Relative Layouts and device independent pixels
- Optimize assets for different screen resolutions
- Use reflection to determine what APIs are available

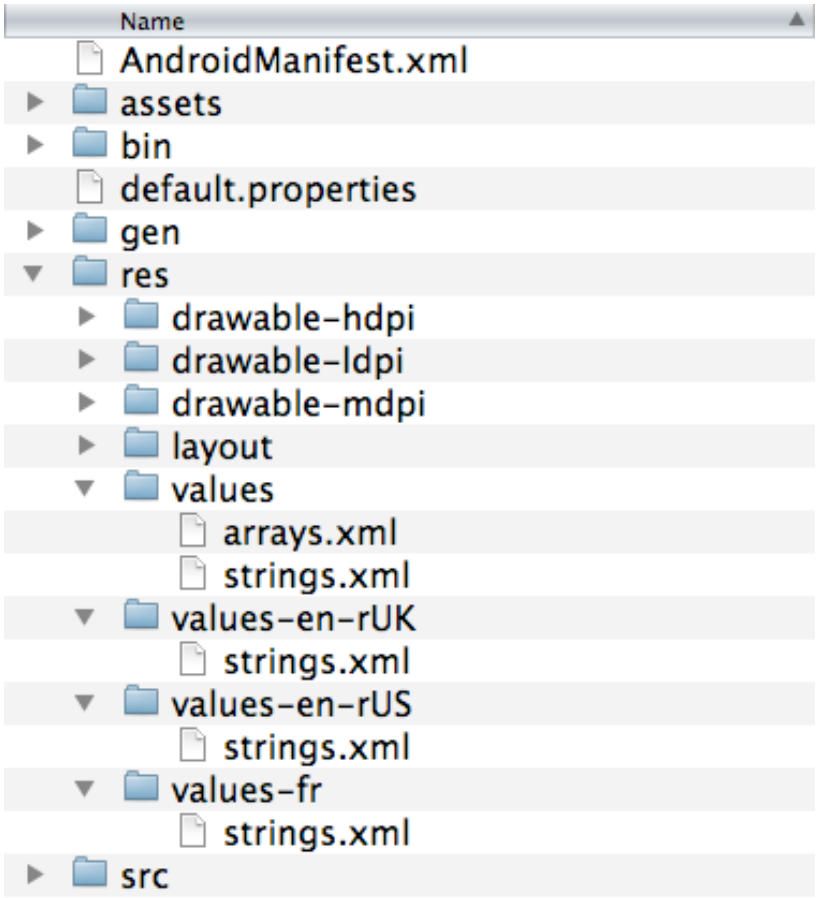


Store Values as Resources

- Define strings, colors, dimensions, and arrays
- Also store images and layouts
- Never rely on hard-coded values
- Reference resources in code and XML
- System will select from the right resource folder

Resource Hierarchy





Agenda

- The Five Deadly Sins
- The Five Glorious Virtues
- Two Practical Examples

The Five Glorious Virtues

The Five Glorious Virtues

BEAUTY

Hire a designer

Beauty

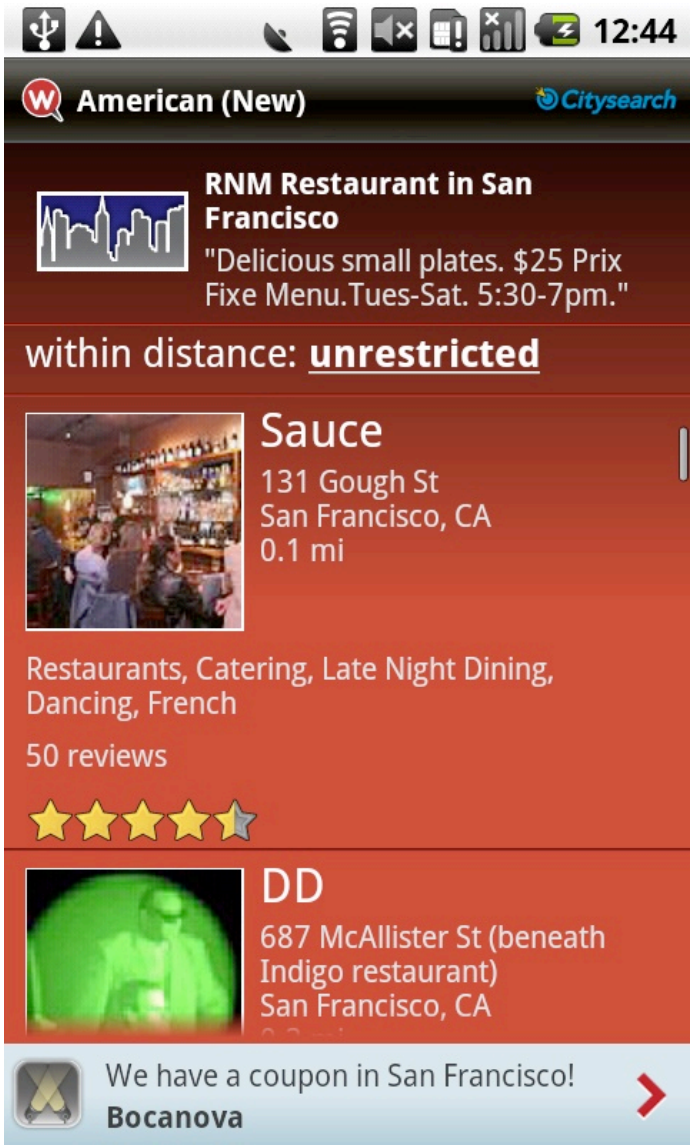
- Programmers are not designers!
- 4.15pm today "Android UI Design Patterns"
- Create assets optimized for all screen resolutions
  - Start with vectors or high-res raster art
  - Scale down and optimize for supported screen
- Support resolution independence
- Use tools to optimize your implementation
  - layoutopt
  - hierarchyviewer

