Namespaces

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

Einbinden eines Namespaces z.B. System.Collections:

```
xmlns:Collections="clr-namespace:System.Collections"
```

Data Validation

```
{Binding Path="whatever"  ValidatesOnExceptions="True"
  ValidatesOnDataErrors="True"} // Default: Rot Umranden
```

ErrorTemplate:

Bsp: Fehlertext neben der Box

```
<ControlTemplate x:Key="TBErrorTemplate">
  <DockPanel>
    <AdornedElementPlaceholder x:Name="adornedElement"/>
    <TextBlock Text="Hier ein Fehlertext! "FontSize="12"
      Foreground="Red"  Margin="5 0 0 0"/>
  </Dockpanel>
</ControlTemplate>
<TextBox Validation.ErrorTemplate="{StaticResource TBErrorTemplate}"
  Text="{Binding... ValidatesOnExceptions="True"
  ValidatesOnDataErrors="True"}"/>
```

Eigene Validation Rules

```
public class MyValidationRule : ValidationRule{
public int maxLength {get; set;}
public override ValidationResult Validate(object val, CultureInfo CI){
  if(val.ToString().Length > maxLength)
    return new ValidationResult(false,"");
  return ValidationResult.ValidResult; }
}
```

=> XAML:

```
<TextBox Validation.ErrorTemplate="...">
  <TextBox.Text>
    <Binding Path="Property">
      <Binding.ValidationRules>
        <RichtigerNamespace:MyValidationRule maxLength="10"/>
      </Binding.ValidationRules>
    </Binding>
  </TextBox.Text>
</TextBox>
```

Bindings:

!Nur public Propertys bindbar!

Modes: TwoWay, OneWay(Kein UpdateSourceTrigger), OneWayToSource(Schreibend), OneTime(InitialiseComponent)

UpdateSourceTrigger: PropertyChanged, LostFocus(default bei Textbox)

```
XAML: z.B. <Textbox Text="{Binding ElementName=box1, Path=Text}"/>
```

```
C#: Binding b = new Binding("Text"); // QuellProperty
b.Source = box1; //Objekt
box2.SetBinding(TextBox.TextProperty, b)
```

Binding auf eigene Klassenobjekte mit automatischem aktualisieren:

z.B.

```
using System.ComponentModel;
class TestData: INotifyPropertyChanged {
  public event PropertyChangedEventHandler PropertyChanged;
  string myProperty;

  public string MyProperty {
    get{return myProperty;}
    set{ myProperty = value;
      OnPropertyChanged("MyProperty");}
  }

  protected void OnPropertyChanged(string pname) {
    PropertyChangedEventHandler handler = PropertyChanged;
    if (handler != null)
      handler(this, new PropertyChangedEventArgs(pname));
    }
  }
```

Datenkontext

```
DataContext="{Binding RelativeSource={x:Static
  RelativeSource.Self}}"     //Kontext ist Code-Behind
```

Routed Events

Reihenfolge: zuerst Tunneling, dann Bubbling;

Direkte Events: nicht weitergereicht (z.B. Click)

Bubbling Weiterreichen nach Oben (zum Window) z.B. MouseLeftButtonDown

Tunneling Weiterreichen nach Unten (zum Button) z.B. PreviewMouseLeftButtonDown

Styles

```
<Window.Resources>
   <Style x:Key="MyStyle"  TargetType="{x:Type Button}">
      <Setter Property="Background"  Value="Green"/>
   </Style>
</Window.Resources>
<Button Style="{StaticResource MyStyle}"...
```

allgemein

Stapelverarbeitung: synchrone Abläufe, sequentiell; (Transaktionsbasiert)
Ereignissgesteuert: asynchrone Abläufe, parallel; (Ereignis -> Aktion)

Bedienungsparadigmen:

Horizontal: Ablauforientiert, viele einfache Dialoge z.B. Auskunftssystem
Vertikal: zentraler Bearbeitungsgegenstand, wenige komplexe Dialoge z.B. GIMP

historie

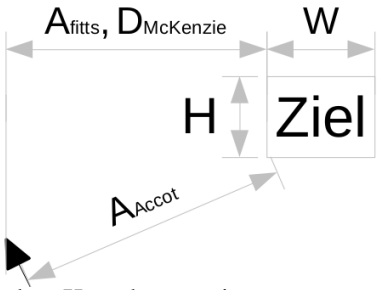
Command Line Interpreter -> Text User Interface -> Graphical User Interface
Memex: System zur Verwaltung von Wissen (analog)
Sketchpad: Objektorientierte Schnittstelle für CAD
Xerox: GUI, Mehrere Applikationen, mehrere Fenster
Macintosh: Xerox+Menüleiste

