

Natural User Interfaces - HS20

Pascal Brunner - brunnpa7

Inhaltsverzeichnis

| | |
|---|-----------|
| 1 From GUIs to NUIs | 4 |
| 1.0.1 Text-based UIs | 4 |
| 1.0.2 Klassisches GUI | 4 |
| 1.0.3 3 Prinzipien der direkten Manipulation | 5 |
| 1.0.4 Vor- / Nachteile GUIs | 5 |
| 1.0.5 Charakteristik | 5 |
| 1.0.6 Main issues of the new modalities | 6 |
| 1.0.7 Ziele | 6 |
| 1.0.8 Die drei wichtigsten Anforderungen rund um die Usability (auch Usability Engineering genannt) | 6 |
| 1.0.9 Die sieben wichtigsten Charakteristiken | 7 |
| 1.0.10 Begrifflichkeiten | 8 |
| 1.0.11 UCD Prozess | 9 |
| 2 UCD: Conceptual Design and Prototype | 12 |
| 2.1 Ablauf eines Conceptual Design | 12 |
| 2.1.1 Interaction Styles | 12 |
| 2.1.2 Application Posture | 13 |
| 2.1.3 Interaction Pattern | 13 |
| 2.1.4 Metaphern | 14 |
| 2.1.5 Daten-Objekte identifizieren | 14 |
| 2.1.6 Function Blocks and Screens | 15 |
| 2.1.7 Touch Interaction Patterns | 15 |
| 2.2 Prototype | 17 |
| 2.2.1 Low Fidelity | 17 |
| 2.2.2 Evaluieren | 17 |
| 3 Usability Guidelines | 18 |
| 3.1 8 goldige Regeln von Ben Shneiderman | 18 |
| 3.2 10 Heuristics of Jacob Nielsen | 18 |
| 3.3 Touch interfaces | 19 |
| 3.3.1 Zusammenfassend | 20 |
| 3.4 Styleguides | 20 |
| 4 Detailed Design of NUIs | 21 |
| 4.1 Affordance | 21 |
| 4.1.1 Affordance für Gesten | 21 |
| 5 App Design | 22 |
| 5.1 Prozess | 22 |
| 5.2 Research | 22 |
| 5.2.1 Design Consideration | 22 |
| 5.3 Design Process | 23 |
| 5.4 Design for Android | 23 |

| | | |
|-----------|--|-----------|
| 5.4.1 | Alternate Layouts | 23 |
| 5.4.2 | px, dp, sp | 23 |
| 5.4.3 | Android Desnity Buckets | 23 |
| 5.4.4 | 9-Patch Files | 23 |
| 5.4.5 | Was ist klickbar? | 24 |
| 5.5 | Material Design | 24 |
| 5.5.1 | Basic Structure | 24 |
| 5.5.2 | Navigation in Android | 24 |
| 5.6 | Design Empfehlungen | 25 |
| 6 | App Design mit Kotlin | 26 |
| 6.1 | Aufbau | 26 |
| 6.2 | App Strukturierung | 26 |
| 7 | Android App Development II | 27 |
| 7.1 | Threading | 27 |
| 7.2 | Fast lists | 27 |
| 7.3 | Concurrent Programming | 27 |
| 8 | Patterns for UIs | 28 |
| 8.1 | Observer Patterns | 28 |
| 8.2 | Model View Controller and Friends | 29 |
| 8.2.1 | Model View Controller | 29 |
| 8.2.2 | MVP | 29 |
| 8.2.3 | Model View ViewModel | 30 |
| 8.3 | Chain of Responsibility | 30 |
| 8.4 | Composite | 30 |
| 9 | User and Domain Research | 31 |
| 9.1 | User | 31 |
| 9.1.1 | Interview | 33 |
| 10 | Menschliche Wahrnehmung | 34 |
| 10.1 | Das Menschliche Wahrnehmungssystem | 34 |
| 10.1.1 | Sehen | 34 |
| 10.1.2 | Hören | 36 |
| 10.2 | menschlicher Verstand | 36 |
| 10.2.1 | Kurzzeitgedächtnis | 36 |
| 10.2.2 | Langzeitgedächtnis | 36 |
| 10.2.3 | Mental Model | 37 |
| 10.2.4 | Erfahrung | 38 |
| 10.2.5 | Lerntypen | 38 |
| 10.3 | Menschliche Erwartungen | 38 |
| 10.4 | Menschliches Handeln | 38 |
| 11 | Web Accessability | 39 |
| 11.1 | WCAG | 39 |
| 11.2 | UAAG | 39 |
| 11.3 | ATAG | 39 |
| 11.4 | ARIA | 40 |

| | |
|---|-----------|
| 12 Code-Snippets Android | 41 |
| 12.1 Material Design as Library | 41 |
| 12.2 item Layout | 41 |
| 12.3 Kotlin Main Activity | 42 |
| 12.4 Activity Lifecycle | 42 |
| 12.4.1 onCreate() | 44 |
| 12.4.2 onResume() | 44 |
| 12.4.3 onPause() | 45 |
| 12.4.4 onStop() | 45 |
| 12.4.5 Preserving Instance State | 46 |
| 12.5 Intents | 46 |
| 12.5.1 Creating Intents | 46 |
| 12.5.2 Passing Arguments | 46 |
| 12.5.3 Real-Life Example | 47 |
| 12.6 Application Object aka Single Thread | 47 |
| 12.7 Handling Events | 47 |
| 12.8 Accessing Ressources | 48 |
| 12.9 System Ressources | 48 |
| 12.10 UI Thread | 49 |
| 12.11 Callback to UI Thread | 49 |
| 12.12 Adapter-Class - Fast-List | 50 |
| 12.12.1 Adapter Class extends Viewholder | 50 |
| 12.12.2 viewholder | 50 |
| 12.12.3 onBindViewHolder | 50 |
| 12.12.4 Java | 50 |
| 12.13 PresentationModel | 51 |

Kapitel 1

From GUIs to NUIs

Graphic User Interface (GUI)

1.0.1 Text-based UIs

- Beispielsweise Terminal
- Designed, damit man nicht zu schnell tippen kann

1.0.2 Klassisches GUI

- Input: Keyboard, Mouse, Pen
- Output: Displays

WIMP-Paradigma

Basiert auf dem WIMP-Paradigma

- Windows → modal (alles andere blockiert - bspw. Speicher-Fenster) / nicht-modal (kann anderweitig weiterarbeiten - Suchfenster im Word)
- Icons → für Dokumente, Programme
- Menus → für Befehle und Auswahl
- Pointer → Pointing, Selecting, Activating, Drawing, dragging

Metaphern

- Desktop
- Files und Folders
- Recycle bin

Widget

- Buttons
- Sliders
- etc

1.0.3 3 Prinzipien der direkten Manipulation

- **Continuous representations** of objects and actions of interest with meaningful visual metaphors
- **Physical Actions** (e.g. dragging) instead of complex syntax
- **Rapid, incremental, reversible (undo) actions** whose effects on the objects of interest are immediately visible (WYSIWYG)

Beispiel Dokument unbenennen

- **Continuous representations** → Icon bzw. Dokument bleibt ersichtlich
- **Physical Actions** → neuer Namen kann einfach eingetragen werden
- **Rapid, incremental, reversible (undo) actions** → Falls man doch nicht umbenennen will, kann man einfach die Aktion stoppen

1.0.4 Vor- / Nachteile GUIs

Vorteile

- Visual representations of task concepts
- allows easy learning
- allows easy retention
- allows errors to be avoided
- encourages exploration
- affords high subjective satisfaction
- allows immediate feedback

Nachteile

- Direct Manipulation is indirected by mouse, trackpad, etc → large translational distance
- High computational demand
- accessibility is a challenge
- automation of tasks more challenging

Natural User Interfaces (NUI)

Wieso braucht es mehr als das WIMP-Prinzip?

→ WIMP ist nicht geeignet für kleine, mobile oder 'unsichtbare' Computers

1.0.5 Charakteristik

- Die Bedingung soll so natürlich wie möglich sein
- Gerade für den Touchscreen wichtig
- allow multimodal interaction → sehr natürlich für die Anwender, da Menschen immer Multimodal agieren (Sprechen, Gestik, Mimik alles gleichzeitig)
- are context aware → From context of task to context of usage ⇒ Can significantly improve mobile applications and user assistance

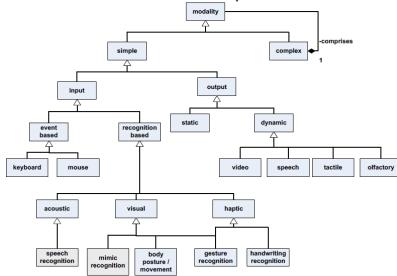


Abbildung 1.1: Übersicht über die verschiedenen Arten der Modalitäten

1.0.6 Main issues of the new modalities

tbd

1.0.7 Ziele

- reduces translational distance (more direct Manipulation)
- fits basic skills of users
- feel natural from the beginning
- all in all: improve usability

Usability

- Usability → Wie einfach kann ich die Anwendung verwenden (Deutsch: Gebrauchstauglichkeit)
- User Experience (UX) → Look and feel der Anwendung
- Customer Experience → Usability + Desirability + Brand Experience