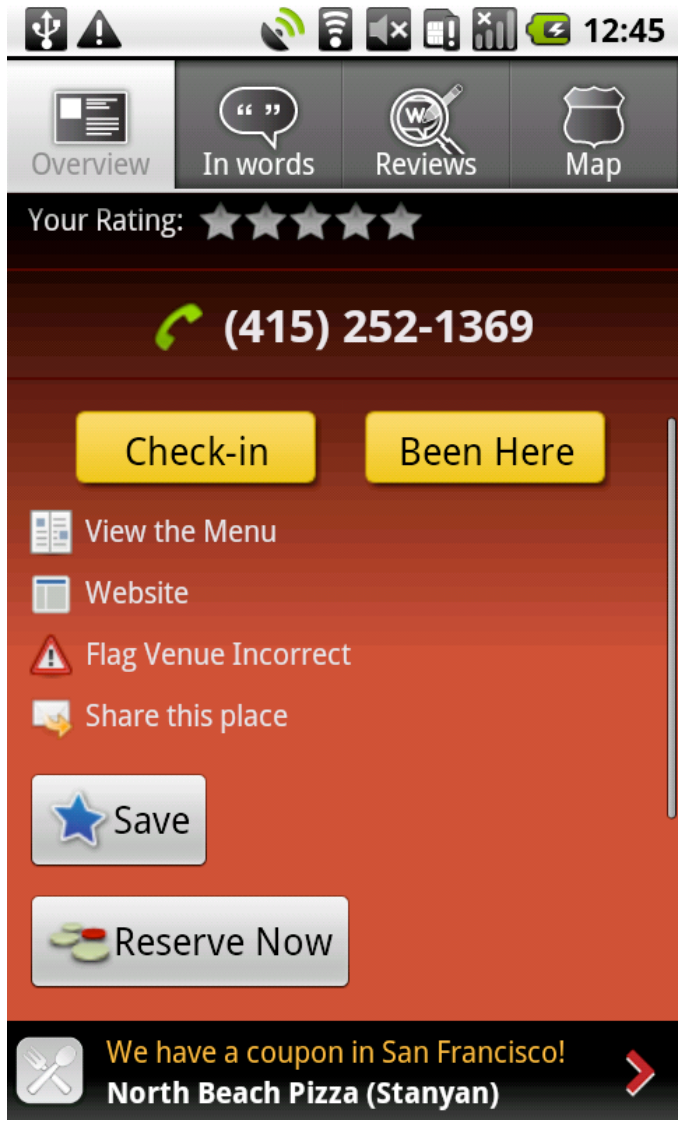
The Five Glorious Virtues

GENEROSITY

Share and consume

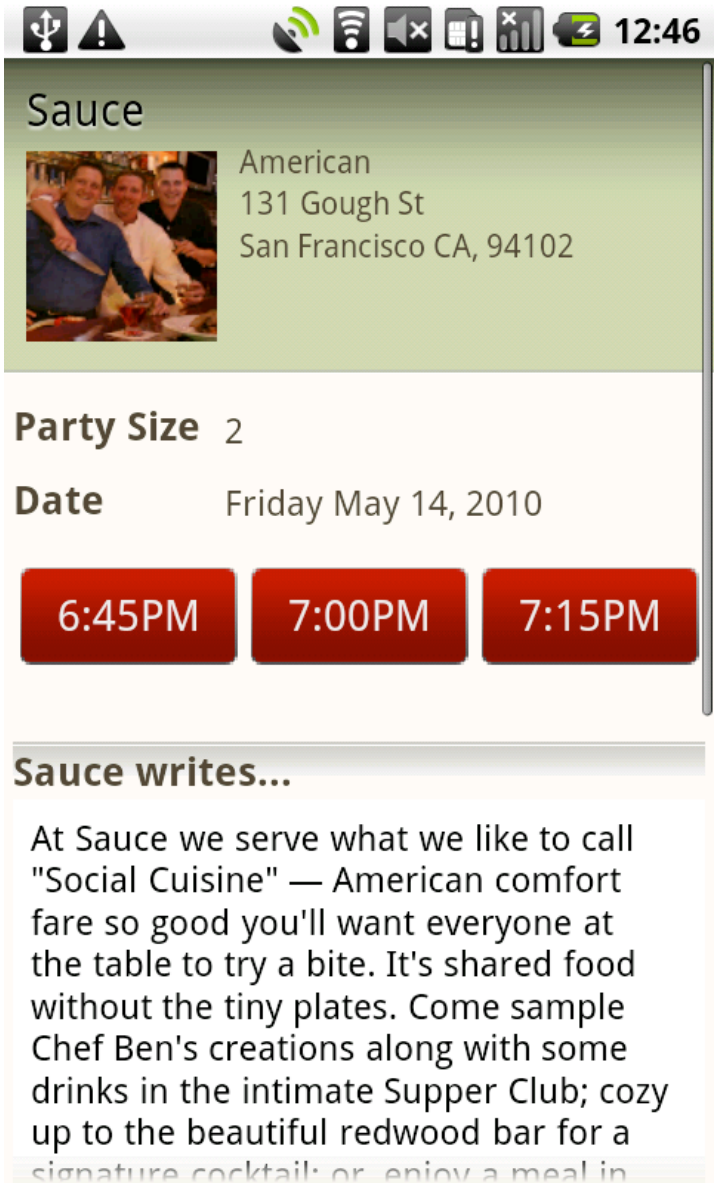
Where and OpenTable





Google I/O 10

Where and OpenTable



Google I/O 10

Generosity

- Use Intents to leverage other people's apps
- Define Intent Filters to share your functionality

Using Intents to Start Other Apps

- Works just like your own Activity
- Can pass data back and forth between applications
- Return to your Activity when closed

```
String action = "com.hotelapp.ACTION_BOOK";

String hotel = "hotel://name/" + selectedhotelName;
