1 Motivation und Einführung

Single- vs. Cross-Plattform:
Single: Codebasis für jede Plattform
Cross: Shared Code + Platform Code
Native-, Hybrid-, Web-Apps:
Native: Plattform(NativeApp(Binary))
Hybrid: Plattform(NativeApp(HTML))
Web: Plattform(Web Browser(HTML))
Vorteil Native Apps:

- Voller Funktionsumfang
- keine Tools/Einschränkungen von Drittanbietern

2 Grundkonzepte

Apps bestehen aus lose gekoppelten, wiederverwendbaren Komponenten (Activities, Content Providers, Services & Broadcast receivers).
Android hat die Kontrolle über ausgeführte Apps:

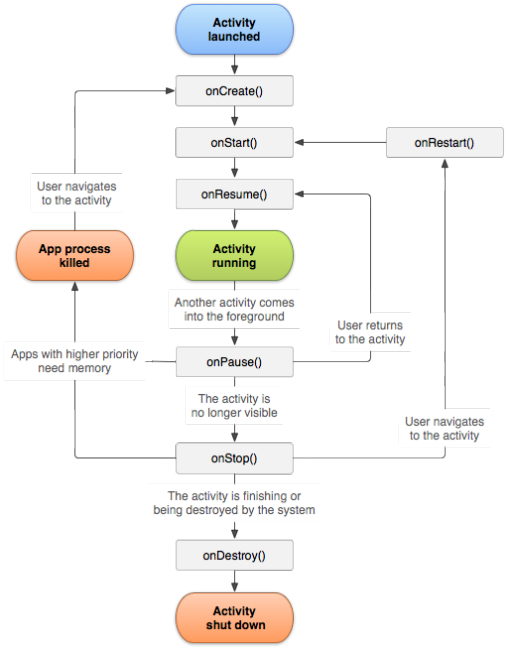
- Verwaltung des Lebenszyklus
- Kommunikation zwischen Komponenten
- Terminierung bei Bedarf (z.B. Speicherknappheit)

2.1 Activities

Beim App-Start wird die Main Activity von Android erzeugt und ausgeführt. Activities besitzen eine grafische Obefläche und verarbeiten Benutzereingaben.

```
public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

Activity Lebenszyklus & Zustände:
Android ruft beim Zustandwechsel Callback-Methoden auf der Activity auf. Diese Methoden können überschrieben werden.



2.1.1 Anwendungsfälle

- Erzeugung des GUI: onCreate()
- Datensicherun: onPause für schnelle Operationen, ansonsten onStop()
- Dienste wie Lokalisierung aktivieren/deaktivieren: onResume() und onPause()
- Zustand des GUI erhalten, z.B. bei Rotation: onSaveInstanceState() und onRestoreInstanceState()

2.2 Intents

- Die Kommunikation zwischen Komponenten erfolgt über Intents (Absicht, Vorhaben)
- Zwei Arten von Intents:
  - Explizit: Aufruf einer definierten Komponente (typischerweise für Komponenten der eigenen App)
  - Implizit: Aufruf einer passenden Komponente (typischerweise für Komponenten aus anderen Apps)
- Apps können sich im Android Manifest mit Intent Filters auf implizite Intents registrieren
- Intents werden stets von Android verarbeitet

```
// Expliziter Intent
Intent secondActivityIntent = new Intent(this,
    SecondActivity.class);
startActivity(secondActivityIntent);
// Impliziter Intent
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.setType("text/plain");
sendIntent.putExtra(Intent.EXTRA_TEXT, "Hey!");
startActivity(sendIntent);
```

2.2.1 Beispiel

```
Button button = findViewById(R.id.buttonNavigate);
button.setOnClickListener(v -> {
   //Explizit
   Intent secondActivityIntent = new Intent(this,
       SecondActivity.class);
   startActivity(secondActivityIntent);
   //Implizit
   Intent intent = new Intent(Intent.ACTION_VIEW, Uri.
       parse("http://www.ost.ch"));
   startActivity(intent);
});
```

2.3 Intents mit Parametern

Zusätzliche Parameter können als Key-Value Paar (Bundle) mit putExtra()/putExtras() übergeben werden.

