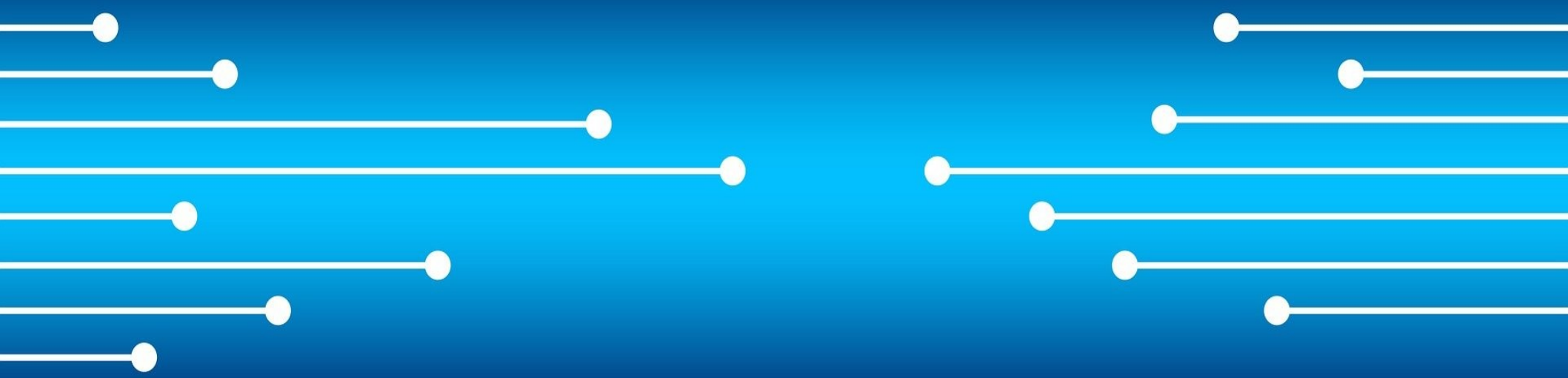


Datenverarbeitung

Teil des Moduls 5CS-DPDL-20



Prof. Dr. Deweß

Thema 1



Gestaltungsrichtlinien

- Was ist das?
- Stil: Schnittstelle
- Stil: Quellcode

Gestalten



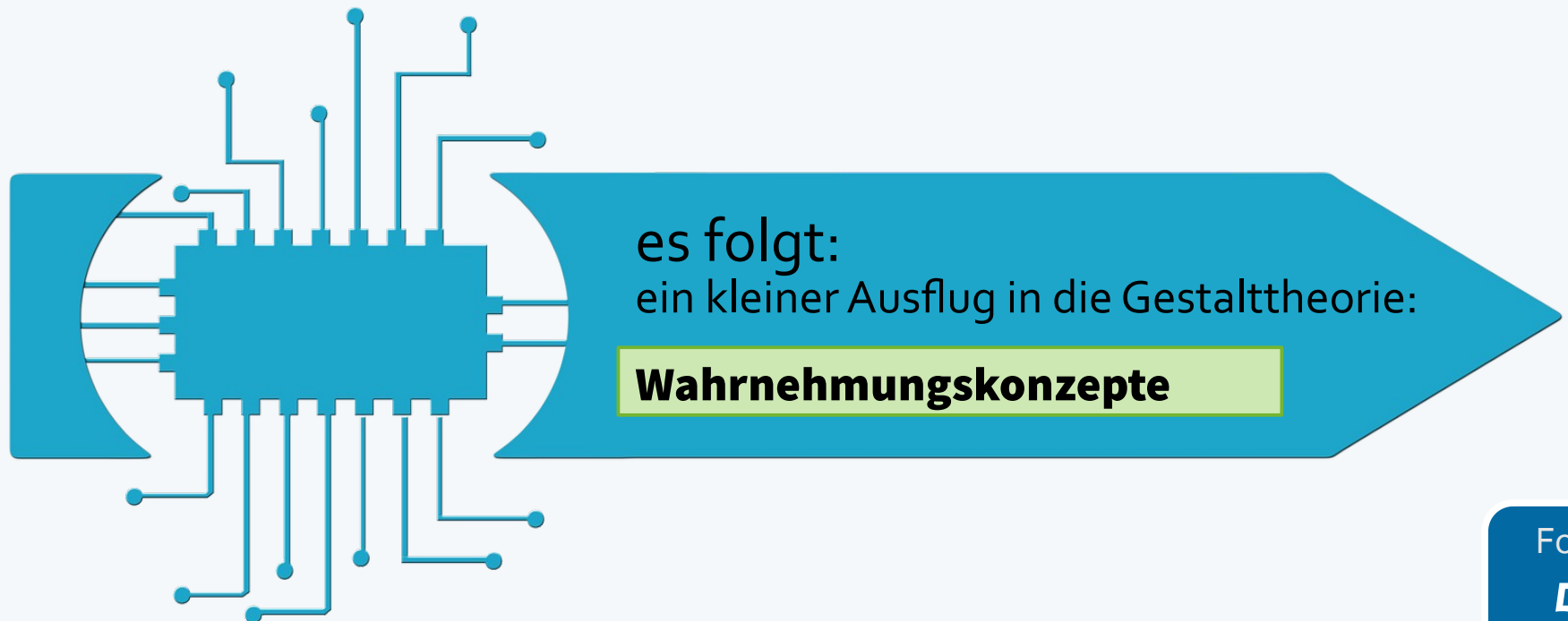
Gestaltungsrichtlinien



- Erscheinungsbild und Beschaffenheit des Objektes
- enthält oft „muss“, „soll“ und „kann“-Bestimmungen

Einschub: Gestalttheorie

"Das Ganze ist etwas anderes
als die Summe seiner Teile."