Uri data = Uri.parse(hotel);

Intent bookingIntent = new Intent(action, data);
startActivityForResult(bookingIntent);
```

Activity Intent Filters

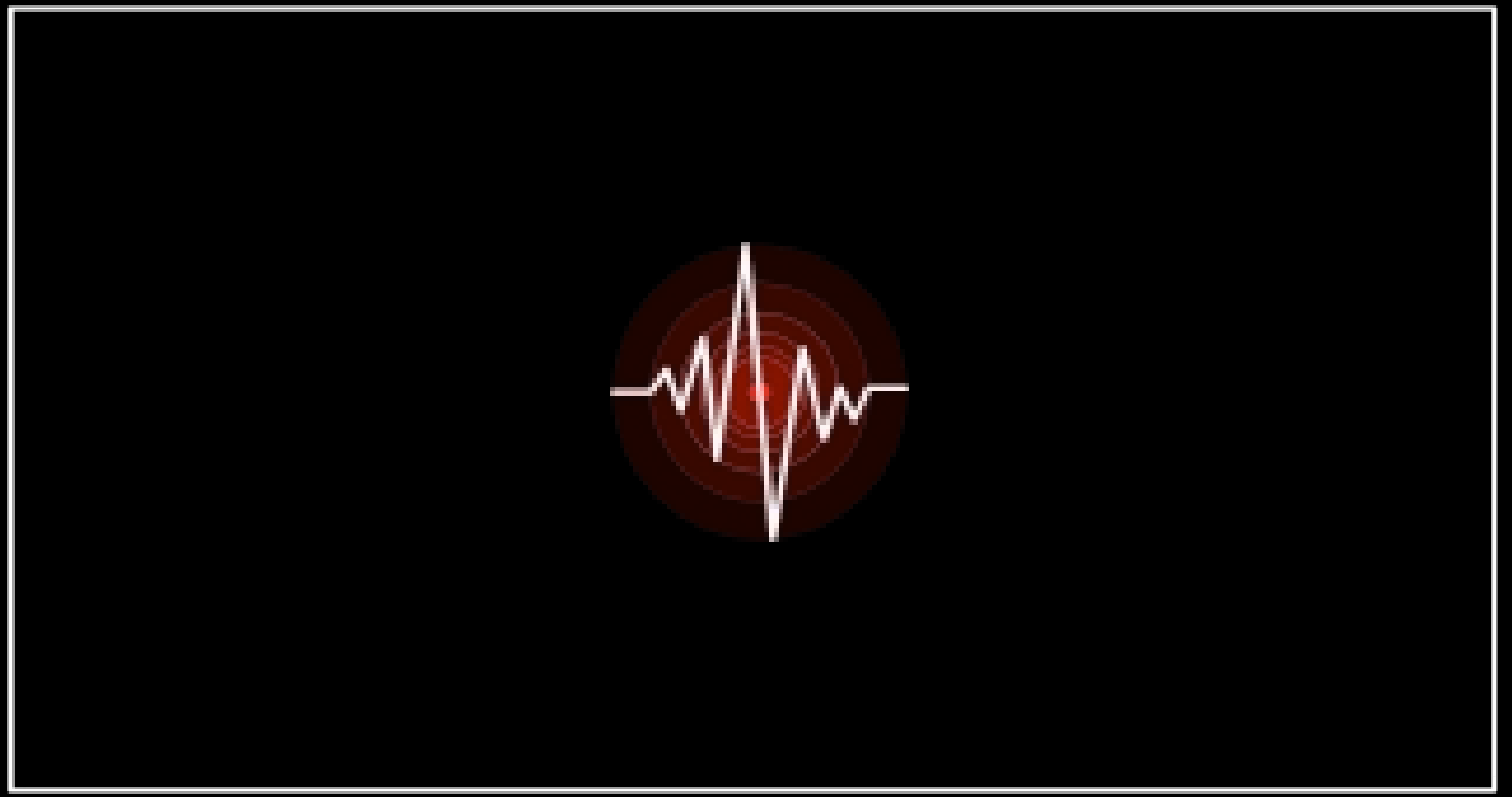
- Indicate the ability to perform an action on data
- Specify an action you can perform
- Specify the data you can perform it on

```
<activity android:name="Booking" android:label="Book">
  <intent-filter>
    <action android:name="com.hotelapp.ACTION_BOOK"/>
    <data android:scheme="hotel"
          android:host="name"/>
  </intent-filter>
</activity>
```