```
// MainActivity.java
Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra("myKey", 42);
startActivity(intent);
// SecondActivity.java
Intent intent = this.getIntent();
String parameter = intent.getStringExtra("key");
```

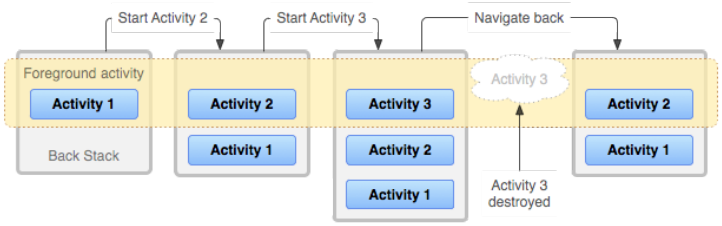
2.3.1 Hinweise

Mit Intents startet man andere Activities.
→ Ohne Rückgabewert: startActivity(Intent)
→ Mit Rückgabewert: startActivityForResult(Intent, int)
Implizite Intents müssen nicht immer einen Empfänger haben.
→ Darum immer überprüfen ob Intent einen Empfänger hat:

```
//MainActivity.java
if(intent.resolveActivity(getPackageManager()) != null){
   startActivity(intent);
}
//AndroidManifest.xml
<uses-permission android:name="android.permission.
    QUERY_ALL_PACKAGES" />
```

2.4 Back Stack (Task)

- Activities werden im Back-Stack verwaltet
- Activities eines Stacks können zu verschiedenen Apps gehören
- Dieselbe Activity kann mehrfach im selben Stack enthalten sein



Ein Back Stack wird auch Task genannt. Android verwaltet die Ausführung von Tasks. Bei Bedarf können Activities in neuen Tasks gestartet werden.

2.5 Tasks, Prozesse und Threads

- Alle Teile eines Apps werden in einer APK-Datei ausgeliefert
- Jedes APK wird mit einem eigenen Linux User installiert (Sandbox)
- Jedes APK wird in einem eigenen Linux Prozess ausgeführt
- Jeder Prozess hat mindestens einen Thread (Main Thread)

2.5.1 Main-Thread

- Automatisch erzeugt beim Start einer Applikation
- Blockierung des Main Threads führt zum ANR-Screen (Application Not Responding)
- Langlaufende Operationen immer in eigenen Threads ausführen (Runnable)
- **Achtung:** Nur der Main Thread darf das GUI aktualisieren, sonst Exception

2.6 GUI

Das GUI kann auf zwei Arten erstellt werden: Deklarativ (Beschreibung in XML) und Imperativ (Beschreibung im Quellcode).

2.7 Event Handling

Listener reagieren auf GUI-Ereignisse und werden bei GUI-Objekt registriert

```
final TextView textView = this.findViewById(R.id.
    text_example);
Button button = this.findViewById(R.id.button_example);
button.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      textView.setText("Button pressed");
   }
});
// Lambda
button.setOnClickListener(v -> { ... });
// XML
android:onClick="onExampleButtonClicked"
public void onExampleButtonClicked(View view)
```

2.8 Resources

Alle Dateien, die keinen Code enthalten, werden als Resources bezeichnet. colors.xml für Farbwerte, dimens.xml für Dimensionen, strings.xml für Texte, styles.xml für Styles. Veränderliche Werte immer in passenden Files definieren und referenzieren. Der Zugriff erfolgt jeweils über die Resource ID. → Zugriff via **R-Klasse**

2.9 Dimensionen

Android erlaubt die Verwendung folgender Dimensionen:

- dp: Density-independent Pixels
- sp: Scale-independent Pixels
- px: Pixel
- pt: Punkte (1/72 eines physikalischen)
- in: Inch
- mm: Millimeter

Empfehlung: Für Schriften immer in sp, Alles andere in dp

2.10 Qualifier

Resources können in unterschiedlichen Varianten hinterlegt werden:

- Texte für verschiedenen Sprachen
- Bilder für verschiedenen Auflösungen
- Layouts für unterschiedliche Gerätetypen

2.10.1 Mehrsprachigkeit

Kein Hardcoded Text sondern über String resource file. Mehrere values Ordner (values_en, etc.) mit strings.xml Dateien anlegen.

2.11 App Manifest

Das AndroidManifest.xml enthält essenzielle Informationen zur App.

- ID, Name, Version und Logo
- Enthaltene Komponenten
- Hard- und Softwareanforderungen
- Benötigte Berechtigungen

2.11.1 Application ID und Version

package: Eindeutige Identifikation der App, Definiert Namespace, Reversed Internet Domain Format (ch.ost.rj.helloworld)
versionName: Ein menschenlesbarer String, Typischerweise Semantic Versioning
versionCode: Ein positiver Integer für interne Verwendung, Je höher die Zahl, desto "neuer"die App, Unterschiedliche Ansätze zur Inkrementierung

2.11.2 Application-Element

- Parent der Komponenten ist der Application-Knoten
- Application ist auch eine Klasse, die den globalen Zustand der App hält
- Eigene Ableitung von Application kann registriert werden
- Application enthält LifeCycle-Methoden, die überschrieben werden können

2.11.3 API Level

- minSdkVersion gibt an, welche Version das Gerät mindestens haben muss
- maxSdkVersion gibt an, welche Version das Gerät maximal haben darf
- targetSdkVersion ist die Version, welche die App bei der Ausführung verwendet
- compileSdkVersion gibt an, mit welcher API das App kompiliert wird

3 GUI Programmierung

3.1 View und ViewGroup

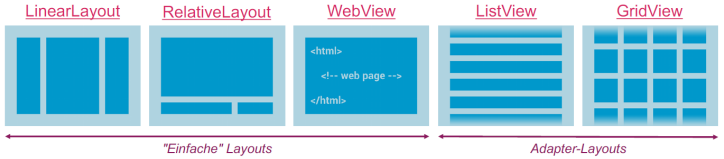
View ist die Basisklasse aller GUI Elemente. Es belegt einen rechteckigen Bereich und kümmert sich um die Darstellung und Event Verarbeitung. Die Ableitung ViewGroup enthält View-Objekte (Parent-Child Beziehung). ViewGroup-Klassen ordnen ihre Kinder nach einem Muster an, sind strukturierend und unsichterbar. Werden auch Layouts oder Container genannt.

3.2 Layouts Allgemein

Im onCreate der Activity wird das layout geladen:

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

Layouts Übersicht:



Es gibt noch weitere Layouts und eigene können auch definiert werden. Layouts können beliebig verschachtelt werden, jedoch mit negativem Einfluss auf die Performance. → Am besten Flache, breite Hierarchie

3.2.1 Layout-Parameter

Verschachtelte Child View teilt Parent mit, wie sie angeordnet werden wollen. Child setzt auf sich selber diese Parameter.

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:gravity="center">
</LinearLayout>
```

Mögliche Werte: match_parent (so gross wie möglich), wrap_content (so klein wie Content) und Zahl (unüblich, meist in dp).

3.2.2 padding und Margin

Padding wird auf sich selbst gesetzt. Margin wird dem Parent übergeben, da ein Child nicht einfach den Platz dem Parent wegnehmen kann.

```
android:padding="20dp"
android:layout_margin="20dp"
```

3.3 Linear Layout

Vertikal oder horizontal angeordnet. Mit layout_weight kann die Grösse beeinflusst werden. → Verwendung in Kombination mit wrap_content

```
android:orientation="vertical"
android:orientation="horizontal"

// Beispiel mit weight:
| | |       | // Links: minimaler Platz (kein weight)
| | |       | // Mitte: android:layout_weight="1"
| | |       | // Rechts: android:layout_weight="3"
```

3.4 Frame Layout

Kinder werden übereinander angeordnet. z.B. Live-Kamerabild mit Auslöse-Button und Hilfslinien. → Anpassung der "Höhe"über dem Bild: Standardmässig gilt die Reihenfolge im XML. Manuelle Anpassung mit android:translationZ möglich

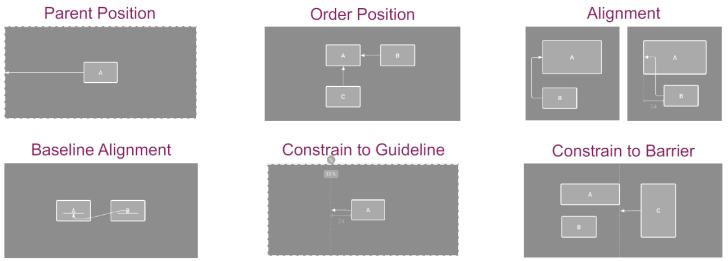
3.5 Relative Layout

Kinder werden relativ zueinander angeordnet. Identifizierung der anderen Kinder über Resource IDs. Mächtig, kann als effizienter Ersatz für verschachtelte Linear Layouts dienen.

```
// Beispiele
android:layout_alignParentTop="true"
android:layout_toStartOf="@id/..."
android:layout_alignStart="@id/..."
```

3.6 Constraint Layout

Das modernste und flexibelste Layout. Ist Teil von Jetpack/AndroidX. Grundidee: Definieren von Beziehungen zwischen Views. Pro View muss mindestens eine horizontale und vertikale Einschränkung definiert werden.



3.7 Widgets

Namespace: android.widget
Basisklasse: View

3.7.1 TextView und ImageView

TextView zur Anzeige von Text:

```
<TextView
   ...
   android:text="TextView"
   android:textSize="20sp"
   android:textStyle="bold"
   android:typeface="monospace"
   android:textColor="@android:color/white"
   android:background="@color/colorPrimaryDark"
   android:drawableEnd="@drawable/ic_emoji"
   android:drawableTint="@android:color/white" />
```

ImageView zur Anzeige von Bildern:

```
<ImageView
   ...
   android:layout_height="80dp"
   android:src="@drawable/ic_emoji"
   android:scaleType="fitCenter"
   android:tint="@color/colorPrimaryDark" />
```

3.7.2 Button und ImageButton

Buttons. Lösen via Listener Aktionen aus. Ableitung von TextView bzw. ImageView.

```
<Button
   ...
   android:text="Button"
   android:drawableEnd="@drawable/ic_emoji"
   android:drawableTint="@color/colorPrimary"/>
```

3.7.3 EditText

EditText dient als Eingabefeld für Texte und Zahlen. android:inputType beeinflusst Verhalten und aussehen (auch Keyboard).

```
android:inputType="textPassword"
android:inputType="date"
android:inputType="textMultiLine"
// Auch kombinierbar
android:inputType="textCapCharacters|textAutoCorrect"
```

Bei der Texteingabe kann auf Ereignisse reagiert werden. Dazu muss ein TextWatcher als Listener registriert werden. Folgende 3 Methoden können überschrieben werden:

- beforeTextChanged
- onTextChanged
- afterTextChanged

```
myEditText.addTextChangedListener(new TextWatcher() {
   public void afterTextChanged(Editable editable) {
      if (editable.length() < 8) {
         passwordInput.setError("Passwort zu kurz.");
      }
   }
})
```

Weitere, häufig verwendete Widgets:
Checkbox, Picker, Floating Action Button, Radio Buttons, Seek Bar, Spinner

3.7.4 UI-Elemente ohne XML

Werden direkt aus dem Code heraus erzeugt. Anpassbarkeit ist oft eingeschränkt (Farben, Texte, etc.)
Toasts: Einfache Rückmeldung zu Vorgang (Pop Up)
Snackbars: Wie Toast, aber mit Interaktion.
Dialoge: Erzwingen Aktion von Benutzer
Notification: Mitteilung ausserhalb aktiver Nutzung. NotificationCompat in AndroidX verwenden.
Manus: Existieren in verschiedenen Varianten. Options Menu, Contextual Menu, Popup Menu. → Wird generell als Resource in res/menu definiert.

3.8 ScrollView

- Ist ein spezielles Layout mit nur einem Kind-Element
- Erlaubt das vertikale Scrolling des Inhalts
- Horizontal nur mit HorizontalScrollView
- Alternative in AndroidX: NestedScrollView (erlaubt beide Richtungen)

```
<ScrollView
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <!-- Genau ein Kind hier -->
</ScrollView>
```

3.9 ListView und ArrayAdapter

Gut für Darstellung von Collections. Ein Adapter vermittelt zwischen der Darstellung und der Datenquelle.

```
// main_activity.xml
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android=" ... "
   android:id="@+id/list_example"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
</ListView>

// MainActivity.java
setContentView(R.layout.activity_main);
String[] data = new String[] { ... };
ArrayAdapter<String> adapter = new ArrayAdapter<>(
   this,
   android.R.layout.simple_list_item_1,
   android.R.id.text1,
   data);
ListView listView = findViewById(R.id.list_example);
listView.setAdapter(adapter);
```

3.10 RecyclerView

Die RecyclerView ist eine moderne Alternative zu ListView und GridView. Ist Teil von AndroidX und erzwingt die Verwendung von View Holdern.

```
// main_activity.xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.recyclerview.widget.RecyclerView
   android:id="@+id/recycler_view"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
</androidx.recyclerview.widget.RecyclerView>

// MainActivity.java
setContentView(R.layout.activity_recyclerview);
RecyclerView recyclerView = findViewById(R.id.
    recycler_view);
RecyclerView.LayoutManager layoutManager;
layoutManager = new LinearLayoutManager(this);
recyclerView.setLayoutManager(layoutManager);
ArrayList<User> data = UserManager.getUsers();
UsersAdapter adapter = new UsersAdapter(data);
recyclerView.setAdapter(adapter);

// UsersAdapter.java
public class UsersAdapter extends RecyclerView.Adapter<
    ViewHolder> {
   private ArrayList<User> users;

   @Override
   public ViewHolder onCreateViewHolder(ViewGroup parent
       , int vt) {
      Context context = parent.getContext();
      LayoutInflater inflater = LayoutInflater.from(
          context);

      View view = inflater.inflate(
         android.R.layout.simple_list_item_2,
         parent,
         false);

      return new ViewHolder(
         view,
         view.findViewById(android.R.id.text1),
         view.findViewById(android.R.id.text2));
   }
   @Override
   public void onBindViewHolder(ViewHolder holder, int
       position) {
      User user = this.users.get(position);
      holder.text1.setText(user.name);
      holder.text2.setText(user.age + " Jahre");
   }
   @Override
   public int getItemCount() {
      return this.users.size();
   }
}
```

4 Strukturierung, Material Design und Styling

4.1 Fragments

Activities können nicht kombiniert werden, Fragments aber schon. Ein Fragment ist ein modularer Teil in einer Activity mit eigenem Lebenszyklus.
Zusätzliche Callbacks gegenüber Activity:

- **onAttach:** Fragment an Activity angehängt
- **onCreateView:** UI des Fragments erstellen
- **onActivityCreated:** Activity wurde erzeugt
- **onDestroyView:** Gegenstück zu onCreateView
- **onDetach:** Gegenstück zu onAttach

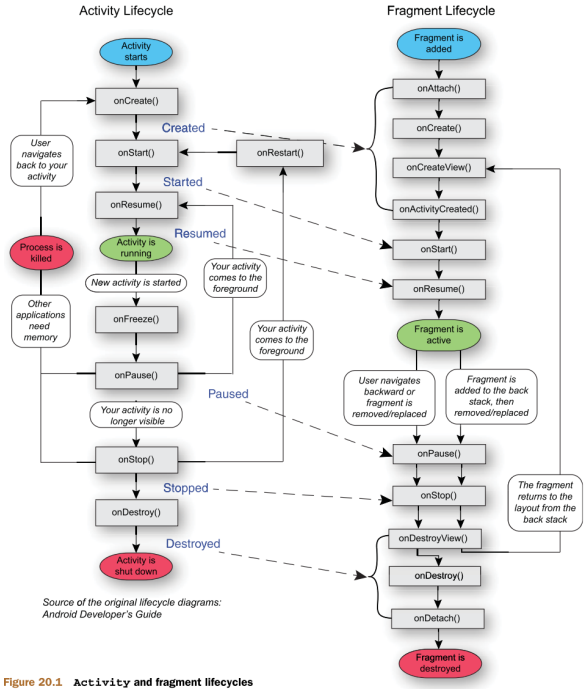


Figure 20.1 **Activity and fragment lifecycles**

4.1.1 Dynamische Einbindung

```
// activity_main.xml
<LinearLayout xmlns:android="( ... )"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <FrameLayout android:id="@+id/main_fragment_container
       "
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>

//MainActivity.java
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      FragmentManager mgr = getSupportFragmentManager();
      FragmentTransaction trans =mgr.beginTransaction();

      OutputFragment fragment = new OutputFragment();
      trans.add(R.id.main_fragment_container, fragment);
      trans.commit();
   }
}
```

4.1.2 Activity-Fragment Kommunikation

Fragments sollen wiederverwendbar sein. Einbindung in verschiedene Activities und keine direkten Abhängigkeiten zu Activities haben.
Best Practices:
Activity → Fragment: Parameter und Methoden
Activity ← Fragment: Callback-Interfaces

4.1.3 parameter und Methoden

```
// Vorher:
OutputFragment fragment = new OutputFragment();

// Mit Parameter:
OutputFragment fragment;
fragment = OutputFragment.create("Initial Value");
// In fragment.java: public static OutputFragment create
    (String text) { ... }

// Zusätzlich: Event-Listener
Button button = findViewById(R.id.main_button);
button.setOnClickListener(v -> {
   fragment.updateText("Updated value");
});
```

4.1.4 Callback-Interfaces

```
// Callback.java
public interface OutputFragmentCallback {
   void onTextTapped(String text);
}

// MainActivity.java
public class MainActivity extends AppCompatActivity
    implements OutputFragmentCallback {
   @Override
   public void onTextTapped(String text) {
      // Callback behandeln
   }
}

// OutputFragment.java
public class OutputFragment extends Fragment {
   private OutputFragmentCallback callback;
   @Override
   public void onAttach(Context context) {
      super.onAttach(context);
      try {
         callback = (OutputFragmentCallback) context;
      } catch (ClassCastException e) {
         throw new ClassCastException(" ...");
      }
   }
   @Override
   public View onCreateView( ... ) {
      View fragment = inflater.inflate( ... );
      textOutput = fragment.findViewById(R.id.
          output_text);
      textOutput.setOnClickListener(v -> {
         callback.onTextTapped("");
      });
      return fragment;
   }
}
```

4.1.5 Fragmente austauschen

Fragmente sind austauschbar und Übergänge können animiert werden. (XML-Beschreibung der Animation (res/anim))

```
fragmentManager.beginTransaction()
   .setCustomAnimations(
      R.anim.slide_in, // Einblendung neues Fragment
      R.anim.fade_out, // Ausblendung altes Fragment
      R.anim.fade_in, // Einblendung altes Fragment(Pop)
      R.anim.slide_out)//Ausblendung neues Fragment(Pop)
   .replace(R.id.main_fragment_container, newFragment)
   .addToBackStack(null)
   .commit();
});
```

4.1.6 Fragmente verschachteln

Sind verschachtelbar, gleiches Vorgehen bei der Einbindung. Unterschied: getChildFragmentManager() anstelle von getSupportFragmentManager()

4.2 Material Deisgn

Eine Designlanguage ist eine Hilfestellung für den Designprozess. Klare Regeln oder Empfehlungen zu Farbschema, Icons, Schriften, Abständen, etc. Martial Design ist die Design Language von Google.

4.2.1 Grundprinzipien

Material is the metaphor:

- Inspiriert von der physischen Welt
- Oberflächen erinnern an Papier und Tinte
- Materialien reflektieren Licht & werfen Schatten

Bold, graphic, intentional:

- Basiert auf Prinzipien von Print-Medien
- Hierarchie, Raster, Schriften, Farben, etc.

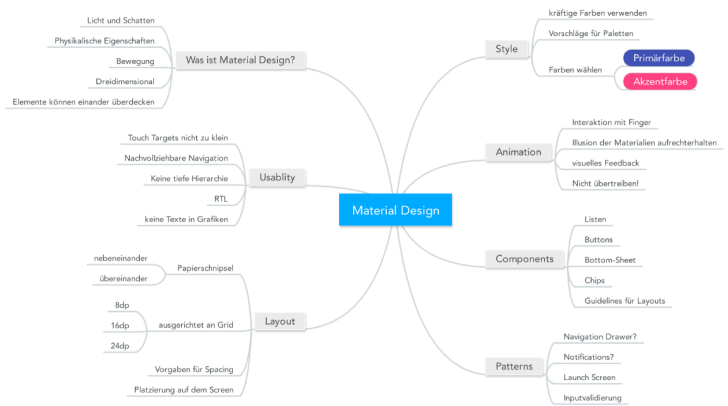
Motion provides meaning:

- Bewegung bedeutet Aktion
- Zurückhaltende, subtile Verwendung

4.2.2 Vorgaben

- Material ist immer 1dp dick ("Papier")
- Material wirft Schatten
- Material hat eine unendliche Auflösung
- Inhalt hat keine Dicke und ist Teil des Materials
- Material kann sich verändern
- Material kann sich bewegen

4.2.3 Zusammenfassung



4.3 Styling

Widgets werden über XML Attribute gestyled. Mögliche Probleme bei umfangreichen Apps wie Code-Duplizierung, Inkonsistenzen, Unübersichtlichkeit. Styles können wiederverwendbar gemacht werden.

4.3.1 Styles

Styles werden in der styles.xml resource definiert. Styles können aber auch mit der .Notation geerbt werden:

```
// layout.xml
<TextView
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Element 1"
   style="@style/HeaderText" />
<TextView
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Element 2"
   style="@style/HeaderText.Big" />

// styles.xml
<style name="HeaderText">
   <item name="android:textSize">24sp</item>
   <item name="android:background">#ff9999</item>
   <item name="android:padding">8dp</item>
   <item name="android:layout_margin">8dp</item>
   <item name="android:gravity">center</item>
</style>
<style name="HeaderText.Big">
   <item name="android:textSize">40sp</item>
</style>
```

4.3.2 Themes

Themes sind spezielle Styles, die für eine ganze App oder einzelne Activities gelten. Definition wie normale Styles in Resources:

```
// styles.xml
<resources>
   <style name="AppTheme" parent="">
      <item name="android:textViewStyle">@style/MyText</
          item>
   </style>
   <style name="MyText">
      <item name="android:textSize">24sp</item>
      <item name="android:background">#ff9999</item>
      <item name="android:padding">8dp</item>
      <item name="android:layout_margin">8dp</item>
      <item name="android:gravity">center</item>
   </style>
</resources>

// Anwenden: Manifest oder Activity:
// AndroidManifest.xml
<application ... android:theme="@style/AppTheme">
   <activity ... android:theme="@style/AnotherAppTheme"
       />
</application>
// MainActivity.java
@Override
protected void onCreate(Bundle savedInstanceState) {
   setTheme(R.style.AnotherAppTheme);
   setContentView(R.layout.activity_styling);
}
```

4.3.3 Material Components Library

Damit erhält man, ausser den Themes, auch Zugriff auf Material Design-Widgets