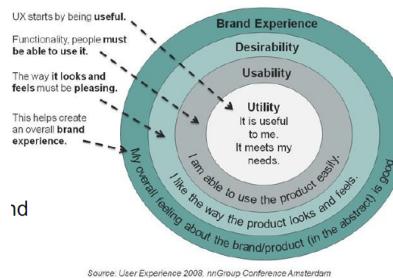


Abbildung 1.2: Usability

Definition nach DIN ISO 9241: The effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments

1.0.8 Die drei wichtigsten Anforderungen rund um die Usability (auch Usability Engineering genannt)

- zu Deutsch: Ergonomie
- Effektiv → die Aufgabe kann komplett abgeschlossen werden
- effizient → mit möglichst wenig Aufwand
- Zufriedenheit → Es soll Spass machen die Aufgabe zu erfüllen

1.0.9 Die sieben wichtigsten Charakteristiken

- Suitability for the task (Aufgabenangemessenheit)
- Suitability for learning (Lernförderlichkeit)
- Suitability for individualisation (Individualisierbarkeit) → Experte möchte schneller sein als Anfänger
- Conformity with user expectation (Erwartungskonformität)
- self descriptiveness (Selbstbeschreibungsfähigkeit) → UI ist selbst erklärend
- Controllability (Steuerbarkeit)
- Error tolerance (Fehlertoleranz)

self Descriptiveness

- genügend Informationen zur Verfügung stellen
- Consider user's knowledge
- Terminologie an die Domäne angepasst
- Bestätigung anfordern vor kritischen Aktionen
- Erschwinglichkeit
- Mapping

suitability for the task

- Möglichst wenig Schritte für die jeweilige Aufgabe
- Nur die notwendigen Informationen für den jeweiligen Prozessschritt angeben
- Kontext-abhängige Hilfe
- möglichst wenig User-input wie möglich
 - same input only once
 - standardwerte
 - vordefinierte Liste für Eingabe (bspw. Länder)

Controllability

- Dialog sollte durch den User bestimmt werden
- Alle Dialogschritte sollten eine undone-Funktion beinhalten
- Möglichkeit anbieten um Aktionen zu canceln

Conformity to expectations

- Betreffend folgenden Punkten
 - Designe
 - Interaktion
 - Struktur
 - Komplexität
 - Funktion
- Terminologie, welche für den User bekannt sind
- Verhalten
- Präsentation der Informationen
 - bspw. Reihenfolge der Benutzereingaben (Name, Adresse etc.)

Error Tolerance

- Fehler vermeiden (Validitätskontrolle)
- Gewisse Aktionen in den bestimmten Situation ein oder ausblenden
- Hilfestellung für den User liefern
- Einfache Möglichkeit den Fehler zu korrigieren
- Auf keinen Fall sollte was verloren gehen, was der User bereits gemacht hat

Suitability of Individualisation

- Sollte für alle User anwendbar sein
 - Level des Users
 - Allfällige Einschränkungen
 - Sprachen

Learnability

- Das System sollte dem User Informationen zu den darunterliegenden Konzepte und Regeln liefern
 - Tool-Tips
 - Tutorials
- Die Basis Funktionalität sollte bedienbar sein, ohne ein Tutorial machen zu müssen

User Centered Design (UCD)

Eine Möglichkeit wie man das Design angehen kann

1.0.10 Begrifflichkeiten

- User Experience Design → Alle Aspekte der Interaktion
- User-centered Design →
- User interface Design → interface zwischen User und System
-

1.0.11 UCD Prozess

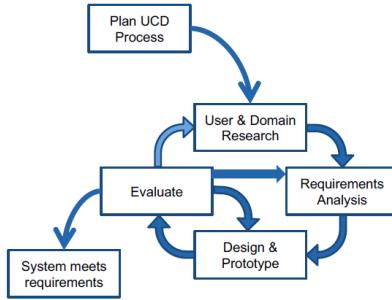


Abbildung 1.3: Prozessablauf

- User und Domain Research
 - Zielgruppe und Domäne definieren
 - Aufnahme möglicher Anforderungen
- Requirement Analysis
 - Aufgaben, Kontext, ...
- Design und Prototype
- Evaluation

User and Domain Research

Wichtigste Phase: Man muss den User und den Kontext verstehen!

- User verstehen
- Kontext verstehen
- klares Verständnis über das Business

User verstehen

- Wer sind die Users?
- Was ist ihre Arbeit, Aufgaben und Ziele?
- Was ist der Kontext ihrer Arbeit?
- Was brauchen Sie um die Ziele zu erreichen?
- Wie ist die Domänensprache?
- Normen (Organisatorisch etc.)
- Pain Points im Workflow

Kontext verstehen

- Wo wird die App verwendet?
- Wann wird die App verwendet?
- Wieso wird die App verwendet?

Personas

Ist eine fiktionale Person, welche eine Gruppe repräsentiert. Dies ist sehr wichtig und gilt als Grundlage für die nächsten Diskussionen.

Usage Scenarios

Beschreibt den **aktuellen** Ablauf, beziehungsweise an allfälliger Use-Case. Dabei gibt es immer die gleichen Aspekte

- title
- Kontext
- Trigger
- Ziel
- Aktion und Interaktion

Artefakten

- Personas
- Usage Scenarios
- Mental Model
- Domain Model
- Stackholder Map → Wer interagiert, wie stark mit der App
- Business Process Model
- Service Blueprint

Requirement Analysis

- Beschreibt was man in der Zukunft möchte
- Kann nicht einfach gesammelt werden
- Keine Features
- Keine Spezifikationen

Context Scenario

- Beschreibt Zukunft
- Beschreibung ist optimistisch
- textliche und in Form einer Geschichte
- aus User-perspektive aus Persona-Sicht
- Braucht folgende Punkte
 - Motivatino, Trigger
 - Persona
 - Kontext
 - Ablenkungen
 - Ziel

1. Szenarios identifizieren für jede Persona
2. Erarbeitung der einzelnen Stories
3. Anforderungen ableiten

UI Design and Prototype

Design einer möglichen Lösung basierend auf den Anforderungen, welche die Ziele effektiv, effizient und zufriedenstellend abdecken.

Evaluate

Testen und evaluieren des UI-Prototypes, mit echten Anwendern und / oder Usability Experten

Kapitel 2

UCD: Conceptual Design and Prototype

Das Hauptziel ist, dass man ein passendes Design-Konzept hat.

- Einigung auf die Design-Prinzipien
 - Charakteristik beschreiben des Produkts mit ein paar Statements
 - kurz und bündig
 - App Definition Statement → bei einer Android-App ist das beispielsweise einen Satz (Was? für wen? welchen (Mehr-)wert? Kontext?)
- Sources
- Wie: Workshop mit dem Design-Team und Stakeholder

2.1 Ablauf eines Conceptual Design

1. Interaktion Styles, Posture und Interaktion Pattern auswählen

2.1.1 Interaction Styles

Können Prozess- oder Produkt-orientiert sein.

Prozess-orientiert:

- Instructing
- Form filling → System fragt Daten ab
 - Für effizienten Daten-input via Text-Feld und Auswahl (bspw. Webformulare oder Login)
- Conversing → Konversationen führen mit dem System
 - User fragt, System Antwortet (text oder speech) (bspw. Chatbot, Suchmaschinen, Roboter)
- Manipulating / Navigating
 - Virtuelle Umgebung von Objekten (bspw. Direkte Manipulation auf dem GUI → Desktop)
 - Man kann navigieren, auswählen, öffnen, zoomen etc.
- Exploring / Browsing
 - Suchen und Finden von Informationen (Bspw. Webseiten)

Produkt-orientiert

- Direkte Interaktion mit einem Objekt
 - Analogie zu 'echten Objekten' (Bspw. Excel, Zahlungsapplikationen, Kompass-App)

2.1.2 Application Posture

- Sovereign → Applikation dominiert den screen (bspw. PowerPoint, Excel)
- Transient → Für eine spezifische Aufgabe (sind die meisten Apps, bspw. Rechner auf dem Handy)
- Daemonic → Arbeitet die meiste Zeit im Hintergrund (bspw. Firewall, Anti-Virus-System)
- Auxiliary → Bleibt aktiv über eine lange Zeit (bspw. Toolbar, media Player)

2.1.3 Interaction Pattern

Sind Lösungen für verschiedene Interaktionsprobleme

Diese Pattern werden häufig miteinander kombiniert (bspw. MS Outlook organizer + tab)

Command-Line

⇒ Textliche Konversation

Vorteil:

- Effizient und flexibel
- Sehr mächtig
- Sehr gut für Experten

Nachteile:

- Nicht einfach zu lernen
- schlecht für Anfänger

Forms

Textliche Konversation, das System fragt explizit die Informationen ab

Vorteil:

- Einfach zu lernen
- Effizient für die Dateneingabe (mit Tastatur)
- Sehr flexibel

Nachteile:

- Weniger effizient für Touch-Interaktionen
- Fehleranfällig (Typos etc.)

Organizer / workspace

Zwei panes (1x Organizer, 1x Workspace) bspw. PowerPoint, Outlook etc.

Parallel workspaces

- erlaubt schnelles Wechseln der Workspaces
- Sehr sinnvoll, wenn man mehrere Aufgaben bearbeiten muss
- Meistens mit Tabs implementiert
- Bspw. Ribbons in Office

Wizard (tunnel)

Der User führt genau eine spezifische Abfolge von Aktionen durch, wobei der User nur vor und zurück kann. Wird vor allem für kritische Aufgaben verwendet (bspw. Software-Installation)

Multi Document Interface (MDI)

ein Hauptwindow und erlaubt mehrere child-windows
→ wird sehr häufig in desktop Applikationen verwendet (bspw. mehrere Word-Fenster offen)

1st person environment

Man sieht die Umgebung wie aus einer eigenen perspektive, wie man dort stehen würde

- video games
- google street
- Schach
- Immersive Environment

3rd person Environment

Der Anwender sieht von aussen auf die Applikation drauf. Wird häufig für Schulanwendungen verwendet.

Hub-And-Spoke

- Nabe und Speichern
- hierarchisches Menü
 - bspw. Mobile-Geräte Gruppieren von Apps, durch das klicken in den App-Ordner, kann man anschliessend die gewünschte App auswählen
- Sehr häufig bei mobilen Geräte verwendet

2.1.4 Metaphern

Beispielsweise das Konzept des Warenkorbs in Online-Shops oder der Papierkorb.

dadurch hat man relativ schnell das Interaktionskonzept definiert, jedoch muss man aufpassen, dass es durch die ganze Applikation konsistent ist

Sehr häufig werden dann mehrere Konzepte miteinander verbunden, da es nicht immer 1:1 übernommen wird - bspw. Fenster (Frische Luft kann man nicht reinlassen im virtuellen Fenster, jedoch kann man es öffnen und schliessen)

Man muss jedoch aufpassen, dass man nicht kulturelle Regeln nicht verletzt. (gibt noch weitere Sachen) ⇒ kann zu Problemen führen

2.1.5 Daten-Objekte identifizieren

Ein Daten-Objekt hat jeweils folgende Typen

- Object-Type (bspw. TimeSlots)
- Definition (bspw. Uninterrupted period of time)
- Relationship (bspw. StartDate, EndDate, StartTime, EndTime)
- States (bspw. Proposed)
- Actions (bspw. CRUD, addVote, removeVote)
- Attributes (bspw. NumVotes, Comment)

...

2.1.6 Function Blocks and Screens

1. Als erstes soll durch die wichtigsten Kontext-Szenarien durchgegangen werden (Funktionen und Daten identifizieren welche gebraucht werden)
2. ...

Storyboarding Screens and Navigation

- mit dem Hauptanwendungsfall und der primären Persona beginnen
- Entscheidung welches Interaktionspattern
- Nur thumbnails (empty or named rectangles)
- Szenario Flow suggests position (Top-down, left-right)
- Object nature suggests size and shape
- Prozedere mit einem anderen Kontext-Szenario und der primären Persona durchgehen
- Probiere Screens zu konsolidieren
- Entwicklung eines groben Storyboards für alle key path scenarios

2.1.7 Touch Interaction Patterns

Gestures

Tab (touch)

- Touch with one finger
- simplest manual touch Gestures
- activate-on-press
- activate-on-release (Better to touch)
- press-and-hold

Tap to select: object, menu item, list item

tap to activate: button, menu, link, app

Double-Tap

- Onto an object / into an area
- to zoom in / out
- To select
- to activate / open

Long-Press

- Onto an object / into an area
- to show context menu

Drag (Pan)

- Drage one finger over the screen (move object, drag-and-drop etc)

- combined with tap, double-tap, long-press
- Wichtig! Sofortiges Feedback

Slide

- Continuous movement, similar to pan
- Movement restricted to horizontal/vertical
- to scroll screen, list of items slowly
- wichtig sofortiges visuelles Feedback

Slide and Hold: for Continuous scrolling (wichtig sofortiges Feedback)

Fling (Swipe)

- to browse a list
- to scroll a screen
- exhibits a momentum
- movement often restricted

Flick

- To brows a list quickly
- To scroll a page quickly
- ...

Pinch

- Two finger tips move closer to eachother on an object / screen
- to shrink object / view (zoom out)

Spread

- two finger tips move away from eachother on an object / screen
- to enlarge object / view (zoom in)

Navigation

- Springboard (Hub and Spoke)
- Cards
- List Menu
- Tab Menu
- Gallery
- Dashboard
- Metaphor / Skeumorphism

2.2 Prototype

- Low fidelity
 - Sketches
 - Storyboards
 - Paper Prototypes
 - wireframes (klickbar)
- High fidelity
 - SW Prototype
 - Video Prototype

2.2.1 Low Fidelity

1. Als erstes mit Stift und Papier
2. Überführung in einem prototyping tool (Balsamiq, Marvelapp)

2.2.2 Evaluieren

Durch den Prototype evaluiert man sein Design, idealerweise direkt mit echten Usern. Die User erhalten eine Aufgabe (bspw. mache einen Termin ab mit der App)

How to do:

- Moderator erklärt
 - die Aufgabe und das Ziel
 - Wie der Prototype funktioniert
- Tester
 - Führt die Aufgabe durch
 - Denkt laut (Sehr wichtig)
- Papier prototype
 - zweite Person gibt dem Tester die Screens welcher er faktisch klicken würde

→ ist ein iterativer Prozess bis man das optimal UI-Design Concept hat

Kapitel 3

Usability Guidelines

3.1 8 goldige Regeln von Ben Shneiderman

1. Streben nach Konsistenz
 - gleiche Aktionen, sollen gleich bedient werden
 - konsistente Befehle
2. Anbieten von Shortcuts
3. informatives Feedback liefern
4. Dialoge sollen so gestaltet werden, dass sie abgeschlossen werden können
 - Sequenzen nach ihren Aktionen gruppieren bspw. Anfang, Mitte und Ende
 - Es ist mental weniger anstrengend für die Bedienenden
5. Fehler verhindern und einfaches Fehlerhandling
 - als erstes sollen Fehler grundsätzlich verhindert werden
 - als zweites sollen Fehler frühzeitig erkannt werden
 - tbd
6. Anbieten von undo-Funktion
 - NUIs: Ist schwierig für Gesten
7. Unterstützung des internen locus
 - Man möchte eine Aufgabe erledigen und nicht sich mit dem Computer auseinanderzusetzen
 - Man möchte nicht das Gefühl des Beifahrers haben
8. Reduzieren sie die Last des Kurzzeitgedächtnisses

3.2 10 Heuristics of Jacob Nielsen

Jacob Nielsen ist einer der Usability-Gurus

1. Visibility of system status
2. Übereinstimmung von Systemen und der realen Welt
 - Sollte sich an dem Domänenmodell bezgl. Begrifflichkeiten und Umgebung anlehnen

3. Benutzerkontrolle und Freiheit

Anbieten eines *emergency exit* um aus einem ungewollten Zustand zu gelangen
support undo und redo
NUIs: schwierig für Gesten

4. Konsistenz und Standard

wörter, sätze und aktionen konsistent verwenden

5. Fehler vermeiden

6. Erkennung statt Erinnerung

Affordances

7. Flexibilität und Effizienz der Benutzung (Analog Shneiderman: Shortcuts)

Accelerators

8. Aesthetic und minimales Design

irrelevante Informationen weglassen

9. Usern helfen Fehler zu erkennen, zu diagnostizieren und erholen von Fehlern

10. Helfen und Dokumentation

3.3 Touch interfaces

Es gibt unterschiedliche Art und Weise wie man das Smartphone bedient.

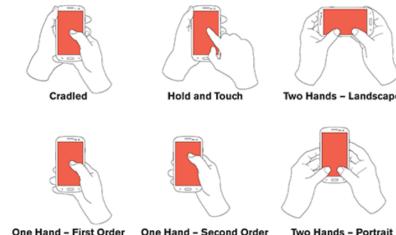


Abbildung 3.1: Unterschiedliche Art und Weise wie ein Smartphone bedient wird

→ Das hat Auswirkung, wo man welche Elemente platziert

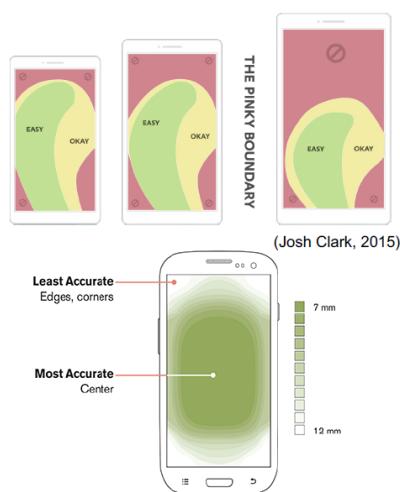


Abbildung 3.2: Übersicht der Flächen für Rechtshänder

- Platzieren von wichtigen Elementen in den Bereiche 'Easy'
- Platzieren von seltenen Elementen in den Bereich wo man schlecht hinkommt
- Platzieren von gefährlichen Aktionen in den Bereich wo man schlecht hinkommt
- Man scrollt häufiger als irgendwelche Schaltflächen zu bedienen
- Links und Rechtshänder bedenken
- Touch-targets müssen gross genug sein (7-12mm in jede Richtung)