Activity Intent Filters

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(r.layout.main);

  Intent intent = getIntent();

  String action = intent.getAction();
  Uri data = intent.getData();

  String hotelName = data.getPath();

  // TODO Provide booking functionality

  setResult(RESULT_OK, null);
  finish();
}
```

The Five Glorious Virtues



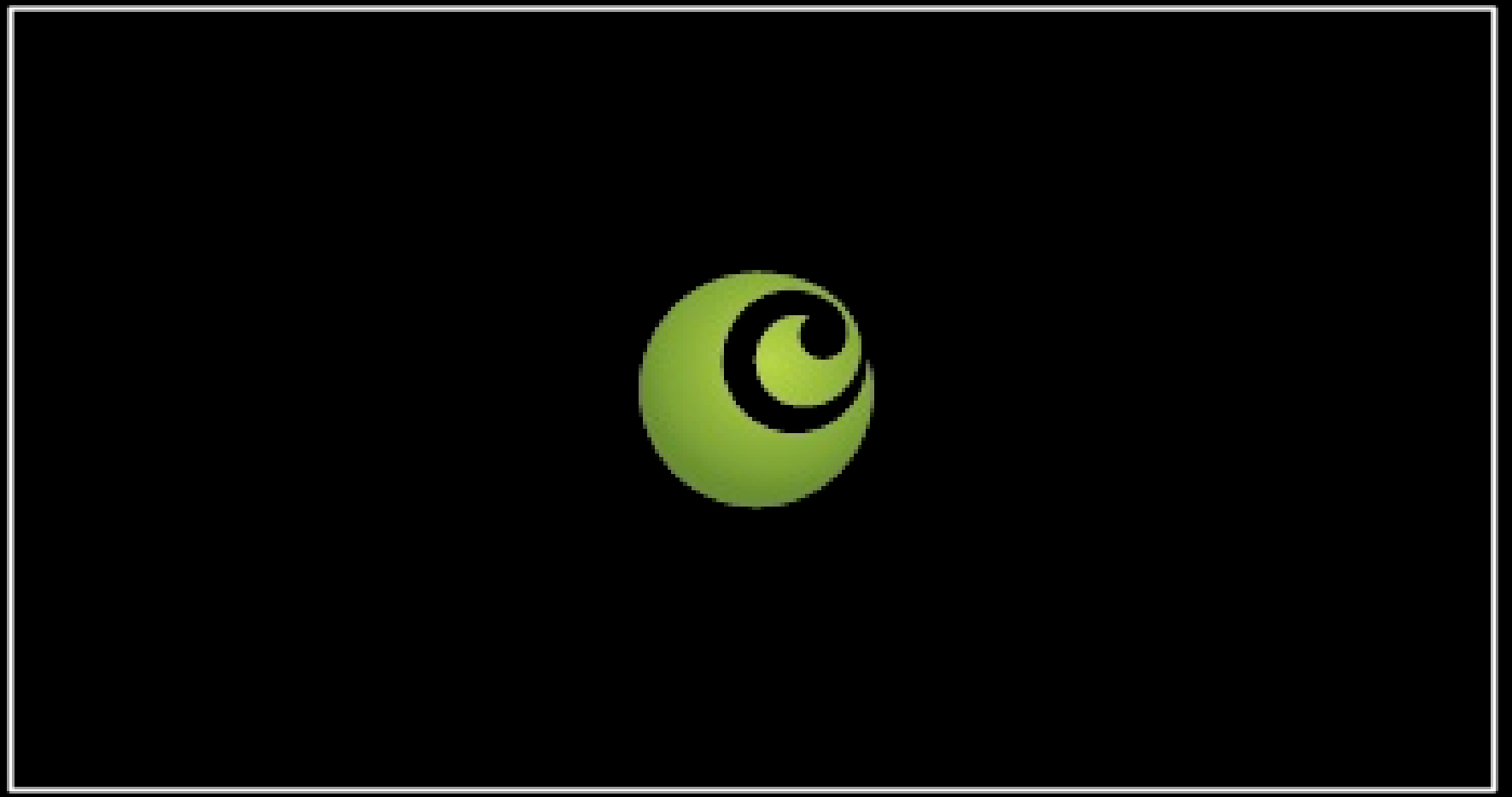
UBIQUITY

Be more than an icon

Ubiquity

- Create widgets
- Surface search results into the Quick Search Box
- Live Folders
- Live Wallpapers
- Expose Intent Receivers to share your functionality