Hick-Hyman

Reaktionszeit $RT = a + b\log_2(n+1)$
a,b konstante, n: Auswahlmöglichkeiten
Relevant in Menüs, bei gelernten Einträgen (Unbekannte Einträge: lineare Suchzeit)

Fitts

$Zeit = a + b \cdot ID$
Fitts: $ID = \log_2(\frac{2A}{W})$
MacKenzie: $ID = \cdot \log_2(\frac{D}{W} + 1)$
Accot: ID = $\log_2(\sqrt{(\frac{D}{W})^2 + \mu(\frac{D}{H})^2} + 1)$



=> Over-/Undershooting: Verfehlen des Zieles, Korrektur nötig
Konkurrenz der GUI-Elemente (kleine Ausgabe)
stärkere Abweichung für kleine Ziele
Unterstützung:
- Vergrößern Ziel/Zeiger
- Cursor zum Ziel/ Ziel zum Cursor bewegen

=> evtl. irritierend, ungewollte Auswahl

Interaktionstechniken

Sprachinteraktion

Varianten: Kommandosprachen, Textuelle Suche, Natürlich-sprachlich
Audiosignal, Tastatureingabe

Kommandosprachen

synthetische Sprachen: effizient Implementier-/Benutzbar
Kriterien: eindeutige kurze Schlüsselwörter, konsistent, nahe Natürlicher Sprache
Autovervollständigung, Hilfssysteme, History-Funktion sinnvoll

Natürlich-sprachlich

Vorteil: kein Lernaufwand(intuitiv)
Nachteil: effiziente Implementierung schwer, Mehrdeutigkeiten, Dialekte, Unterscheidung Eingabe/keine Eingabe

Menüs

hierarchische Gruppierung atomarer Kommandos
Strukturierung: kurze einheitliche Kommandos, häufigkeit, Gruppen, Alphabet ...

Pulldown

Pfad bleibt sichtbar, Untermenüs
Vorteil: Übersichtlich, Einheitliche Auswahl, Kein Schreibaufwand, Leicht Erlernbar
Nachteil: Unübersichtlich bei Vielen Einträgen, Suchproblem(Anfänger), Geschwindigkeit(fortgeschritten), Sprachabhängig

Popup

Lokale Kontextmenüs => Nah am Cursor

Pie

Kreisförmige Anordung => gleiche Nähe zum Cursor, Platzbedarf (v.A. Icons),
Mobile:Half-pie, Ähnlich: Marking Menüs (Joystickprinzip)

(Semi-)Transparente Menüs

geringe Verdeckung vs geringe Lesbarkeit

Befehlstasten/Hotkeys

Direkter Aufruf schneller als Menü; Kürzel sichtbar im Menü, Redundant zum Menü, Guidelines, kritische Kommandos(löschen) mit Nachfrage, 1-Hand vs 2-Hand

Direkt Manipulative Benutzerschnittstellen

Direktes bearbeiten eines Objektes (z.B. Drag & Drop)
Vorteil: sprachunabhängig, ohne schreibkenntnis möglich (Kinder)
Nachteil: Realisierungsaufwand, u.U. ungenaue Handhabung

Agentenbasierte Interaktion

Delegieren von Aufgaben ans System => Benutzermodell, Aufgabenmodell(z.B. aus Historie) notwendig, Kontrolle und Undomechanismen
aktiv oder vom Benutzer aktiviert, Vorschläge oder direkt Ausführen,

Wizardinteraktion

schrittweises Führen durch komplexe Aufgabe, Freiheiten in den Einzelschritten

Eingabegeräte

Natural User Interface: Eingabegerät nuss nicht Erlernt werden
Intuitive User Interface: Erlernen während Bedienung
Räumliche UI: Präsentation und Interaktion in 3D => 2d u.U. problematisch
Haptic User Interface: Bewegungen werden Erkannt, haptische Rückmeldungen (Vibration, drehung)
Brain Computer Interface: Steuerung durch Gehirnströme
Gestenbasiert: abstraktion eines Eingabegerätes (Touchscreen, Kamera,...);
Vorteile: Vielseitig Einsetzbar, komplexe Befehle möglich
Nachteile: schlechte Erkennungsrate, Performancelastig
Erkennung (Rubine): bilde Eigenschaftsvektor (aus z.B. Diagonale der Begrenzungsbox, Winkel der Diagonale, Dauer der Geste, ...) und suche passendste Geste
Tastatur: Eingabe von Symbolen oder Kommandos (Hotkeys), Anwendungsspezifisch oder -unabhängig, Taktil oder virtuell, evtl. konfigurierbares Layout
Zeigegeräte:
Direkt: kaum Lernaufwand, Verdeckte Ausgabe, Ermüdung, Präzisionsprobleme