Einschub: Gestalttheorie

1. Prägnanz

Sinneseindrücke werden in „Figur“ und „Grund“ eingeordnet und dementsprechend wahrgenommen.

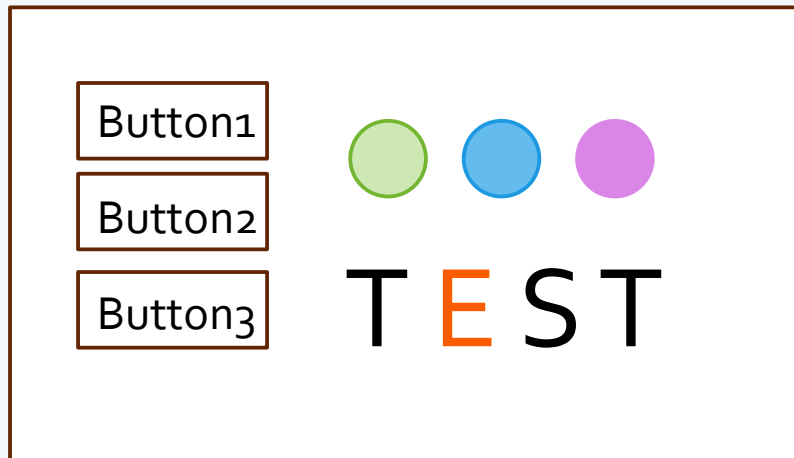
Button1 Button2 Button3 Button4

Button1 Button2 Button3 Button4

Einschub: Gestalttheorie

2. Ähnlichkeit

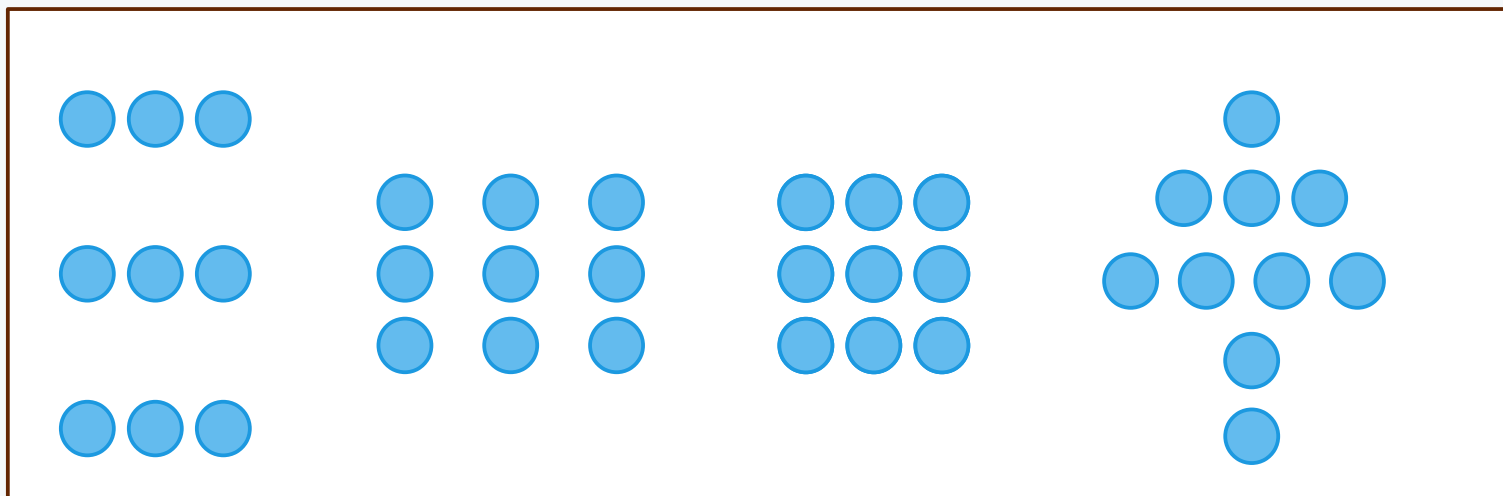
Je ähnlicher sich Objekte sind, desto eher werden sie als Einheit wahrgenommen.



Einschub: Gestalttheorie

3. Nähe

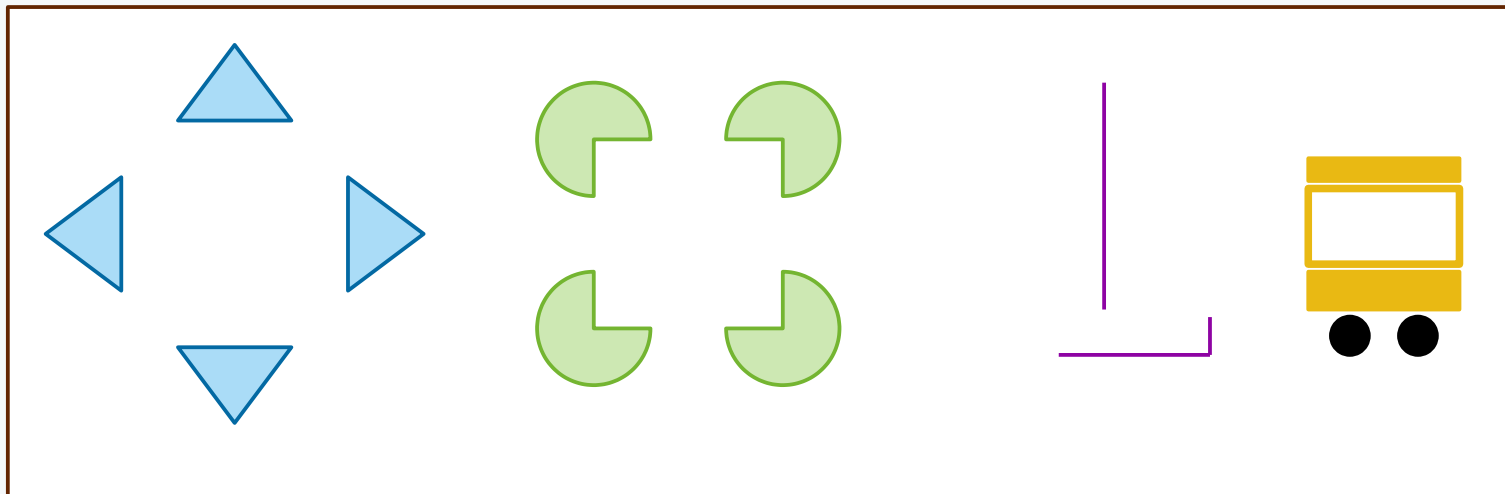
- Menschen versuchen, nah beieinander liegende Objekte als Einheit wahrzunehmen
- nach der „Einheitsbildung“ wird versucht, dem „Gesamteindruck“ einen Sinn zu geben
- als „Einheit“ wahrgenommene Objekte konkurrieren nicht um Aufmerksamkeit und bringen Ruhe in den Eindruck



Einschub: Gestalttheorie

4. Geschlossenheit

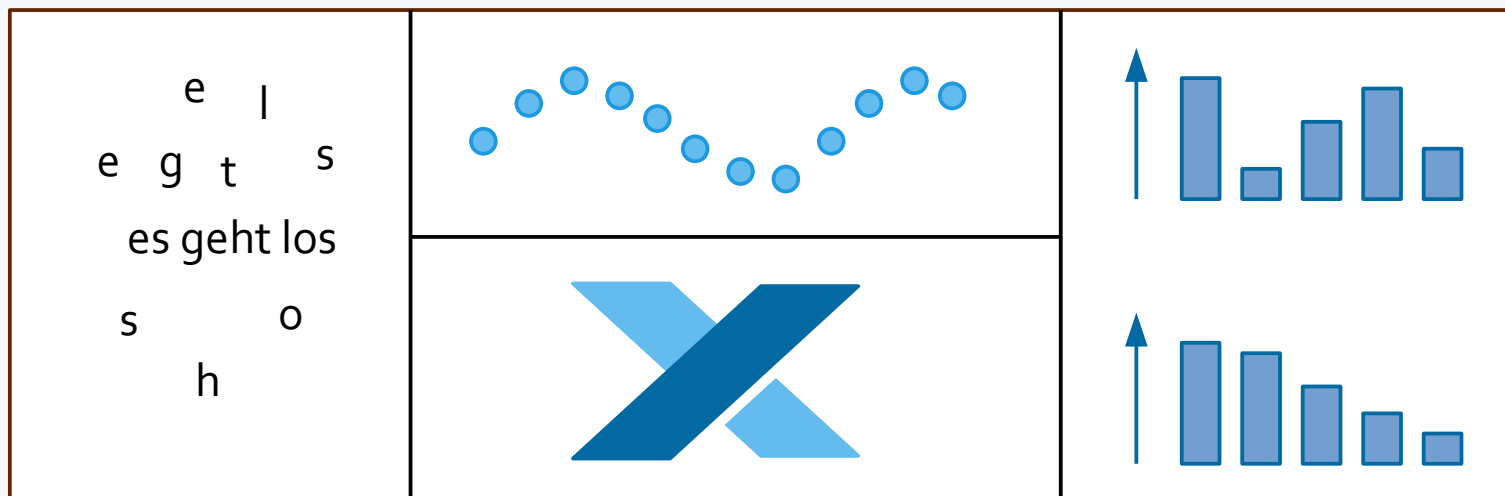
- Menschen versuchen, Zusammenhänge zu entdecken
- wenn Objekte scheinbar eine geschlossene Form ergeben, nehmen wir diese als derartige Einheit wahr
- unvollständige Einheiten werden über unser visuelles Gedächtnis ergänzt