- Fire notifications

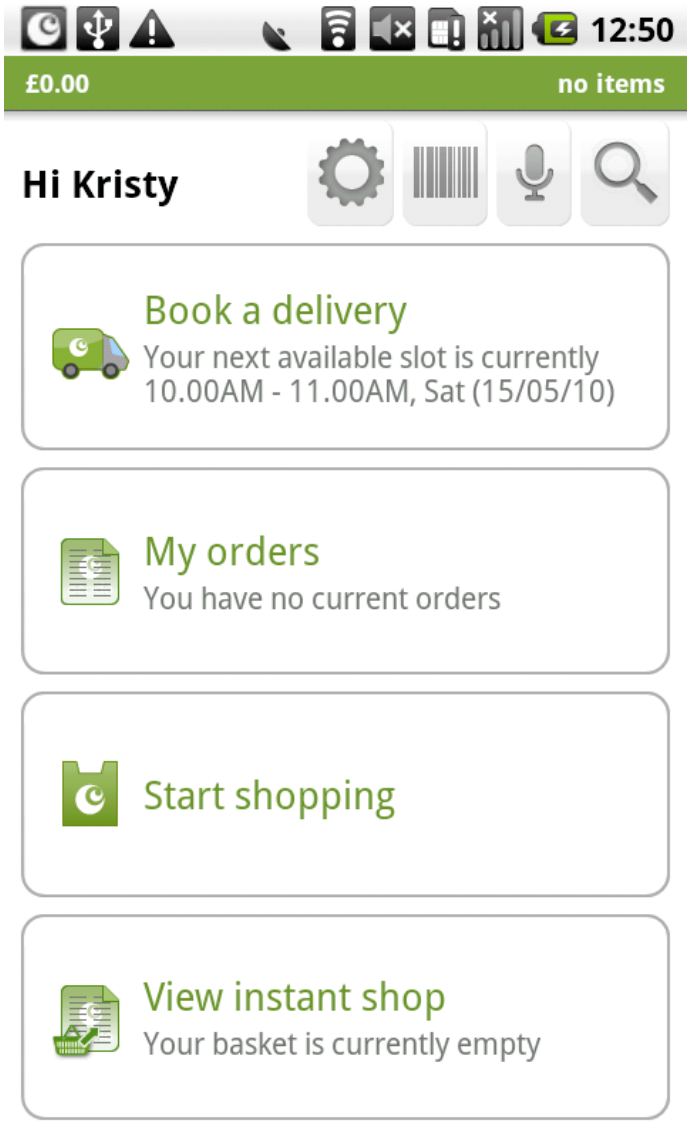
The Five Glorious Virtues

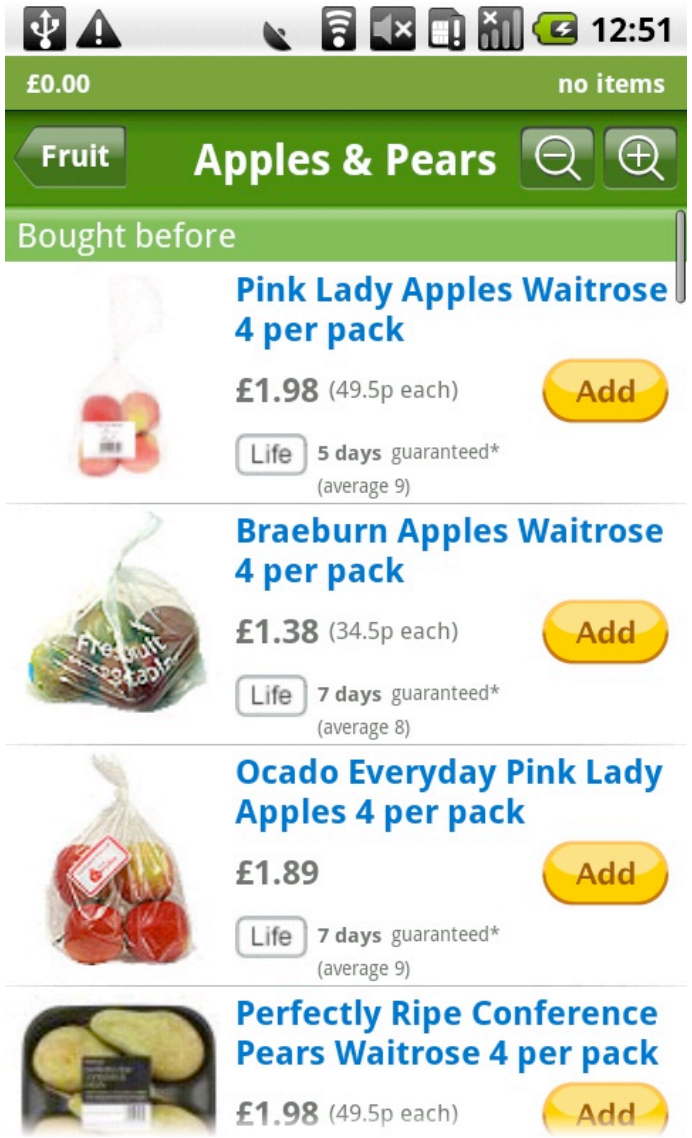


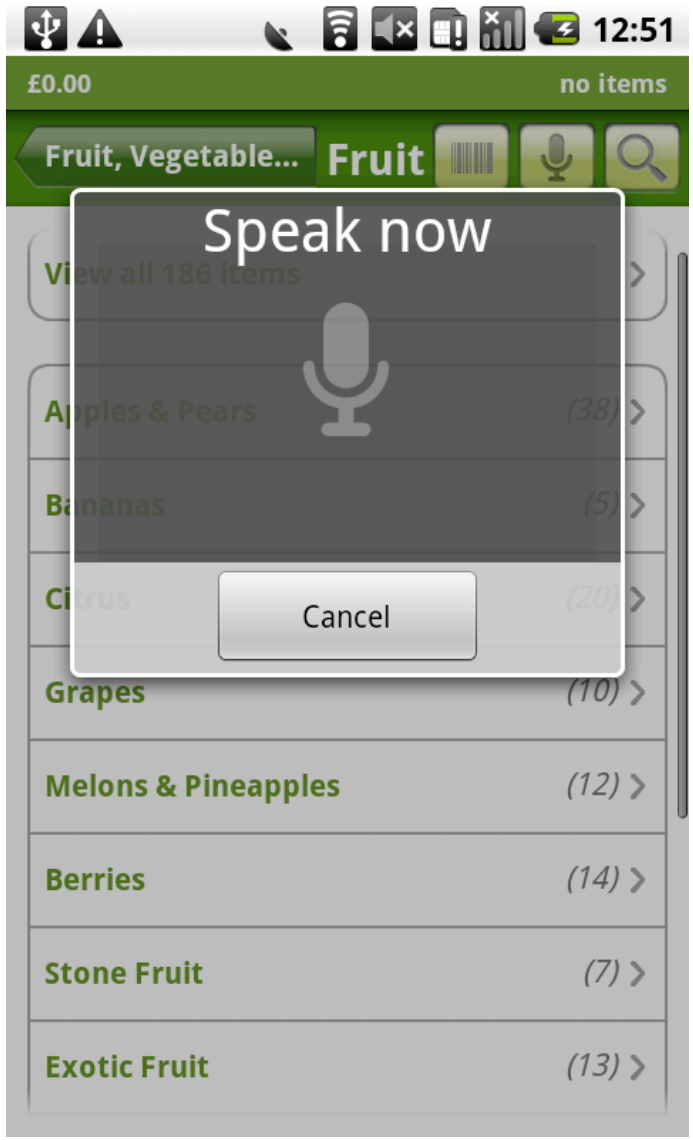
UTILITY

Be useful. Be interesting.

Ocado





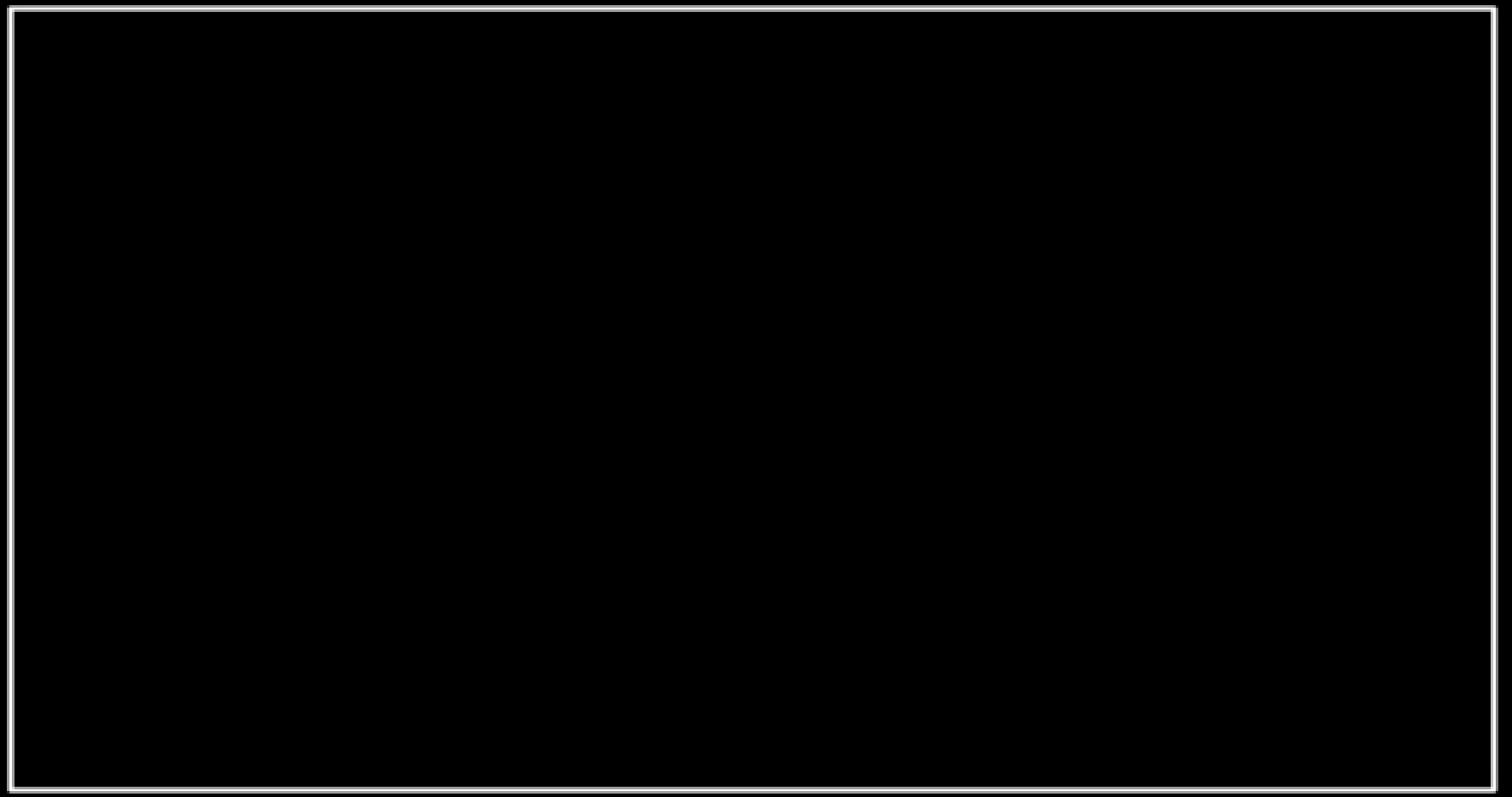


Google I/O 10

Utility & Entertainment

- Create an app that solves a problem
- Present information in the most useful way possible
- Create games that are ground breaking and compelling

The Five Glorious Virtues



EPIC (NESS)

Be legendary

Epicnessicity

- Don't be satisfied with *good*
- Create unique solutions
- Invent new paradigms
- Leverage the hardware

Agenda

- The Five Deadly Sins
- The Five Glorious Virtues
- Two Practical Examples

Services and Alarms

Don't be "That Guy"

- Let the runtime kill your background Service
- Let your users kill your foreground Service
- Kill your own Service
- Don't even start your Service
- Do you even need a Service?
- Use Alarms
- Use inexact Alarms

Let the Runtime Kill Your Service
START_NOT_STICKY

- Services that perform a single action
- Action is performed regularly (polling!)
- Reduces resource contention

```
@Override
public int onStartCommand(Intent intent, int flags,
                          int startId) {
  // Start an ASyncTask to do work
  return Service.START_NOT_STICKY;
}
```

Don't be "That Guy"

- Let the runtime kill your background Service
- Let your users kill your foreground Service
- Kill your own Service
- Don't even start your Service
- Do you even need a Service?
- Use Alarms and Intent Receivers
- Use inexact Alarms

Let Your Users Kill Your Service

- Only use a foreground Service if it's necessary
  - User is directly interacting with it
  - Music playback
- Provide clear options for disabling your Service
- Always use an ongoing notification
- Once it's been stopped, don't restart it without user action!

Don't be "That Guy"

- Let the runtime kill your background Service
- Let your users kill your foreground Service
- Kill your own Service
- Don't even start your Service
- Do you even need a Service?
- Use Alarms and Intent Receivers
- Use inexact Alarms

Kill Your Own Service

- Services should only be running when needed
- Complete a task, then kill the Service

```
stopSelf();
```

Kill Your Own Service

```
@Override
public int onStartCommand(Intent i, int f, int sId) {
  myTask.execute();
  return Service.START_NOT_STICKY;
}
AsyncTask<Void, Void, Void> myTask = new AsyncTask<Void, Void, Void>() {
  @Override
  protected Void doInBackground(Void... arg0) {
    // TODO Execute Task
    return null;
  }

  @Override
  protected void onPostExecute(Void result) {
    stopSelf();
  }
};
```

Don't be "That Guy"

- Let the runtime kill your background Service
- Let your users kill your foreground Service
- Kill your own Service
- Don't even start your Service
- Do you even need a Service?
- Use Alarms and Intent Receivers
- Use inexact Alarms

Don't be "That Guy"

- Let the runtime kill your background Service
- Let your users kill your foreground Service
- Kill your own Service
- Don't even start your Service
- Do you even need a Service?
- Use Alarms and Intent Receivers
- Use inexact Alarms

Don't be "That Guy"

- Let the runtime kill your background Service
- Let your users kill your foreground Service
- Kill your own Service
- Don't even start your Service
- Do you even need a Service?
- Use Alarms and Intent Receivers
- Use inexact Alarms

Alarms and Intent Receivers

- Schedule updates and polling
- Listen for system or application events
- No Service. No Activity. No running Application.

Intent Receivers

```
<receiver android:name="MyReceiver">
  <intent-filter>
    <action android:name="REFRESH_THIS" />
  </intent-filter>
</receiver>
```

```
public class MyReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent i) {
    Intent ss = new Intent(context, MyService.class);
    context.startService(ss);
  }
}
```

Alarms

```
String alarm = Context.ALARM_SERVICE;
AlarmManager am;
am = (AlarmManager)getSystemService(alarm);

Intent intent = new Intent("REFRESH_THIS");
PendingIntent op;
op = PendingIntent.getBroadcast(this, 0, intent, 0);

int type = AlarmManager.ELAPSED_REALTIME_WAKEUP;
long interval = AlarmManager.INTERVAL_FIFTEEN_MINUTES;
long triggerTime = SystemClock.elapsedRealtime() +
                   interval;

am.setRepeating(type, triggerTime, interval, op);
```

Don't be "That Guy"

- Let the runtime kill your background Service
- Let your users kill your foreground Service
- Kill your own Service
- Don't even start your Service
- Do you even need a Service?
- Use Alarms and Intent Receivers
- Use inexact Alarms

Inexact Alarms

- All the Alarm goodness
- Now with less battery drain!

```
int type = AlarmManager.ELAPSED_REALTIME_WAKEUP;
long interval = AlarmManager.INTERVAL_FIFTEEN_MINUTES;
long triggerTime = SystemClock.elapsedRealtime() +
                   interval;

am.setInexactRepeating(type, triggerTime,
                       interval, op);
```

Location Based Services

Location Based Services

```
String serviceName = Context.LOCATION_SERVICE;
lm = LocationManager)getSystemService(serviceName);

LocationListener l = new LocationListener() {
  public void onLocationChanged(Location location) {
    // TODO Do stuff when location changes!
  }

  public void onProviderDisabled(String p) {}
  public void onProviderEnabled(String p) {}
  public void onStatusChanged(String p, int s, Bundle e) {}
};

lm.requestLocationUpdates("gps", 0, 0, l);
```

Location Based Services

- How often do you need updates?
- What happens if GPS or Wifi LBS is disabled?
- How accurate do you need to be?
- What is the impact on your battery life?
- What happens if location 'jumps'?

Restricting Updates

- Specify the minimum update frequency
- Specify the minimum update distance

```
int freq = 5 * 60000; // 5mins
int dist = 1000;      // 1000m

lm.requestLocationUpdates("gps", freq, dist, l);
```

Use Criteria to Select a Location Provider

```
Criteria criteria = new Criteria();
criteria.setPowerRequirement(Criteria.POWER_LOW);
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setSpeedRequired(false);
criteria.setCostAllowed(false);

String provider = lm.getBestProvider(criteria, true);

lm.requestLocationUpdates(provider, freq, dist, l);
```

Use Criteria to Select a Location Provider

- Specify your requirements and preferences
  - Allowable power drain
  - Required accuracy
  - Need for altitude, bearing, and speed
  - Can a cost be incurred?
- Find the best provider that meets your criteria
- Relax criteria (in order) until a provider is found
- Can limit to only active providers
- Can use to find all matching providers

Implement a Back-off Pattern

- Use multiple Location Listeners
  - Fine and coarse
  - High and low frequency / distance
- Remove listeners as accuracy improves

Location Based Services

```
lm.requestLocationUpdates(bestprovider, freq, dist, l);
lm.requestLocationUpdates(coarseProvider, 0,0, lcoarse);
lm.requestLocationUpdates(bestprovider, 0, 0, lbounce);
```

Location Based Services

```
private LocationListener lbounce = new LocationListener(){
  public void onLocationChanged(Location location) {
    runLocationUpdate();
    if (location.getAccuracy() < 10) {
      lm.removeUpdates(lbounce);
      lm.removeUpdates(lcoarse);
    }
  }
  [...]
};

private LocationListener lcoarse = new LocationListener(){
  public void onLocationChanged(Location location) {
    runLocationUpdate();
    lm.removeUpdates(lcoarse);
  }
  [...]
};
```

Summary

- Be good
- Don't be lazy
- Think about performance
- Think about the user experience
- Respect your users
- Respect the system
- Think BIG!

Questions?

- Twitter **@retomeier**
- Stack Overflow tag: android
- developer.android.com

- Use Wave for Q&A *right now*

http://bit.ly/ioandroid1


Google™
10