- Touchscreen: evtl. mit Druckstärke, Multitouch
- Stifteingabe: handschriftgeeignet (OCR), Präziser als Finger

Indirekt: Lernaufwand, Präzision, billiger, absolut&relative Positionierung möglich

- Maus: ermüdungsarm, hohe Auflösung
- Trackball: keine Fläche benötigt, fest verbaut (robust im öffentichen Raum), drag&drop schwieriger
- Touchpad: kleine Fläche, fest verbaut (anschluss von Peripherie schwer möglich)
- Joystick: geeignet für Steuerung und Verfolgung von Objekten
- Trackpoint: ungenau, oft direkt in Tastatur (kombination, direkter zugriff)
- 3D-Interaktion: virtuelle Räume, CAD; Tischgebunden oder frei bewegbar (Datenhandschuh)

Klassifikation

Mackinlay

T= (M,In,S,R,Out,W)
M: Operation, z.B. Rotation um X, Verschieben in Y, ...
In: Eingabebereich
S: Status des Geräts
R: Funktion von In nach Out
Out: Ausgabebereich
W: innere Funktionsbeschreibung (default:)
z.B. Lautstärkeregler = (Rot z,$[0°,270°]$,$0°$,id,$[0°,270°]$,{})
Radioknopf = (Rot z,$[0°,270°]$,$0°$,id,$\{0°,120°,140°\}$,{}) (vgl. oben, Rastet ein)

Krauß

- Nicht koordinatengebend: z.B. Taste, Geste ...
- Koordinatengebend
  1. Dimensionalität: 1-dimensional (Schieberegler, Drehknopf), 2-dimensional (Maus), mehrdimensional (Datenhandschuh)
  2. direkt (Touchscreen) VS indirekt(Maus)

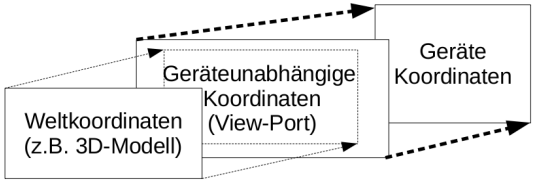
Absolute Positionierung: 1:1 Umrechnung der aktuellen Position
Relative Positionierung: Umrechnung der Veränderungen in Zeigerbewegungen
Control-Display-Gain: durch Bewegung des Zeigegerätes erzeugter Effekt
Gain klein: Fein-, Gain hoch: Grobpositionierung $\Rightarrow$ Lösung: Dynamische Anpassung
Konflikte: Genauigkeit vs Abtastrate, Verzögerung vs Abtastrate
Abtastrate: Erfassung des Zustandes in diskreten Zeitabständen
Verzögerung: Zeit zwischen Abtasten und Änderung Bildschirm
Genauigkeit: Abweichung zwischen gemessener und tatsächlicher Position
Mehrere Eingabegeräte:
Redundant: verschiedene (alternative) Eingabegeräte für gleiche Eingabe
Komplementär: abhängige Geräte: kopplung der Geräteinputs zu einer Eingabe
unabhängige Geräte: Eingaben steuern verschiedene Anwendungsteile

Ausgabegeräte

stereoskopische Displays: verschiedene Bilder für rechtes/linkes Auge;
u.U. Hilfsmittel(Brille) nötig, sonst Autostereoskopie, Nicht für alle geeignet
Holographische Displays: echtes 3d Display, z.B. Laser
Taktile Displays: Fühlbares Interface, z.B. Reibungsbeeinflussung bei Touch

Endlicher Automat mit Ausgabe
Tupel (X,Y,S,s_0,δ,σ)
X: Eingabealphabet, Y: Ausgabealphabet, S: Zustandsmenge, s_0: Startzustand,
σ: Ausgabefunktion (bei Mealy SxY -> Y), (bei Moore S -> Y)
Moore: Ausgabe bei Zustandseintritt (z.B. N/0)
Mealy: Ausgabe bei Kantenübergang (c/a)
Zustandsautomat: Beschreibung von Eingabe/Ausgabe
(Explizit: Automat an Benuterschnittstelle sichtbar,
implizit: geht aus Verhalten hervor)