Einschub: Gestalttheorie

5. Fortsetzung

- einheitliche Objekte mit erkennbarer Fortsetzung werden fokussiert
- Objekte, die entlang einer gemeinsamen Fluchtlinie angeordnet sind, werden als zusammengehörig interpretiert
- Informationen lassen sich schneller wahrnehmen, wenn sie entsprechend gut strukturiert sind

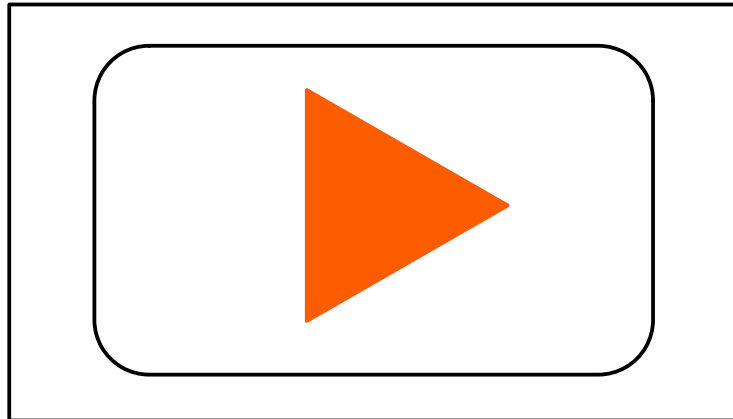


Einschub: Gestalttheorie

6. ... (tausend mehr)

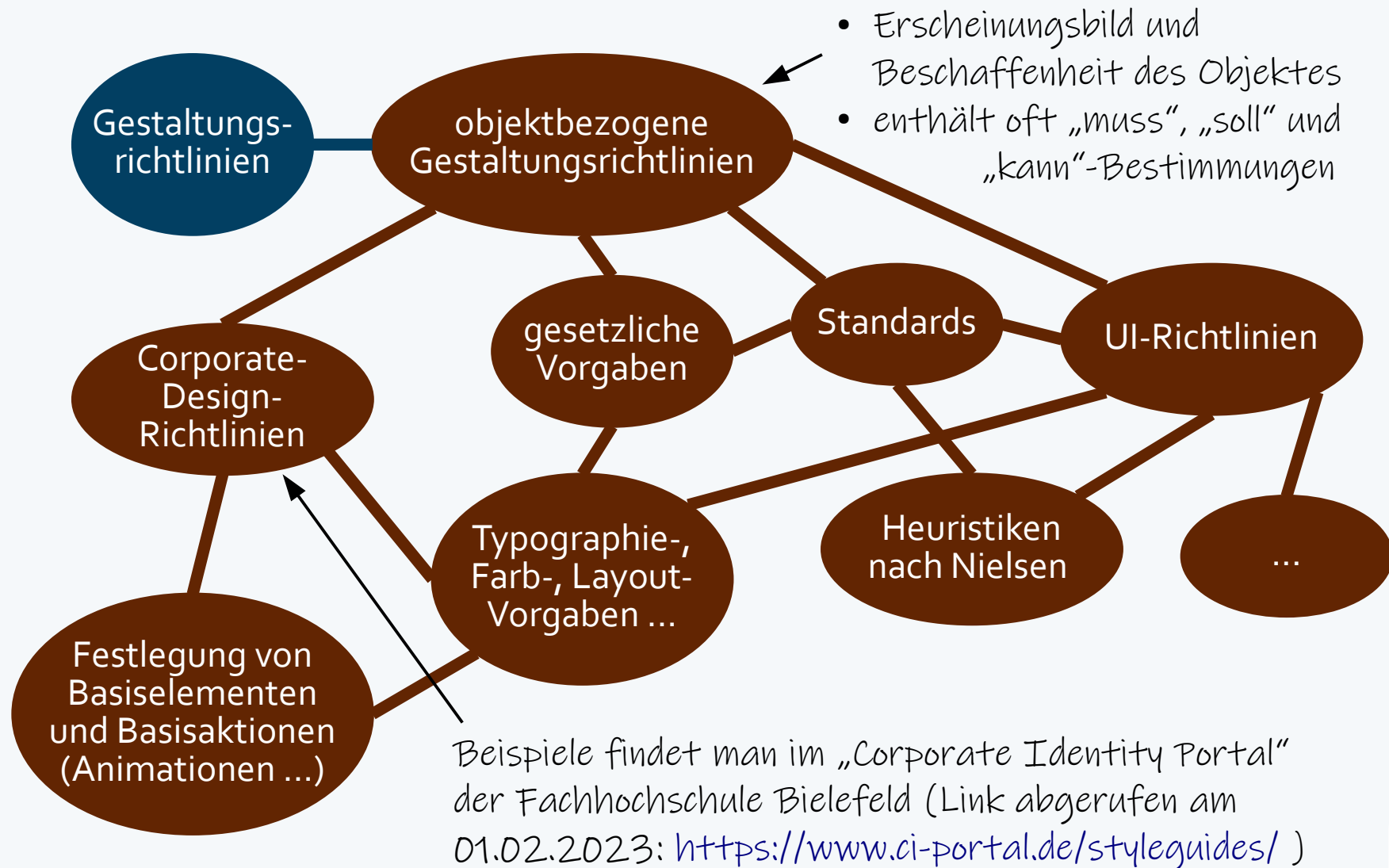
- einfache Sachverhalte werden besser wahrgenommen
- zu viele Auswahlmöglichkeiten auf einmal überfordern
- widersprüchliche Angaben sind zu vermeiden

Experiment „Stroop-Effekt“



Quelle: Fellingner, Leo (2014, 13. April). *Stroop Effekt*. [Video]. YouTube.
<https://www.youtube.com/watch?v=kzcOFtOXllc> (abgerufen am 02.02.2023)

Gestaltungsrichtlinien



Benutzerschnittstellen (rechnergestützt)

Kommandozeilen („Command Line Interface“ – CLI)

- ursprüngliche Interaktionsmöglichkeit
- Befehlseingabe per Tastatur

Zeichenorientierte Schnittstellen („Text User Interface“ – TUI)

- Übergang von der Kommandozeile zur graphischen Benutzeroberfläche
- Bildschirm wird jetzt flächig und nicht mehr nur zeilenorientiert genutzt
- Befehlseingabe meist menügeführt per Tastatur (z.B. bei Bootloadern, BIOS-Setup)

graphische Benutzeroberflächen („Graphical User Interface“ – GUI)

- derzeit üblicher Standard
- besitzen meist komplizierte, flächige Eingabemöglichkeiten, die mit der Maus oder anderen geeigneten Eingabegeräten angesprochen werden können

Benutzerschnittstellen (rechnergestützt)

natürliche Benutzerschnittstellen („Natural User Interface“ – NUI)

- Weiterentwicklung graphischer Benutzeroberflächen
- Interaktion durch Reaktion auf Finger- und Handbewegungen
- **Beispiele:**
Touchpad, Eingabecontroller von Spielekonsolen, OmniTouch-Projekt von Microsoft

sprachbasierte Benutzerschnittstellen („Voice User Interface“ – VUI)

- Kommunikation mittels gesprochenem Wort
- setzt Spracherkennungsmöglichkeiten voraus
- **Beispiele:**
Spracheingabeassistenten von Betriebssystemen, Sprachdialogsysteme

Benutzerschnittstellen (rechnergestützt)

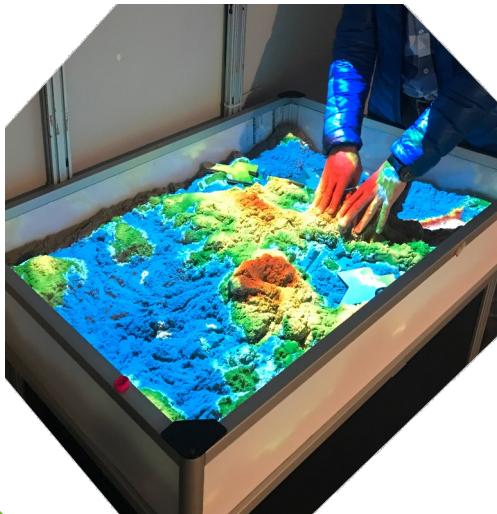
wahrnehmungsgesteuerte Schnittstellen („Perceptual User Interface“ – PUI)

- Verbindung von sprachbasierten und graphischen Benutzerschnittstellen mit einer elektronischen Gestenerkennung
- z.B. Kopfbewegungen für Cursorbewegungen und Blinzelbewegungen für Mausklicks
- Anwendungsbereiche hauptsächlich bei der Unterstützung behinderter Menschen und im Gaming-Bereich

Benutzerschnittstellen (rechnergestützt)

gegenständliche Benutzerschnittstellen („Tangible User Interface“ – TUI)

- haptisch erfahrbare Interaktion mit dem Rechner mittels physischer Gegenstände
- Interaktion im dreidimensionalen Raum möglich
- Kopplung von Steuerung und Repräsentation, da die Schnittstellen zugleich Ein- und Ausgabegerät repräsentieren



Beispiel:

SandScape – eine TUI für die Gestaltung und das Verständnis von Landschaften mittels sandbasierter Computersimulation

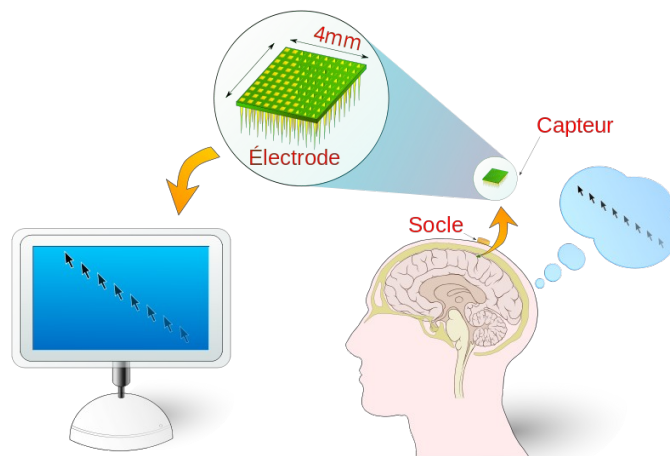
Quelle: Ilya.osipov

(<https://commons.wikimedia.org/wiki/File:SandScape.jpg>),
<https://creativecommons.org/licenses/by-sa/4.0/legalcode>

Benutzerschnittstellen (rechnergestützt)

Gehirn-Computer-Schnittstellen („Brain Computer Interface“ – BCI)

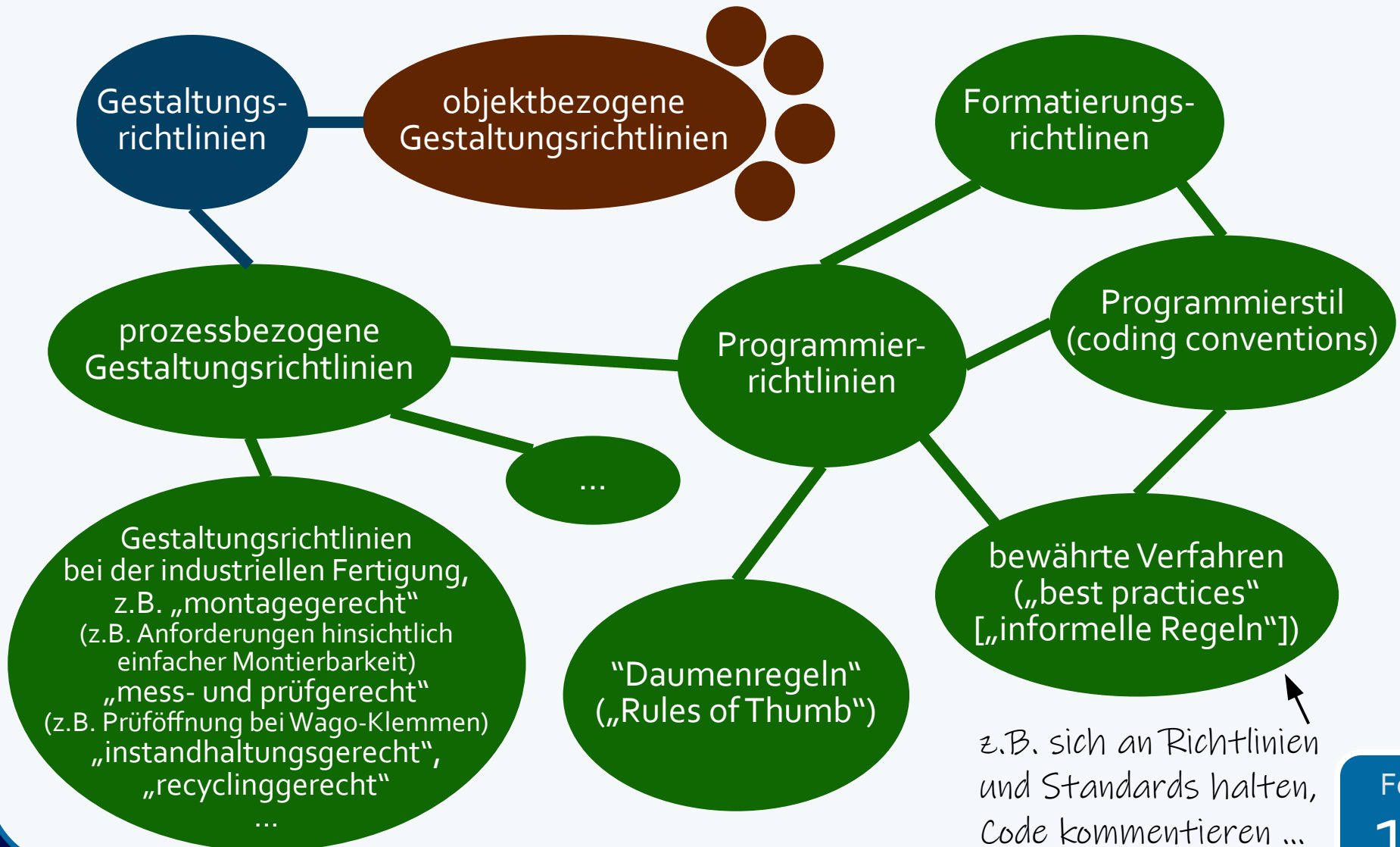
- Bedienung über Gedanken (Hirnstrommessung durch Elektroden)
- großer Anwendungsbereich bei der Unterstützung körperlich behinderter Menschen
- Achtung: ethische Aspekte („Persönlichkeitsabfragen“ und „Persönlichkeitsmanipulation“ sind möglich) berücksichtigen



Schema einer
Gehirn-Computer-Schnittstelle

Quelle: Balougador
(<https://commons.wikimedia.org/wiki/File:InterfaceNeuronaleDirecte-fr.svg>),
<https://creativecommons.org/licenses/by-sa/4.0/legalcode>

Gestaltungsrichtlinien



Programmierrichtlinien – Sinn?

Verbesserte Lesbarkeit

- effizientere Zusammenarbeit verschiedener Entwickler
- neue Entwickler können sich schnell einarbeiten

Konsistente Qualität und Einhaltung von gesetzlichen Standards

- sicher (verursacht keine Schäden)
- zuverlässig (macht bei jeder Anwendung, was es soll)
- testbar (auf Code-Ebene)

Zugriffsschutz

- Software kann nicht gehackt werden

Geringere Entwicklungskosten und kürzere Markteinführungszeiten

- schnelle Weiterentwicklung
- gute Wartbarkeit
- bessere Portabilität

Programmierrichtlinien decken oft ab:

Dateiorganisation, Einrückung, Leerräume, Kommentare, Deklarationen, Anweisungen, Namenskonventionen, Programmierprinzipien ...

Programmierrichtlinien (sprachunabhängig)

- schreibe einfach lesbar (also konsistent und mit Einrückungen, Leerzeichen)
- dokumentiere für interne Weiterentwicklung und für externe Benutzung

Programmierrichtlinien (sprachunabhängig)

- schreibe einfach lesbar (also konsistent)
- dokumentiere

Dokumentation

- beschreibe Variablen, Klassen, Methoden ...
- erkläre „Was?“ und „Warum?“ - nicht „Wie?“ (das sieht man am Quellcode)
- konzentriere dich auf Dinge, die nicht im Quellcode ersichtlich sind

Programmierrichtlinien (sprachunabhängig)

- schreibe einfach lesbar (also konsistent und mit Einrückungen, Leerzeichen)
- dokumentiere für interne Weiterentwicklung und für externe Benutzung
- selbsterklärender Quellcode ist besser als komplizierter Quellcode mit Kommentaren
- vermeide unnötige Dopplungen (Stichwort: DRY – Don't Repeat Yourself)
- gestalte Methoden wiederbenutzbar und möglichst kompakt
- vermeide „magische Zahlen“

Programmierrichtlinien (sprachunabhängig)

- schreibe einfach lesbar (also konsistent)
- dokumentiere für interne Weiterentwicklung
- selbsterklärender Quellcode ist besser
- vermeide unnötige Dopplungen (Duplikate)
- gestalte Methoden wiederbenutzbar
- vermeide „magische Zahlen“

„magische Zahlen“

- Zahlen mit Bedeutung als Konstante festlegen
- dann immer Konstante benutzen

```
final int ANZAHL_STUDIERENDE = 32;  
for (int i = 0; i < ANZAHL_STUDIERENDE; i++) {  
    ...  
}
```


statt

```
for (int i = 0; i < 32; i++) {  
    ...  
}
```

Programmierrichtlinien (sprachunabhängig)

- schreibe einfach lesbar (also konsistent und mit Einrückungen, Leerzeichen)
- dokumentiere für interne Weiterentwicklung und für externe Benutzung
- selbsterklärender Quellcode ist besser als komplizierter Quellcode mit Kommentaren
- vermeide unnötige Dopplungen (Stichwort: DRY – Don't Repeat Yourself)
- gestalte Methoden wiederbenutzbar und möglichst kompakt
- vermeide „magische Zahlen“
- pro Einsatzzweck eine Variable
- vielsagende Namen, keine unnötigen Abkürzungen

Programmierrichtlinien (sprachunabhängig)

- schreibe einfach lesbar (also konsistent)
- dokumentiere für interne Weiterentwicklung
- selbsterklärender Quellcode ist besser
- vermeide unnötige Dopplungen (Duplikate)
- gestalte Methoden wiederbenutzbar
- vermeide „magische Zahlen“
- pro Einsatzzweck eine Variable
- vielsagende Namen, 

Hinweise z.B. für Variable

- ganze, vielsagende Wörter benutzen, z.B.

konto statt kto
kontoNummer statt nummer
- nur für Schleifenvariablen und ähnlich lokale Variable sind i, j, k ... in Ordnung

Programmierrichtlinien (sprachunabhängig)

- schreibe einfach lesbar (also konsistent und mit Einrückungen, Leerzeichen)
- dokumentiere für interne Weiterentwicklung und für externe Benutzung
- selbsterklärender Quellcode ist besser als komplizierter Quellcode mit Kommentaren
- vermeide unnötige Dopplungen (Stichwort: DRY – Don't Repeat Yourself)
- gestalte Methoden wiederbenutzbar und möglichst kompakt
- vermeide „magische Zahlen“
- pro Einsatzzweck eine Variable
- vielsagende Namen, keine unnötigen Abkürzungen
- vermeide globale Variablen
- liefere Ergebnisse und zeige sie nicht nur an
- kümmere dich um eine gute Ausnahmebehandlung, verschleppe Probleme nicht
- berücksichtige Laufzeit- und Speicherplatzbedarf (Programme können wachsen)
- denke zuerst an das „Drumherum“ (Datenanbindung, Schnittstellen, Testmöglichkeiten)

“Daumenregeln” (sprachunabhängig)

K.I.S.S. (Keep It Simple, Stupid)

- Methoden sollten GENAU EINE Aufgabe erfüllen
- Methoden sollten so kurz sein, dass man sie gut inspizieren kann
- Ein-/Ausgabe und sonstige Programmlogik sollten nicht vermischt werden
- Problem sollten in Teilprobleme zerlegt werden
- fange einfach an, kompliziert wird es von allein

"Daumenregeln" (sprachunabhängig)

"Rule of Three"

- wiederhole Quellcode maximal einmal
- ab dem dritten Mal lagere diesen in eine eigene Methode aus

“Daumenregeln” (sprachunabhängig)

“Ninety-ninety rule”

„The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.“

— Tom Cargill, Bell Labs

- Softwareprojekte überschreiten oft ihren ursprünglichen Zeitplan
- Zeitplanungen sollten einfache und schwierige Aufgaben passend abbilden

"Daumenregeln" (sprachunabhängig)

K.I.S.S.	"Rule of Three"	"Ninety-ninety rule"	<h2>"Efficiency vs. code clarity"</h2> <p>„The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization – For experts only: Don't do it yet.“</p> <p>— Michael A. Jackson</p> <p>[zitiert in: Bentley, Jon Louis (1988). <i>More Programming Pearls: Confessions of a Coder</i>. S. 61]</p> <ul style="list-style-type: none">• Compiler können auch optimieren• für vermeintliche Effizienz sollte nicht auf Lesbarkeit des Quellcodes verzichtet werden• nur wenige Codestellen sind für Effizienzbetrachtungen wichtig und diese sollte zuerst identifiziert werden, bevor wahllos sinnlos optimiert wird
----------	-----------------	----------------------	--

"Daumenregeln" (sprachunabhängig)

K.I.S.S.	"Rule of Three"	"Ninety-ninety rule"	"Efficiency vs. code clarity"	<p>Regeln, für und gegen die umfangreich argumentiert wird, basieren oft nicht auf einem „MUSS“, sondern persönlichen Ansichten und Geschmacksunterschieden ... (und können daher nach eigenem Geschmack angewendet werden)</p> <p>Keine Regel ohne Ausnahme ...</p> <p>Keine Regel, die nicht aus wichtigem Grund auch einmal ignoriert werden darf ...</p>
----------	-----------------	----------------------	-------------------------------	--

Programmierrichtlinien (Link-Liste)

- Google Style Guides:
<https://github.com/google/styleguide>
- The GNU Coding Standards:
<https://www.gnu.org/prep/standards/>
- Linux Kernel Coding Style:
<https://www.kernel.org/doc/html/latest/process/coding-style.html>
- PEP 8 – Style Guide for Python Code:
<https://peps.python.org/pep-0008/>
- Microsoft Writing Style Guide:
<https://learn.microsoft.com/en-us/style-guide/welcome/>
- Linksammlung von Wahyu Kristianto auf GitHub:
<https://github.com/Kristories/awesome-guidelines>
- Liste von UI-Hersteller-Styleguides von ProContext:
<https://www.procontext.de/methodik-und-wissen/richtlinien-und-standards/ui-hersteller-styleguides.html>

Java-Namenskonventionen

- Klassennamen beginnen mit einem Großbuchstaben: Fahrzeug, Lampe, String
- Methodennamen benutzen CamelCase: macheMist(), fahreRueckwaerts(), valueOf()
- Konstanten benutzen GROSSE_BUCHSTABEN und werden zuerst deklariert
- Variablennamen benutzen CamelCase und statische Variable werden meist zuerst deklariert
- Paketnamen werden meist (mit Ausnahmen) mit Kleinbuchstaben geschrieben: java.util, java.time, java.lang
- primitive Datentypen werden mit Kleinbuchstaben geschrieben: int, double, char



javadoc – Dokumentation aus spez. Kommentaren

- javadoc ist ein Werkzeug, um aus Deklarationen und spez. Kommentaren innerhalb von Java-Quelldateien dazugehörige HTML-Dateien zu erzeugen, die von außen zugängliche Klassen, Methoden usw. beschreiben
- javadoc kann auf einzelne Quellcode-Dateien oder sogar ganze Pakete angewendet werden
- javadoc kann noch weitere Quellen benutzen: Kommentardateien für Klassen und Module, Bilddateien, HTML-Dateien ...



Link zur offiziellen Beschreibung:

<https://docs.oracle.com/en/java/javase/19/docs/specs/javadoc/doc-comment-spec.html>

Programmierrichtlinien (Java)

- Code Conventions for the Java TM Programming Language (Original Sun Java Style Guide):
<https://www.oracle.com/java/technologies/javase/codeconventions-contents.html>
- Google's Java Style Guide:
<https://google.github.io/styleguide/javaguide.html>
- Android Open Source Project (AOSP) Style Guide:
<https://source.android.com/setup/contribute/code-style>
- Twitter's Java Style Guide:
<https://github.com/twitter/commons/blob/master/src/java/com/twitter/common/styleguide.md>
- The JavaRanch Style Guide:
<https://javaranch.com/style.jsp>
- Spring Java Format:
<https://github.com/spring-io/spring-javaformat>
- Java Code Styles - IntelliJ IDEA code style settings for Square's Java and Android projects::
<https://github.com/square/java-code-styles>

Programmierrichtlinien (Java)

Qual der Wahl ... - Überlegungen zur Festlegung

- Code Conventions for the Java Programming Language
<https://www.oracle.com/technetwork/java/javase/codeconvtoc-136057.html>
 - Google Java Style
<https://google.github.io/styleguide/javaguide.html>
 - Android Java Style
<https://developer.android.com/studio/write/java-style-tooltips>
 - Twitter Java Style
<https://twitter.github.io/styleguide/en/basic-conventions/programmer-requirements.html>
 - The Java Code Conventions
<https://www.oracle.com/technetwork/java/javase/codeconvtoc-136057.html>
 - