Software Engineering Large Practical: Android design principles and practice

Stephen Gilmore
(Stephen.Gilmore@ed.ac.uk)
School of Informatics

October 19, 2016

Contents

Android design principles

Android design is shaped by three overarching goals for users that apply to apps as well as the system at large.

1. Enchant me.
2. Simplify my life.
3. Make me amazing.

developer.android.com/design/get-started/creative-vision.html

1. Enchant me

- Delight me in surprising ways.
- Real objects are more fun than buttons and menus.
- Let me make it mine.
- Get to know me.

`developer.android.com/design/get-started/principles.html`

2. Simplify my life

- Keep it brief.
- Pictures are faster than words.
- Decide for me but let me have the final say.
- Only show what I need when I need it.
- I should always know where I am.
- Never lose my stuff.
- If it looks the same, it should act the same.
- Only interrupt me if it's important.

`developer.android.com/design/get-started/principles.html`

3. Make me amazing

- Give me tricks that work everywhere.
- It's not my fault.
- Sprinkle encouragement.
- Do the heavy lifting for me.
- Make important things fast.

developer.android.com/design/get-started/principles.html

Material design

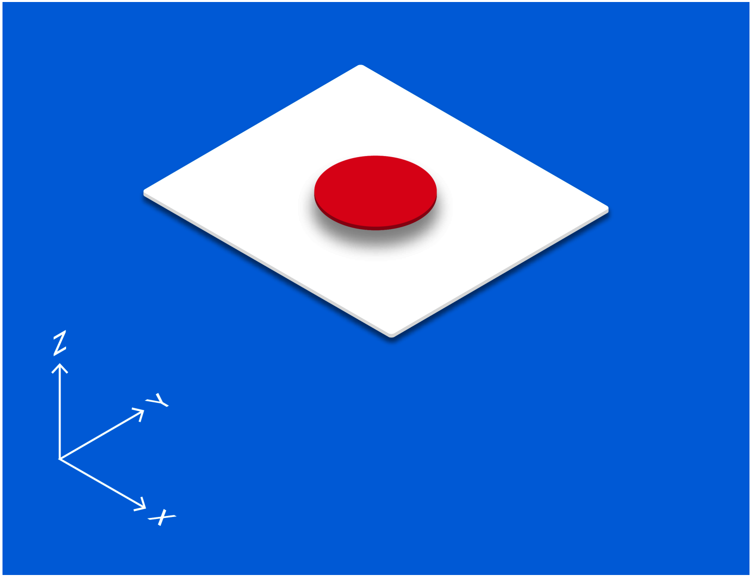
- The above design principles are intended to encourage developers to produce applications which tend to behave the same.
- Graphics may be used instead of menus and buttons; messages are brief and clear; errors are treated sympathetically; and so forth.
- In addition to this, Google User Experience Team wants applications to have a consistent look-and-feel and use the same visual language consistently.
- The design concepts and ideas which encourage this are expressed in the design guidance called *Material design* — `http://material.google.com/`.

Key features of material design

- Material design is a three-dimensional environment containing light, material, and shadows.
- All material objects have x, y, and z dimensions.
- Material objects have varying x and y dimensions (measured in dp — "dp" is *device-independent pixel*, pronounced "dip")
- All material objects are 1dp thick.
- All material objects have a single z-axis position.
- Key lights create directional shadows, and ambient light creates soft shadows.
- Shadows are created by the elevation difference between overlapping material.

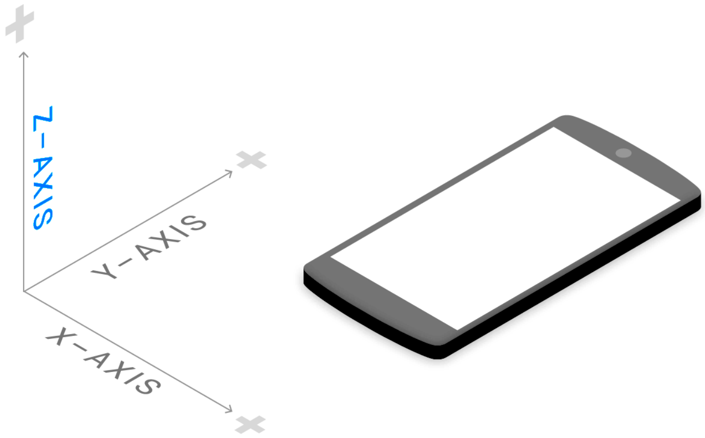
`material.google.com/material-design/environment.html`

Co-ordinates and shadows



material.google.com/material-design/environment.html

3D space with x, y, and z axes



material.google.com/material-design/environment.html

Material properties

- Material is solid.
- Input events cannot pass through material.
    - Input events only affect the foreground material.
- Multiple material elements cannot occupy the same point in space simultaneously.
- Material cannot pass through other material.
    - For example, one sheet of material cannot pass through another sheet of material when changing elevation.

`material.google.com/material-design/material-properties.html`

Object elevation

- All material objects, regardless of size, have a *resting elevation*, or default elevation that does not change.
  - If an object changes elevation, it should return to its resting elevation as soon as possible.
- Components maintain consistent resting elevations across apps.
- Components may have different resting elevations across platforms and devices, depending on the depth of the environment.
- Some component types have *responsive elevation*, meaning they change elevation in response to user input (e.g., normal, focused, and pressed) or system events.
- These elevation changes are consistently implemented using dynamic elevation offsets.

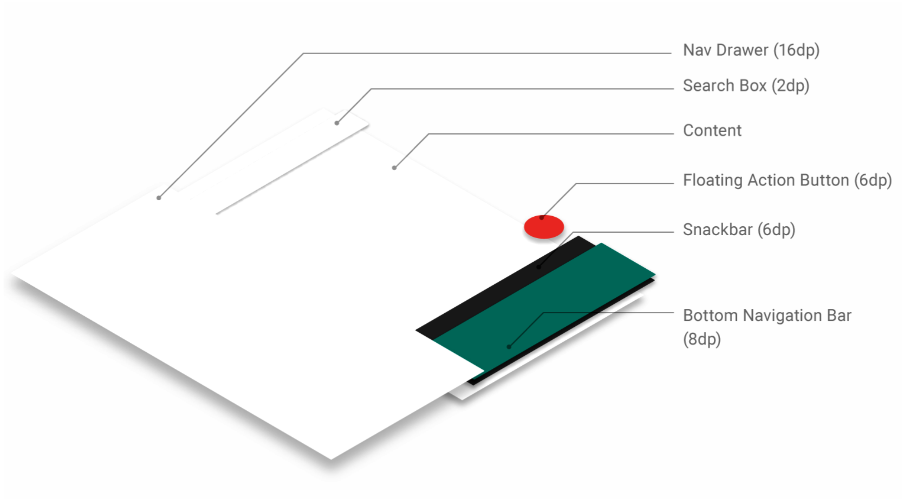
`material.google.com/material-design/elevation-shadows.html`

Components and elevations

Elevation (dp)	Component(s)
24	Dialog, Picker
16	Nav drawer, Right drawer, Modal bottom sheet
12	Floating action button (FAB — pressed)
9	Sub menu (+1dp for each sub menu)
8	Bottom navigation bar, Menu, Card (when picked up), Raised button (pressed state)
6	Floating action button (FAB — resting elevation), Snackbar
4	App Bar
3	Refresh indicator, Quick entry / Search bar (scrolled state)
2	Card (resting elevation), Raised button (resting elevation), Quick entry / Search bar (resting elevation)
1	Switch

`material.google.com/material-design/elevation-shadows.html`

Components and elevations examples



material.google.com/material-design/elevation-shadows.html

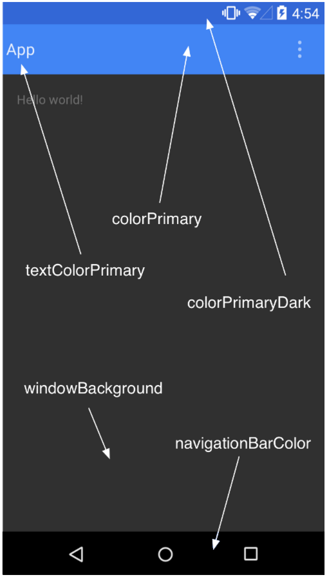
Adding material design to Android applications

In order to add material design to our Android app we first need to specify that we are using the Material style in our app's styles definition file and customise the theme's base colours as necessary.

```
<!-- res/values/styles.xml -->
<resources>
  <!-- inherit from the material theme -->
  <style name="AppTheme" parent="android:Theme.Material">
    <!-- Main theme colors -->
    <!-- your app branding color for the app bar -->
    <item name="android:colorPrimary">@color/primary</item>
    <!-- darker variant for status bar and contextual app bars -->
    <item name="android:colorPrimaryDark">@color/primary_dark</item>
    <!-- theme UI controls like checkboxes and text fields -->
    <item name="android:colorAccent">@color/accent</item>
  </style>
</resources>
```

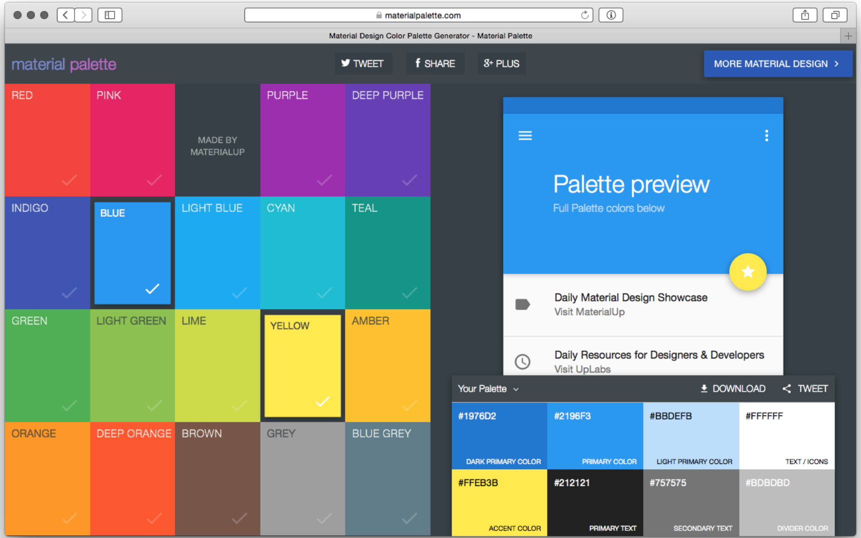
developer.android.com/training/material/get-started.html

Customisation options for the material theme



`https://developer.android.com/training/material/theme.html`

Material design colour palette generator



#1976D2 DARK PRIMARY COLOR	#2196F3 PRIMARY COLOR	#BBDEFB LIGHT PRIMARY COLOR	#FFFFFF TEXT / ICONS
#FFEB3B ACCENT COLOR	#212121 PRIMARY TEXT	#757575 SECONDARY TEXT	#BDBDBD DIVIDER COLOR

https://www.materialpalette.com/

XML download offered

```xml
<!-- Palette generated by Material Palette - materialpalette.com/blue/
      yellow -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="primary">#2196F3</color>
  <color name="primary_dark">#1976D2</color>
  <color name="primary_light">#BBDEFB</color>
  <color name="accent">#FFEB3B</color>
  <color name="primary_text">#212121</color>
  <color name="secondary_text">#757575</color>
  <color name="icons">#FFFFFF</color>
  <color name="divider">#BDBDBD</color>
</resources>
```

https://www.materialpalette.com/

Setting elevations of views

The elevation of a particular view in our app is set by specifying its `android:evelation` attribute.

```
<TextView
    android:id="@+id/my_textview"
    android:layout_width="wrap_content"
    android:layout_height ="wrap_content"
    android:text="@string/next"
    android:background="@color/white"
    android:elevation ="5dp"  />
```

developer.android.com/training/material/get-started.html

Setting up the app's ActivityBar

```java
public class MyActivity extends AppCompatActivity {
  // ...
}
```

```xml
<!-- In AndroidManifest.xml -->
<application
    android:theme="@style/Theme.AppCompat.Light.NoActionBar"
    />
```

```xml
<android.support.v7.widget.Toolbar
   android:id="@+id/my_toolbar"
   android:layout_width="match_parent"
   android:layout_height="?attr/actionBarSize"
   android:background="?attr/colorPrimary"
   android:elevation="4dp"
   android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
   app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

http://developer.android.com/training/appbar/setting-up.html

Method `onCreate`

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my);
    Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbar);
    setSupportActionBar(myToolbar);
    }
```

`http://developer.android.com/training/appbar/setting-up.html`

Adding action buttons

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <!-- "Mark Favourite", as action button if possible -->
    <item
        android:id="@+id/action_favourite"
        android:icon="@drawable/ic_favourite_black_48dp"
        android:title="@string/action_favourite"
        app:showAsAction="ifRoom"/>

    <!-- Settings, should always be in the overflow -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          app:showAsAction="never"/>

</menu>
```

https://developer.android.com/training/appbar/actions.html

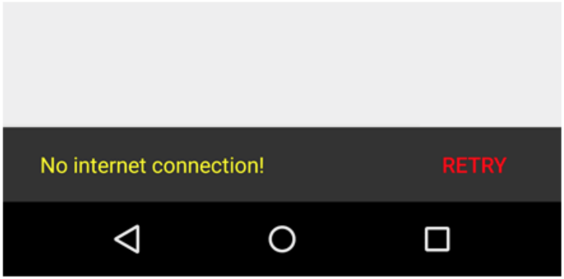
Responding to actions

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            // User chose the "Settings" item, show the app settings UI
            return true;

        case R.id.action_favourite:
            // User chose the "Favourite" action, mark the current item
            // as a favourite ...
            return true;

        default:
            // If we got here, the user's action was not recognized.
            // Invoke the superclass to handle it.
            return super.onOptionsItemSelected(item);

    }
}
```

https://developer.android.com/training/appbar/actions.html

Building and displaying a pop-up message

- You can use a *Snackbar* to display a brief message to the user.
- The message automatically goes away after a short period.
- A *Snackbar* is ideal for brief messages that the user doesn't necessarily need to act on.
- For example, an email app could use a *Snackbar* to tell the user that the app successfully archived an email message, or that there is no internet connection.


No internet connection!

https://developer.android.com/training/snackbar/showing.html

Attaching a SnackBar to a CoordinatorLayout

```
<android.support.design.widget.CoordinatorLayout
    android:id="@+id/myCoordinatorLayout"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Here are the existing layout elements, now wrapped in
              a CoordinatorLayout -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <!--...Toolbar, other layouts, other elements...-->

    </LinearLayout>

</android.support.design.widget.CoordinatorLayout>
```

https://developer.android.com/training/snackbar/showing.html

Showing the message to the user

```
Snackbar.make(findViewById(R.id.myCoordinatorLayout),
                                   R. string . email_archived ,
                                   Snackbar.LENGTH_SHORT)
            .show();
```

https://developer.android.com/training/snackbar/showing.html

Adding an action to a message

```
public class MyUndoListener implements View.OnClickListener{

    @Override
    public void onClick(View v) {
        // Code to undo the user's last action
    }

}
```

```
Snackbar mySnackbar =
            Snackbar.make(findViewById(R.id.myCoordinatorLayout),
                                    R.string.email_archived,
                                    Snackbar.LENGTH_SHORT);
mySnackbar.setAction(R.string.undo_string, new MyUndoListener());
mySnackbar.show();
```

https://developer.android.com/training/snackbar/action.html

Concluding remarks

- Material Design attempts to standardise user interface elements across apps and across platforms.
- There are many aspects which we have not covered here: fonts, text layout, animations, transitions, and others.