Ausgabemodell:
GeräteKoordinaten (pixel),
Geräteunabhängige Koordinaten (z.B. in mm) wegen verschiedener Auflösungen,
Weltkoordinaten: Werte der Daten aus Modell (z.B. in Nanometern, Lichtjahren)



Clipping: entfernen nicht sichtbarer Objekte (Weltkoordinaten → Geräteunabhängige)

Window Manager:
Verschieben, Skalieren, Öffnen, Schließen von Fenstern,
Umsetzen von Platzierungsstrategien(Versuch anordung zu optimieren)
Bei zu großen Inhalten: Scrolling, Panning (v.A. 2D) Zooming(vgl. Google-Maps)

Virtuelle Desktops:
viel Zeit für Verwaltung der Oberfläche, zu kleine Bildschirmfläche;
=> Raum-Metapher: Gruppenbildung von Fenstern (Raum), wechsel zwischen Räumen
Alternativ: Mehr Ausgabegeräte, Previews, Fischauge,
ZUI (Zoomable User Interface): Detailinfos bei höheren Zoomgraden

WIMP
WIMP: Windows, Icons, Menues, Pointers
Post-WIMP: In nicht Bürosituationen => Ergänzen/Auflösen von WIMP

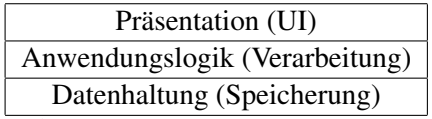
WIMP	**Post-WIMP**(Anti-Mac)	**Technologie** (Post)
Metaphor	Reality	Tangible UI
direct Manipulation	Delegation	Agents
See & Point	Describe & command	Natural Language UI
Consistency	Diversity	Bedingt durch Wechsel Ausgabegeräte
WYSIYG	Represent Meaning	"Letzter Versicherungsbrief"
User Control	Shared Control	Automatismen (z.B. Updates)
Feedback &Dialog	System handles Details	Agents
Forgiveness	Model User actions	Planerkennung / Vorschlag Alternativen
Aesthetic Integrity	Graphic Variety	Bedingt durch Wechsel Ausgabegeräte
Modelessness	Richer cues	Zusatzinformationen nach Verwendung

Icons:
Benutzerführung, ausnutzen Mustererkennung, sprachunabhängig
Arten: Repräsentativ(CD-Laufwerk), Abstrakt(Pfeile), Hybridvarianten
Richtlinien: Einfachheit/Klarheit, Verständlichkeit, Einprägsamkeit, Platzsparend, Kontrast zum Hintergrund, Unterscheidbarkeit vs Konsistenz
Toolbars: flexible Anordnung, konfigurierbar; meistens Sammlung von Icons

Architekturmuster
Motivation: Verbindung von Programmlogik und Darstellung => Schwierige Darstellungsänderung, Arbeitsteilung, Tests, wechsel der Ausgabegeräte, Redundanzen, Unübersichtlichkeit
User Interface Management System: Trennungsmechanismus zwischen Geschäftslogik und GUI (Trennung der Anliegen)
Übergangsdiagramme: (vgl UML) Darstellung von Interaktionsabläufen (Dokumentation etc.); Kantenbeschriftung: Eingabe/Ausgabe

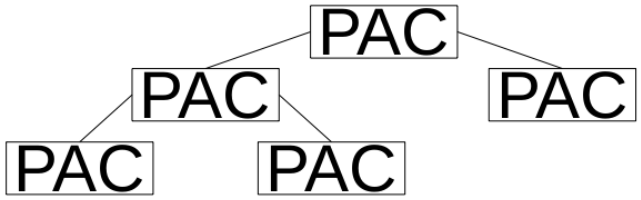
Schichtenarchitektur:
(strikt = Kommunikation nur mit darüber/darunterliegender Schicht)

3-Schichtenarchitektur (Van Dam):



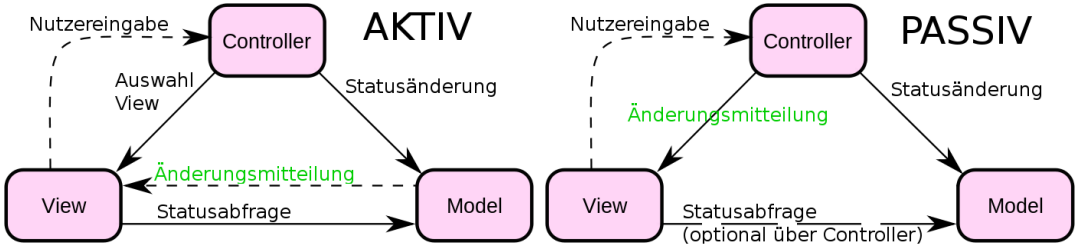
Seeheim:
(mit Bypass kein Schichtenmodell)

PAC:
hierarchische Elemente, jedes mit eigener Presentation, Abstraction, Control (unabhängig);
Vorteil: Multithreading einfach, Nachteil: Overhead



MVC:

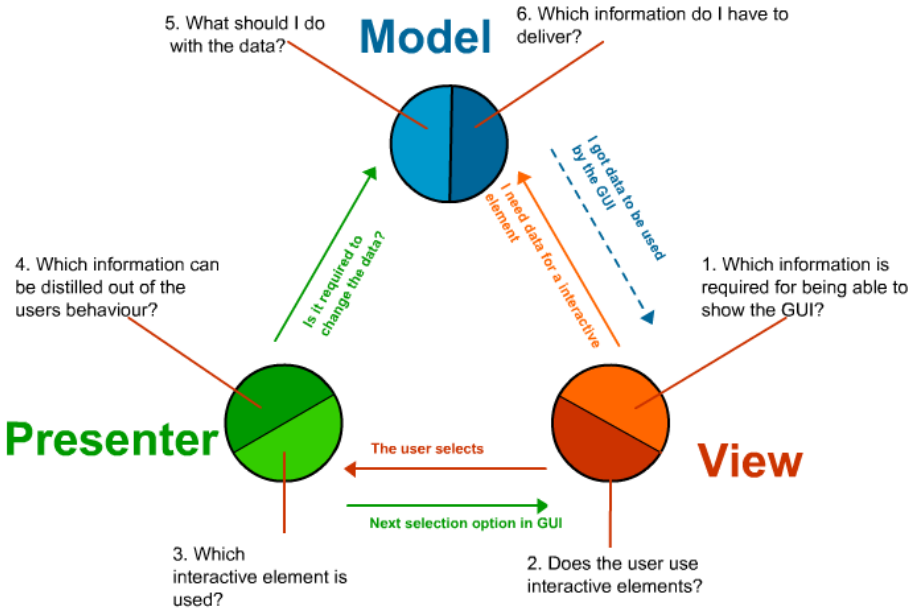
Model(Datenhaltung+Geschäftslogik) - View(Darstellung) - Control(Steuerung)
Vorteil: Model ohne View leicht testbar, Mehrere Controller einer View möglich
Nachteil: Mehraufwand, Trennung View-Control
passive MVC: Model reagiert auf anfragen, Kommunikation über Control



Component-Based-View: redundanzen innerhalb der View
⇒ Zerlegen in Widgets mit eigener View (evtl. auch eigene Control)
Front-Controller: Für Webanwendungen: leitet Anfrage an zuständigen Controller weiter
Application-Controller: Zusätzlicher Controller zur Verwaltung von Abhängigkeiten zwischen Seiten (reihenfolge etc.), mächtiger als Front-Controller

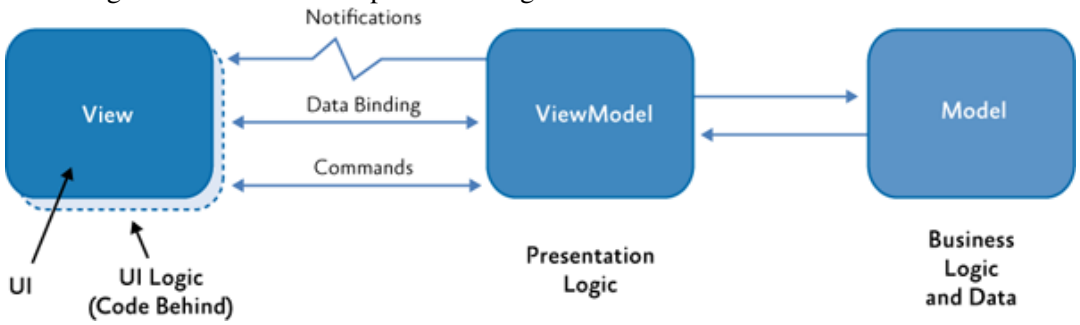
Model-View-Presenter:

Vorteile: sehr lose Kopplung der View, mehrere Presenter pro View möglich



MVVM:

Programmierer und Designer können getrennt arbeiten(XAML)
View: Umfasst XAML und Code-Behind der XAML;
Definition von Fenster und Widgets, UI-Interne Logik
Schnittstelle: ViewModel ist DataContext der View
Die View bindet auf Properties und Commands des ViewModel
ViewModel: Präsentationslogik
Koordiniert zwischen View und Model: Konvertiert, verändert, validiert Daten
Schnittstelle: Referenziert Model,
Implementiert Properties und Commands auf die View bindet;
Informiert (i.A. View) mittels Notification-Events
Model: Kapselt Daten der Anwendung und Geschäftslogik
sichert Konsistenz und Validität der Daten durch Geschäftslogik
Schnittstelle: Keine direkte Referenz zu View/ViewModel
Änderungen im Model werden per Events signalisiert



Webbasierte Systeme:
http zustandslos, v.a. Transaktionsbasiert;
Mehrnutzerszenarien, Benutzer/Anwendung evtl. Anonym
Architektur: Thickclient(kann lokal laufen) vs Thinclient(nur View)

Böswillige Benutzer: (Menschen/Bots)
Vandalismus, Trolling, Spam, Fehlinformation,
DoS, SQL-Injection, Session-Hijacking
vs Moderation, Automatische Prüfung, Turing Test(Captcha)

Böswillige UIs
tut mehr/anders als vom Benutzer gewollt

Zwang
Verwirrung
Ablenkung
Ausnutzen von Fehlern des Benutzers
Zusätzlicher Interaktionsaufwand
Unterbrechung
Navigation manipulieren
Verschleierung
Einschränken der Funktion
Erschrecken
Irreführen

Frameworks:
Struktur zur Unterstützung (z.B. Erstellung von Code-Skeletten)
Vorteile: Wiederverwendung von Expertenwissen, Langfristige Zeitersparnis, Systematisch Testbar, Standardisierung, Fehlervermeidung
Nachteile:Entwicklung zeitaufwändig, Einarbeitungsaufwand, Dokumentation und Wartung aufwändig, Fehlersuche wird erschwert, kombinierung von Frameworks schwierig

Kategorien:
Blackbox (Komponenten schon definiert, werden vom Programmierer zusammengefügt)
Whitebox (Ableitung eigener Komponenten von Basiselementen)
Graybox (Mischung aus Black- und Whitebox)

Support: Systemdienste(DirectX,OpenGL)
Domain: Funktionalität für Speziellen Bereich (z.B. Semantic Web)
Application framework: Funktionalität für komplette Anwendung (WPF)

Entwicklung interaktive Systeme im Software-Entwicklungsprozess
Hinzufügen von Elementen der Benutzerschnittstellen
Detaillieren von Dialoginhalten, -verhalten und -kontext sowie das Verknüpfen mit ihren Anforderungen
Anpassung an Änderungswünsche des Benutzers
Vervollständigen von Benutzerschnittstellen
Klassifizieren der Bestandteile der Benutzerschnittstelle
Protokollierung der Benutzerschnittstellenstruktur und der Benutzerinteraktionen
Kommunikation zwischen den Stakeholdern, d.h. Entwickler, Benutzer und Anwender
Evaluation der bisherigen Benutzerschnittstelle
Probleme:
Abstraktheit (Modelle für Benutzer schwer deutbar),
Mehrdeutigkeit (Kommunikation Benutzer $\rightarrow$ Entwickler),
Fehlende Erlebbarkeit(Konsequenzen der Änderungen Benutzer nicht klar),
Benutzerziele (unterschiedliche Benutzergruppen, Interaktionsstrategien),
Komplexität interaktiver Systeme (viele mögliche Wechselwirkungen)
Entwicklungsprinzipien:
ISO 9241 Grundsätze: Aufgabenangemessenheit Selbstbeschreibungsfähigkeit Erwartungskonformität Fehlertoleranz Steuerbarkeit Individualisierbarkeit Lernförderlichkeit
Prinzipien nach Gould:
Konzentration auf Benutzer (Beobachtung, Befragung von Repräsentativen Benutzern)
Randbedingungen prüfen (Umfeldanalyse; Tätigkeitsanalyse, willig/unwillig)
Frühes und Kontinuierliches Testen
Iteratives Design: in jeder iteration Ziele Identifizieren, Gestaltungsvorschlag, Testen; Mehrere Iterationen einplanen, änderungsfreundliche Systemarchitektur, Mockups und Papier-Prototypen
Integriertes Design(Ganzheitliche Betrachtung des Systems schon in der Entwicklungsphase)