Ebenfalls muss man den Ausschluss von Informationen bedenken

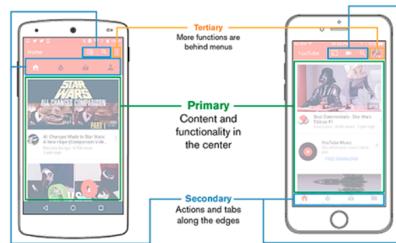


Abbildung 3.3: Einteilung der Bereiche

- wichtige Inhalte in die primäre Zone (Mitte)
- Tendenziell halten die Menschen das Smartphone eher weiter unten, bei sehr grossen Geräten
- Eine geniale erste Erfahrung mit dem Kunden
 - einfache erste Bedienung
 - kein langwiriger Anmeldeprozess
- Funktionale Animation
 - Jedoch sollte dies nicht übertrieben werden → es gibt Personen mit Epilepsie etc.
- Der ganze Service-Prozess überarbeiten, nicht nur die App
 - nicht nur die App testen, sondern der ganze Prozess welcher dazu gehört
 - Push-Notification verhindern

3.3.1 Zusammenfassend

- Touch ist nicht präzise
- Die User nutzen nur was sie sehen
- Telefone sind nicht flach → wenn man den Daumen strecken muss, dann bewegt man das Telefon drei dimensional
- Tests mit den entsprechenden Personen

3.4 Styleguides

Styleguides gibt es für Android, Apple und Windows und definieren wie eine App in etwa auszusehen hat und wie die Interaktionen damit aussehen.

Diese dienen als Orientierung, dass man die oben genannten Regeln nicht auswendig wissen. → auch in Styleguides ist nicht alles perfekt, aus diesem Grund soll beispielsweise die App mit den Nielsen-Regeln überprüft werden

Kapitel 4

Detailed Design of NUIs

4.1 Affordance

Was zeichnet ein gutes Interaktionselement aus? → Affordances (Angebotscharakter)
bspw. bei einem Link die unterstrichene Linie

- Perceptible Affordance
bspw. unterstrichener Link
- False Affordance
bspw. Schaltfläche welche man nicht drücken kann
- Hidden Affordance
bspw. Schütteln bei der SBB ab (Niemand kennt diese Funktion)

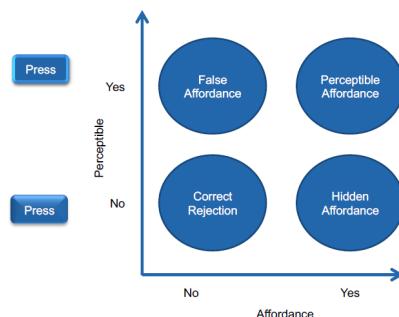


Abbildung 4.1: Affordance Eingliederung

4.1.1 Affordance für Gesten

- tap
- Slide, Swipe, Fling, Flick
- Drag

Kapitel 5

App Design

5.1 Prozess

1. Research
2. Design
3. User Testing (Usability)
4. Coding
5. User Testing (Usability, A/B, Betas)
6. Publish

5.2 Research

5.2.1 Design Consideration

- Context
- Purpose
- Target Audience
- Target Device

Purpose

- Digitaler Assistent für kleine Tasks (Todo, Notizen)
- Zeitvertreib
- Digital enhancement or support

Target Audience

- Desktop: <Role>
- Mobile: <Role> + <Context>
- + 'bonus' user groups like impaired people

5.3 Design Process

1. Sketch
2. Wireframes
3. Detail Design

5.4 Design for Android

5.4.1 Alternate Layouts

Sind dafür da, dass man die unterschiedlichen Device-Typen mit der Android-App definieren kann.

5.4.2 px, dp, sp

- elastisches Design ist notwendig
- skalierbare Größen:
 - dp = px * scaling factor
 - sp = dp * user's font size
- für Bilder kann man vektor Grafiken verwenden

→ dp und sp immer verwenden, falls möglich

Vector vs. Raster Graphics

- Support von vector Grafiken ist nicht all zu gut supported bei Android
- Für jeden Bildpunkt wird 3 Bytes gebraucht

5.4.3 Android Desnity Buckets

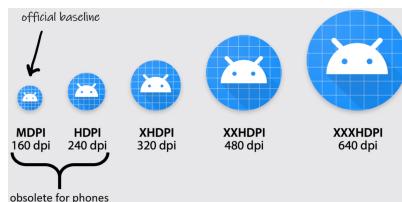


Abbildung 5.1: Android Density Buckets

→ auf jedem Gerät ist die Grösse im Vergleich zum Bildschirm gleich gross

5.4.4 9-Patch Files

- Ist eine Spezialität von Android, gibts bei IOS nicht
- Ist eine Art und Weise wie man Bilder für Android aufbereiten kann

5.4.5 Was ist klickbar?

Woran erkenne ich, wenn eine Schaltfläche klickbar ist?

- Hervorhebung
- Beschriftung
- Farbe
- Platzierung

5.5 Material Design

- Ist inspiriert durch die echte Welt
- reflektiert diese Belichtung und Schattierung
- Manchmal jedoch eher schwierig zu erkennen

5.5.1 Basic Structure

- Top-Bereich (App-Bar) ist mehr oder weniger fixiert, hier sollte man nicht sonderlich kreativ sein
- Jedoch kann man beim Hauptbereich relativ kreativ sein

5.5.2 Navigation in Android

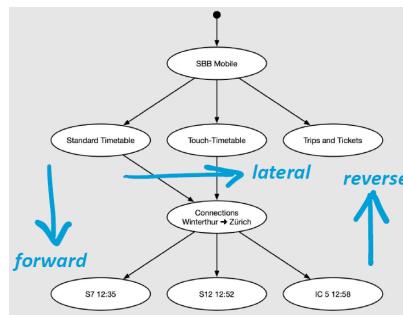


Abbildung 5.2: Navigation in Android

- Bottom navigation → 2-5 items
- Tabs → 2-5 items
 - Funktioniert auf jedem Level bspw. auf jedem Level eigene Tabs
- drawer → 5-9 items
- Springboard → 5-9 items ⇒ nicht grundsätzlich akzeptiert, muss validiert werden
- Know and unknown → 5-9 items ⇒ nicht grundsätzlich akzeptiert, muss validiert werden

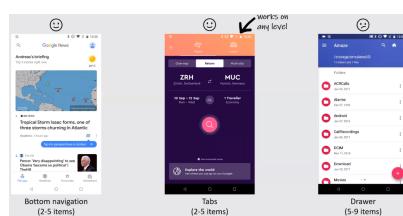


Abbildung 5.3: Lateral Navigation in Android

Was kann man machen, wenn man mehr Elemente hat?

- Informationsarchitektur anpassen
-

5.6 Design Empfehlungen

- Alle Screens nach Nielsen überprüfen
- an Guidelines halten
- Schau was andere machen
- gib immer Feedback
- Navigation immer ersichtlich
- Alle Icons benennen
- Farben allein reichen nicht aus
- Genügend Kontrast herstellen
- genügend grosse und dicke Schriften
- accessibility zulassen
-

Kapitel 6

App Design mit Kotlin

6.1 Aufbau

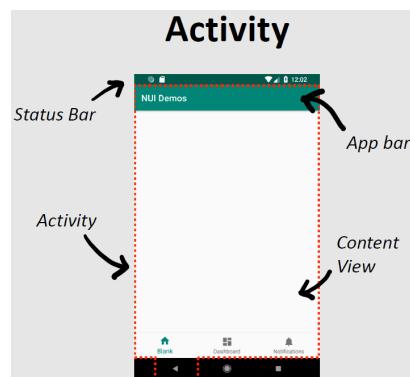


Abbildung 6.1: Activity dargestellt in roter Linie

Frage mente

- Lebt innerhalb der Activity
- Für wechselnder Content (tabs, swiping, dialogs)
- ermöglicht flexible Layouts

Views

- Alles was man auf dem Screen sieht ist eine View
- Activity und Fragments sind leere Hüllen

6.2 App Strukturierung

Möglichkeit

1. Mehrere Aktivitäten
2. 1 Aktivität, viele Fragmente
3. 1 Aktivität, viele Views → heute sehr beliebt bei grossen Teams / Apps

Kapitel 7

Android App Development II

7.1 Threading

UI-Thread: Sämtliche Systeme sind als single Threads abgebildet
Dafür Zuständig, dass die Events realisiert werden bzw. Scrollen, Button klicken etc.

Nachteil: der UI-Thread kümmert sich immer nur um einen Thread → wenn es relativ lange braucht um den Event zu bearbeiten, dann ist die Anwendung so lange blockiert

Man hat 16.7ms Zeit den Screen in der App aufzubauen

7.2 Fast lists

7.3 Concurrent Programming

Kapitel 8

Patterns for UIs

8.1 Observer Patterns

Man hat an diversen Stellen im UI dieselbe Funktionalität. Beispielsweise in Spotify der Pause-Button. das Observer Pattern löst dieses Problem

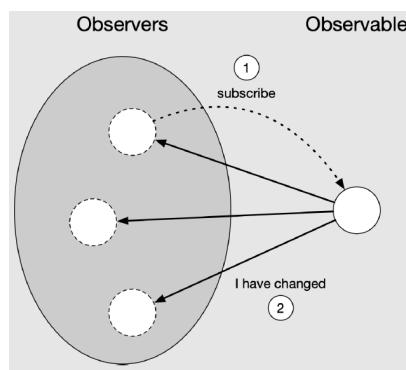


Abbildung 8.1: Der Mechanismus des Observer Pattern

- Observable: Quelle
- Observer: Subscriben das Observable, welche interessiert an diesem sind
- Es ist reactive
- Man muss nicht nachfragen, sondern es wird informiert
- Gute Entkopplung

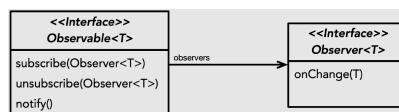


Abbildung 8.2: Die Struktur (in UML) des Observer Pattern

Anwendung von Observer

- Event Handling - addOnClickListener()
- RxJava
- EventBus (Guava, Spring)
- Data Binding

8.2 Model View Controller and Friends

8.2.1 Model View Controller

- Ohne Observer gibt es kein MVC
- Trennung zwischen Model, View und Controller

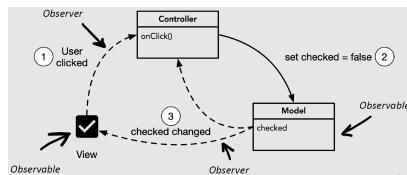


Abbildung 8.3: MVC Abbildung

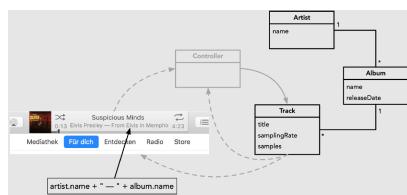


Abbildung 8.4: MVC-Model Abbildung

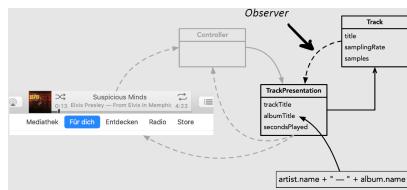


Abbildung 8.5: MVC-Presentation Abbildung

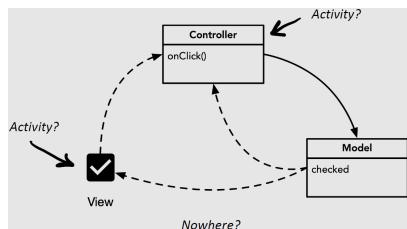


Abbildung 8.6: MVC-Activity Abbildung

8.2.2 MVP

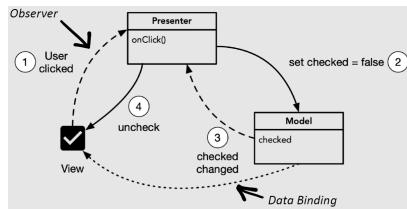


Abbildung 8.7: MVP Abbildung

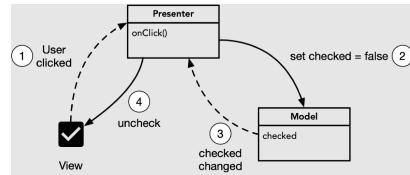


Abbildung 8.8: MVP Passive View Abbildung

8.2.3 Model View ViewModel

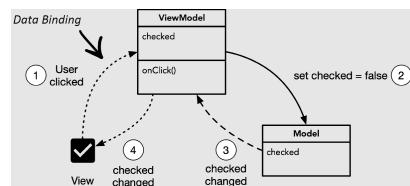


Abbildung 8.9: MVVM Abbildung

8.3 Chain of Responsibility

Behandelt sich darum, wer ist verantwortlich einen Event abzuarbeiten

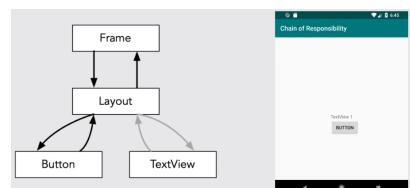


Abbildung 8.10: Chain of Responsibility

Frame kann nichts damit anfangen → Frame fragt Layout, kannst du was damit anfangen? → Layout kann ebenfalls nichts damit anfangen → Layout fragt Button, kannst du was damit anfangen? → Button antwortet, mit ja ich kümmere mich darum

8.4 Composite

Kapitel 9

User and Domain Research

Ist eine der wichtigsten Phase, jedoch gibt es sehr viele Ausreden, weshalb man diese Phase überspringen möchte

9.1 User

Unterscheidung wie folgt

- Primary
 - brauchen das System häufig und interagieren direkt mit dem System
- Secondary
 - indirekte oder unregelmässige Verwendung des Systems
- Tertiary
 - Sind betroffen von der Einführung des Systems oder beeinflussen den Kauf des Systems

Was wollen wir von den Usern wissen?

- Wer sind die User?
- Was sind die Arbeiten, Aufgaben und Ziele?
- Was ist der Kontext der Arbeit?
- Was sind die Bedürfnisse um die Ziele zu erreichen?
- Wie sieht die Domänsprache aus
- Gibt es Normen (Organisatorische, kulturelle und soziale)
- PaintPoints im Workflow
- Zusätzlich für mobile Apps relevant

Wo wird die App verwendet (Umgebung: Zuhause etc.)

Wann wird die App genutzt (Nacht, Tagsüber etc.)

Wieso wird die App genutzt (Motivation und Trigger)

Wie geht man dabei vor?

1. Domäne lernen (Ziele, Aufgaben, Konzepte, Begriffe)
2. Absicht definieren

3. Ressourcen definieren (time, ressources, finanzielle)

4. Design Research

Zieluser (alter, geschlecht, location, beruf, ausbildung, produkterfahrung)

research questions to answer

research methods

roles

Equipment to use

5. Personen rekrutieren und Zeitplan definieren

6. Research und protokollieren

7. Diskussion und Interpretation

single case analysis / cross case analysis

induktiv / dekutiv

8. Zusammenfassen und dokumentieren (user stories, personas etc.)

Techniken für user und domain research

- Kontextuelle Nachfrage (contextual inquiry)

Experte folgt einem User beim Ausüben des Tasks und stellt dabei Fragen

Mischung aus contextual interview und direkter Beobachtung

Das wird benötigt: Aufnahmegerät, Kamera, Notizbuch und Stift

Geeignet für: Fehlerauffindung und Szenarien eruieren

- Contextual interview

vor Ort

Klassifiziert nach: strukturiert (geschlossene Fragen), halbstrukturiert (geschlossene und offene Fragen), nicht-strukturiert (keine vorbereiteten Fragen)

- Interviews

seperates Kapitel

- Direkte oder indirekte Beobachtungen

- Fokusgruppen / Workshops

- Befragungen

- Umfragen

- Desktop research → studying documentation, konkurrenz

| Technique | Good for | Kind of data | Pros | Cons |
|------------------------|--|----------------------------|---|--|
| Questionnaire, Survey | Assessing specific Questions | Quantitative & qualitative | Can reach many people | Design crucial response rate may be low |
| Interview | Exploring issues, problems involve users meet stakeholders | Mainly qualitative | Enforces contact with users, stakeholders | Time consuming |
| Direct observation | Understanding context | Qualitative | Observing actual work in context | Time consuming |
| Studying documentation | Business context, brand, regulations, procedures | Quantitative | No users involved Actual work will differ from documented procedures | |
| Focus groups Workshops | Collecting multiple viewpoints | Mainly qualitative | Highlights areas of consensus or conflicts Enforces contact | Dominant characters |

Abbildung 9.1: Übersicht der Techniken

9.1.1 Interview

Planung:

- User-Rollen identifizieren
- Bestimmen der mind. Anzahl an Interviewpartner pro Rolle
- Für jeden Faktor der das User-Verhalten betrifft, soll es mit der Anzahl unterschiedlichen Gruppen multipliziert werden
- Anzahl Interview herunterbrechen
- Einplanen, dass es gewisse No-Shows geben wird
- Rekrutierung und Planung kann bis zu 2-4 Wochen gehen

Wie?

- Offene Fragestellung (Wer, Was, Wo, Wieso, wann, wie?)
- Es sollte mehr eine Konversation sein
- Sympathisch und nicht verurteiltend
- Sei der "Lernende und nicht der Experten"
- naive Frage stellen
- Die Personen sollen die Sachen zeigen
- Frage nach spezifischen Stories
- Falls etwas relevantes vorkommt oder so aussieht, nachfragen
- vom aktuellen Zustand zum Problem
- auf Inkonsistenz achten
- Gestik und Mimik beachten
- 30-60 min pro Interview inkl. 30min Nachbearbeitung

Struktur:

- Introduction
- Warmup
- Main Session
- Cool-off period
- closing

Kapitel 10

Menschliche Wahrnehmung

10.1 Das Menschliche Wahrnehmungssystem

Die 5 Sinne

- Sehen (80% der Informationen)
- Hören (15% der Informationen)
- Tastsinn
- Geruchssinn
- Geschmacksinn

Die Sensorik ist ein kurzzeit-Gedächtnis und ist im Sinne einer FIFO-Queue

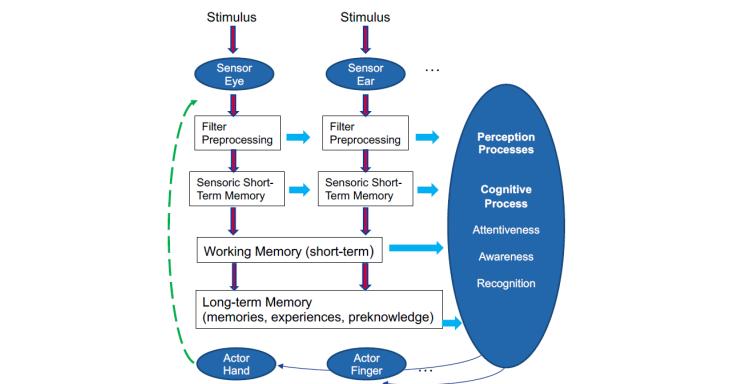


Abbildung 10.1: Abbildung des Ablaufs der Wahrnehmung eines Menschen

10.1.1 Sehen

Das Menschliche Auge:

- Iris, Pupille
- Retina (Netzhaut) enthält die visuellen Rezeptoren
- Macula (gelber Fleck)
- Fovea (Sehgrube) → Ort grösster Sehschärfe
- Blinder Fleck → Hier verlassen die Nerven das Auge, entsprechend keine Rezeptoren

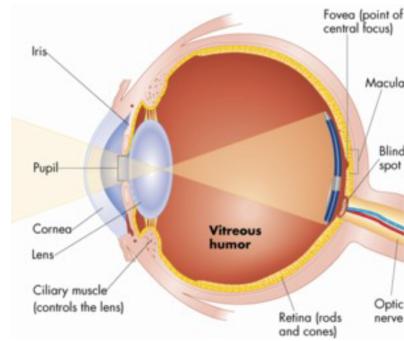


Abbildung 10.2: Abbildung des menschlichen Auges

Netzhaut

Hat zwei unterschiedliche Typen von Rezeptoren

- Zäpfchen-Zellen
L(rot), M(grün), S(blau)
für das farbliche Sehen
Hauptsächlich in der Macula
Nur in Fovea
- Stäbchen-Zellen
Für das Schwarz/weiss Sehen
20mal mehr als Zäpfchen
Rest der Netzhaut
ca. 200-250 grautöne

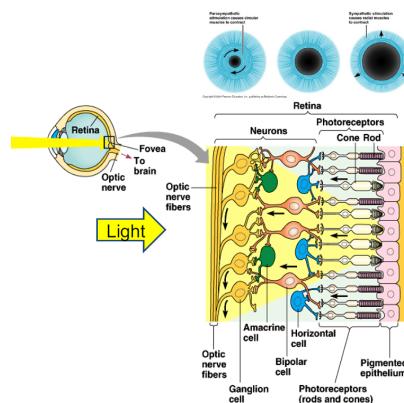


Abbildung 10.3: Abbildung der Netzhaut

Farbblindheit

- Protanopie (red-blindness)
kann nur schwer zwischen rot, grün, gelb und braun unterscheiden
- Deutanopie (green-blindness)
kann nur schwer zwischen rot, grün, gelb und braun unterscheiden
- Tritanopie (blue-blindness)
kann nur schwer rot von orange, blau von grün, etc. unterscheiden

dynamische Sicht

Bei mehr als 22Bilder pro Sekunde nimmt das Auge dies als dynamische Bewegung auf → Film

Tiefenwahrnehmung

tbd

10.1.2 Hören

Der Gehörsinn ist für das GUI nur von sekundärer Relevanz, jedoch für mobile interaktionen elementar.

Das Sprechen basiert auf Schallwellen, welche von unserem Gehör aufgenommen wird. Diese Schallwellen werden in Schallpegel (dB) gemessen
evtl noch weiter ausführen

10.2 menschlicher Verstand

10.2.1 Kurzzeitgedächtnis

- Sensorischer Speicher
 - Speichert Sensorsignale im FIFO-Prinzip für ca. eine Sekunde
- Arbeitsspeicher
 - extrahiert Dinge und Objekte (bspw. analoge Uhr → ablesen der Uhrzeit)
 - speichert die Dinge auf welche mir uns aktuell fokussieren
 - definiert die Aufmerksamkeit, Kontext und Realität
 - kann nicht geteilt werden
 - Kapazität ist limitiert (ca. 7 +/- 2 Dinge / Stücke)

Aufmerksamkeit

- ist durch das Kurzzeitgedächtnis bestimmt und limitiert
- Menschen können verschiedene Dinge gleichzeitig ausführen (vor allem Frauen)
Jedoch können wir uns **nur auf einen einzigen Task gleichzeitig konzentrieren** → die andere Dinge müssen automatisch erledigt werden bspw. Autofahren, zuhören, Fahrrad fahren

⇒ Konsequenzen für das UI Design

- Die Interaktion mit dem UI sollte so einfach wie notwendig ausfallen, dass dies für (regelmässige Anwender) automatisch passiert
- Die Benutzer konzentrieren sich auf die Aufgabe, welche erledigt werden muss, nicht auf die Interaktion mit dem UI

10.2.2 Langzeitgedächtnis

Hier befindet sich alles was wir bis anhin in unserem Leben gelernt haben.

s

ACT-R Model

- deklaratives Modell
 - Fakten, Daten
 - Konzepte (Wörter, Grammatikregeln, Gefühle, Geschmäcke)

- kognitive Unstimmigkeit
gewisse Objekte kommen nicht im gewohnten Kontext und sind dadurch nicht eindeutig zu identifizieren
- produktiver Speicher
Fähigkeiten (motorische Fähigkeiten bspw. Skifahren oder Autofahren)
Automatismen (werden in gewissen Situationen getriggert und automatisch ausgeführt, wozu es keine Aufmerksamkeit braucht)
- Tasks

⇒ Wichtig für GUI-Design: Sequence of steps for the same action should always be the same (consistency)

10.2.3 Mental Model

- Menschliches Model der Realität
- Speichert Konzepte (Nomen) und Assoziationen (Nomen, Attribute, Aktivitäten)
- Assoziationen
Verbindet neue Fakten und Features mit bestehenden Konzepten
Bessere Erinnerung
- sind sehr anpassungsfähig

Mismatch

If the systems UI does not match the actual users mental model, interaction gets very difficult!

Designer

- Entwickelt ein mental model des Problems (domain model)
- Entwickelt das System und die UI nach dem erstellen Model

System

- Stellt ein Bild des Systemverhaltens durch die UI dar
- Basiert auf dem Mental Model des Designers

Users

- Erstellt das Mental Model aufgrund von
seiner aktuellen Erfahrungen und seines Wissens
geliefertes Systembild
interaktionen mit dem System

User Centered Design

1. User Research

Probiert herauszufinden, wie das mentale model der User beim erstmaligen Verwenden des Systems aussieht

2. Design System and UI

Welches das Problem löst

Stellt die UI einem User zur Verfügung

10.2.4 Erfahrung

- Summe des Wissens
- Automatismen
- MentalModel
- Zwei extreme Ausprägungen

Experts

- Viel Wissen und Automatismen
- Wissen ist oftmals implizit
- Kann sich voll auf die Lösung des Problems konzentrieren
- sehr effizient

Beginner

- Keine Erfahrung
- Braucht einfache Fakten und Regeln
- Müssen zuerst ihr eigenes mentale Modell der Applikation entwickeln
- Müssen sich nicht nur auf die Lösung konzentrieren, sondern auch auf die Aufgabe und Interaktionen
- Braucht wesentlich mehr Zeit für die Aufgabe

Normal User

- Oft ein gewöhnlicher User
- Have to recall / rebuild mental model of the system behaviour each time

10.2.5 Lerntypen

Die Systeme sollten die User unterstützen im Lernen des Programms, dabei sollte man beachten, dass es unterschiedliche Lerntypen gibt (Sehen, hören, machen, sprechen)

Das Lernen wird durch folgende Punkte unterstützt:

- Neue mental models werden von existierenden abgeleitet
- Metaphern (bspw. Papierkorb)
- Schrittweise zusammenführen von neuem mit bestehenden Wissen
- Berücksichtigen der Lerntypen

10.3 Menschliche Erwartungen

Erfahrung führt zu Erwartungen

Das Kano-Modell führt zur Unterstützung der Anforderungen

10.4 Menschliches Handeln

tbd

Kapitel 11

Web Accessability

- ca. 10-15 Prozent Weltweit sind behindert
- ca. 300k haben eine Sehbehinderung in der Schweiz
- ca. 500k haben eine Gehörbehinderung

11.1 WCAG

- WCAG: Web Content Accessability Guidelines
- in 4 Prinzipien
 - Perceivable
 - Operable
 - Understandable
 - Robust
- Dabei gibt es Erfolgskriterien A, AA, AAA, AAAA

11.2 UAAG

- User Agent Accessability Guidelines
- Guidelines on how to make user agents accessible
- User agents include browsers, browsers extention

11.3 ATAG

- Authoring Tool accessibility Guidelines
- Guidelines on how to make authoring tools such as code editors accessible
- Lowers the barrier and provide support for people with disabilities to create more accessible Web content

11.4 ARIA

- Accessible Rich Internet Applications
- A specification to make Web content more accessible to people with disabilities
- Especially helpful for dynamic content and advanced user interface controls developed using JavaScript
- ARIA stellt verschiedene Rollen zur Verfügung (bspw. Header, main content, navigation, footer)

Kapitel 12

Code-Snippets Android

12.1 Material Design as Library

```
dependencies {  
    implementation "com.google.android.material:material:1.1.0"  
}
```

Abbildung 12.1: Code Snippet Material Design Dependency

12.2 item Layout

```
<?xml version="1.0" encoding="utf-8"?>  
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_margin="8dp">  
  
    <TextView  
        android:id="@+id/simple_text"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        tools:text="This is some temp text" />  
</FrameLayout>
```

Abbildung 12.2: Code Snippet itemLayout

12.3 Kotlin Main Activity

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(state: Bundle?) {  
        super.onCreate(state)  
        setContentView(R.layout.activity_main)  
    }  
}
```

Abbildung 12.3: Code Snippet Main Activity in Kotlin

```
class ProjectsFragment : Fragment() {  
    private lateinit var projectsViewModel: ProjectsViewModel  
    override fun onCreateView(  
        inflater: LayoutInflater,  
        container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View? {  
        projectsViewModel =  
            ViewModelProvider( owner: this).get(ProjectsViewModel::class.java)  
        val root = inflater.inflate(R.layout.fragment_projects, container, attachToRoot: false)  
        //...  
    }  
}
```

Abbildung 12.4: Code Snippet Using LayoutInflater to set view for fragment

12.4 Activity Lifecycle

Android Documentation

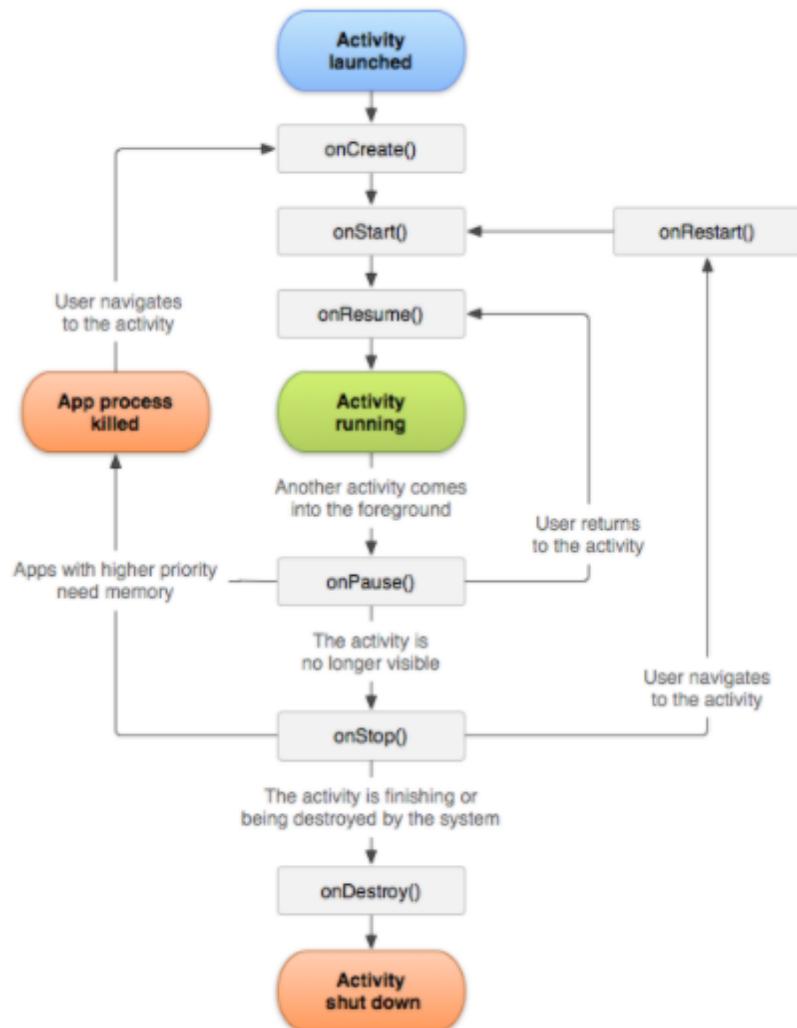
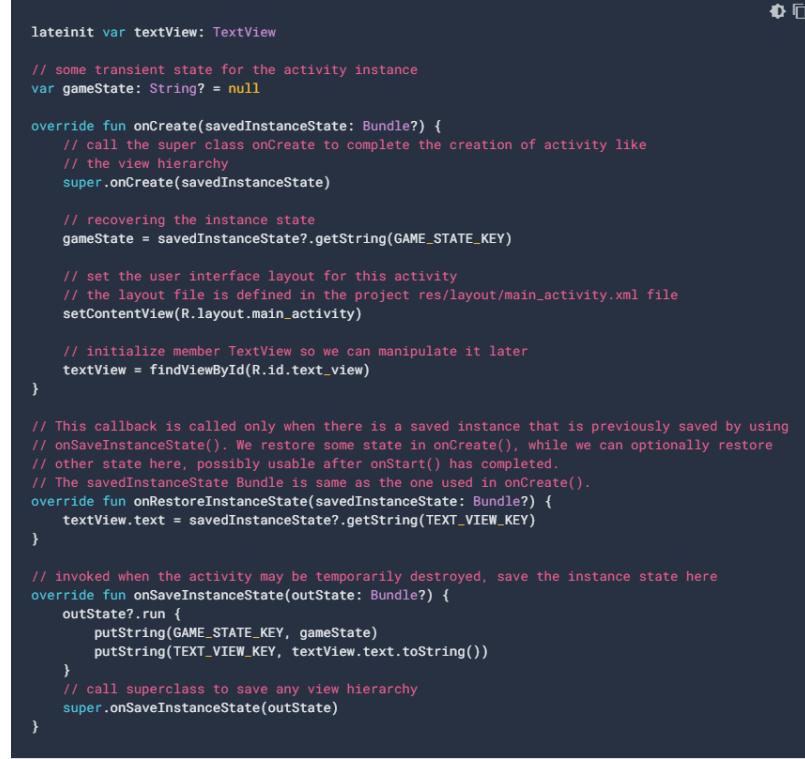


Abbildung 12.5: Activity Lifecycle

12.4.1 onCreate()



```
lateinit var textView: TextView

// some transient state for the activity instance
var gameState: String? = null

override fun onCreate(savedInstanceState: Bundle?) {
    // call the super class onCreate to complete the creation of activity like
    // the view hierarchy
    super.onCreate(savedInstanceState)

    // recovering the instance state
    gameState = savedInstanceState?.getString(GAME_STATE_KEY)

    // set the user interface layout for this activity
    // the layout file is defined in the project res/layout/main_activity.xml file
    setContentView(R.layout.main_activity)

    // initialize member TextView so we can manipulate it later
    textView = findViewById(R.id.text_view)
}

// This callback is called only when there is a saved instance that is previously saved by using
// onSaveInstanceState(). We restore some state in onCreate(), while we can optionally restore
// other state here, possibly usable after onStart() has completed.
// The savedInstanceState Bundle is same as the one used in onCreate().
override fun onRestoreInstanceState(savedInstanceState: Bundle?) {
    textView.text = savedInstanceState?.getString(TEXT_VIEW_KEY)
}

// invoked when the activity may be temporarily destroyed, save the instance state here
override fun onSaveInstanceState(outState: Bundle?) {
    outState?.run {
        putString(GAME_STATE_KEY, gameState)
        putString(TEXT_VIEW_KEY, textView.text.toString())
    }
    // call superclass to save any view hierarchy
    super.onSaveInstanceState(outState)
}
```

Abbildung 12.6: Code Snippet Create Activity Lifecycle onCreate()

12.4.2 onResume()



```
class CameraComponent : LifecycleObserver {

    ...

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    fun initializeCamera() {
        if (camera == null) {
            getCamera()
        }
    }

    ...
}
```

Abbildung 12.7: Code Snippet Create Activity Lifecycle onResume()

12.4.3 onPause()

```
class CameraComponent : LifecycleObserver {  
  
    ...  
  
    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)  
    fun releaseCamera() {  
        camera?.release()  
        camera = null  
    }  
  
    ...  
}
```

Abbildung 12.8: Code Snippet Create Activity Lifecycle onPause()

12.4.4 onStop()

```
override fun onStop() {  
    // call the superclass method first  
    super.onStop()  
  
    // save the note's current draft, because the activity is stopping  
    // and we want to be sure the current note progress isn't lost.  
    val values = ContentValues().apply {  
        put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText())  
        put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle())  
    }  
  
    // do this update in background on an AsyncQueryHandler or equivalent  
    asyncQueryHandler.startUpdate(  
        token,           // int token to correlate calls  
        null,            // cookie, not used here  
        uri,             // The URI for the note to update.  
        values,          // The map of column names and new values to apply to them.  
        null,            // No SELECT criteria are used.  
        null             // No WHERE columns are used.  
    )  
}
```

Abbildung 12.9: Code Snippet Create Activity Lifecycle onStop()

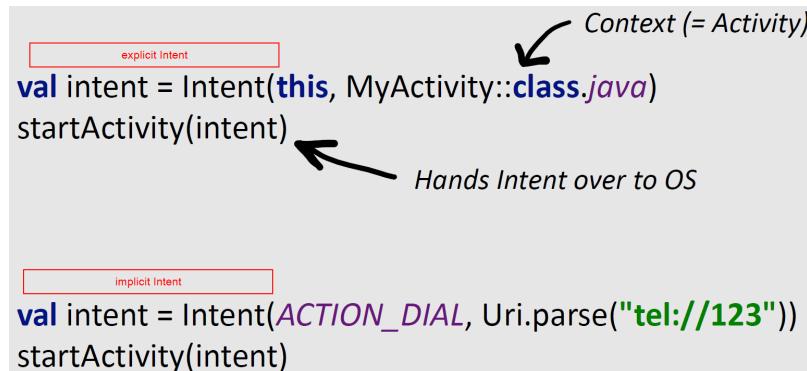
12.4.5 Preserving Instance State

```
override fun onSaveInstanceState(outState: Bundle?) {  
    super.onSaveInstanceState(outState)  
  
    // save state to bundle  
}  
  
override fun onRestoreInstanceState(inState: Bundle?) {  
    super.onRestoreInstanceState(inState)  
  
    // restore state from bundle  
}
```

Abbildung 12.10: Code Snippet Preserving Instance State

12.5 Intents

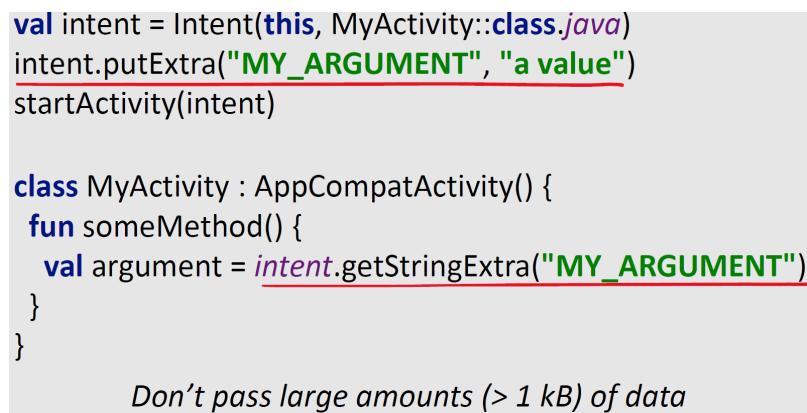
12.5.1 Creating Intents



```
explicit Intent  
val intent = Intent(this, MyActivity::class.java)  
startActivity(intent)  
  
implicit Intent  
val intent = Intent(ACTION_DIAL, Uri.parse("tel://123"))  
startActivity(intent)
```

Abbildung 12.11: Code Snippet Creating explicit and implicit intents

12.5.2 Passing Arguments



```
val intent = Intent(this, MyActivity::class.java)  
intent.putExtra("MY_ARGUMENT", "a value")  
startActivity(intent)  
  
class MyActivity : AppCompatActivity() {  
    fun someMethod() {  
        val argument = intent.getStringExtra("MY_ARGUMENT")  
    }  
}  
  
Don't pass large amounts (> 1 kB) of data
```

Abbildung 12.12: Code Snippet passing arguments

12.5.3 Real-Life Example

```
private fun dispatchTakePictureIntent() {
    val packageManager: PackageManager = requireContext().packageManager
    Intent(MediaStore.ACTION_IMAGE_CAPTURE)
        .also { takePictureIntent ->
            takePictureIntent.resolveActivity(packageManager).also {
                // Create the File where the photo should go
                val photoFile: File? = try {
                    createImageFile()
                } catch (ex: IOException) {
                    // Error occurred while creating the File
                    null
                }
                // Continue only if the File was successfully created
                photoFile?.also {
                    val photoURI = getUriFromFile(
                        requireContext(),
                        BuildConfig.APPLICATION_ID +".fileprovider",
                        it
                    )
                    takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI)
                    startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)
                }
            }
        }
}
```

Abbildung 12.13: Code Snippet Photo intent from neophyta project

12.6 Application Object aka Single Thread

```
class MyApplication : Application() {

    val executor = Executors.newSingleThreadExecutor()
}
```



Lives as long as app

Abbildung 12.14: Code Snippet Creating single Thread which lives as long as app

12.7 Handling Events

```
<Button
    android:id="@+id/a_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tap me!"/>
```



```
val demoButton: Button = findViewById(R.id.a_button)
demoButton.setOnClickListener {
    // trigger action
}
```

Abbildung 12.15: Code Snippet handling events

```
// Floating action button for new todo
val fab: FloatingActionButton = root.findViewById(R.id.todos_fab)
fab.setOnClickListener { it: View!
    findNavController().navigate(R.id.action_TodosFragment_to_TodosAddFragment)
}
```

Abbildung 12.16: Code Snippet handling events with trigger action

12.8 Accessing Ressources

```
textViewOne.setText(R.string.app_name)

<resources>
    <string name="app_name">NUI Demos</string>
</resources>

<TextView
    android:id="@+id/textViewOne"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/app_name"/>
```

Abbildung 12.17: Code Snippet how to access to ressources

12.9 System Ressources

```
textViewOne.setText(R.string.app_name)

<resources>
    <string name="app_name">NUI Demos</string>
</resources>

<TextView
    android:id="@+id/textViewOne"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/app_name"/>
```

Abbildung 12.18: Code Snippet how to access to ressources

12.10 UI Thread

```
manual approach: Thread.currentThread() == Looper.getMainLooper().thread  
Annotations for static analysis @UiThread fun manipulateUi() {  
    // lint warning if not executed on UI thread  
}  
  
Using kind of a debug mode StrictMode.setThreadPolicy(StrictMode.ThreadPolicy.Builder()  
.detectDiskReads()  
.detectDiskWrites()  
.detectNetwork()  
.penaltyLog()  
.penaltyDeath()  
.build())  
  
↳ Writes to logcat  
↳ Kills app
```

Abbildung 12.19: Code Snippet different var for handling UI Threads

12.11 Callback to UI Thread

```
class HandlerActivity : AppCompatActivity() {  
  
    fun someCallback() {  
        this.runOnUiThread {  
            // happens on UI thread  
        }  
    }  
}
```

Abbildung 12.20: Code Snippet how to implement a Callback to UI Thread V1

```
val handler = Handler(Looper.getMainLooper())  
handler.post {  
    // happens on UI thread  
}  
handler.postDelayed({  
    // happens on UI thread after ~1 sec  
, 1000)
```

Abbildung 12.21: Code Snippet how to implement a Callback to UI Thread V2

12.12 Adapter-Class - Fast-List

12.12.1 Adapter Class extends Viewholder

```
class ToDoAdapter(val myDataset: MutableList<ToDo>, private val origin: String) :  
    RecyclerView.Adapter<ToDoAdapter.MyViewHolder>()
```

Abbildung 12.22: Code Snippet implement adapter class

12.12.2 viewholder

```
override fun onCreateViewHolder(  
    parent: ViewGroup,  
    viewType: Int  
>: MyViewHolder {  
    val view = LayoutInflater.from(parent.context)  
        .inflate(R.layout.single_list_entry, parent, attachToRoot: false) as View  
    return MyViewHolder(view)  
}
```

Abbildung 12.23: Code Snippet creating a viewholder

12.12.3 onBindViewHolder

```
override fun onBindViewHolder(holder: MyViewHolder, position: int) {  
    // do stuff depending on list item  
}
```

Abbildung 12.24: Code Snippet onBindViewHolder is specific to app, problem, list entry

12.12.4 Java

onCreateViewHolder(ViewGroup, int): This method is called right when the adapter is created and is used to initialize your ViewHolder(s).

getItemViewType(int): This method returns an ‘Integer‘ which represents the view type. Since the Android system stores a static reference to each layout as an ‘Integer‘ in the “R” (resources) class, we can simply return the layout resource id to be used in the ‘onCreateViewHolder()‘ method.

getItemCount(): This method returns the size of the collection that contains the items we want to display

onBindViewHolder(RecyclerView.ViewHolder, int): This method is called for each ViewHolder to bind it to the adapter. This is where we will pass our data to our ViewHolder.

```

public class SimpleAdapter extends RecyclerView.Adapter {
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent,
                                                   int viewType) {
        final View view = LayoutInflater.from(parent.getContext())
            .inflate(viewType, parent, false);
        return new SimpleViewHolder(view);
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder,
                               int position) {
        ((SimpleViewHolder) holder).bindData(models.get(position));
    }

    @Override
    public int getItemCount() {
        return models.size();
    }

    @Override
    public int getItemViewType(int position) {
        return R.layout.item_simple_itemview;
    }
}

```

Abbildung 12.25: Code Snippet adapter in Java

12.13 PresentationModel

```

class PlayerPresentationModel {
    Observable<String> trackName;
    Observable<String> artistName;
    Observable<String> albumName;
    Observable<File> coverImage; // Pfad, kein Bitmap o.ä.!
    Observable<Boolean> isPlaying;
    Observable<Integer> playPosition; // numerischer Typ
    Observable<Integer> loudness; // numerischer Typ
}

```

Abbildung 12.26: Code Snippet presentation Model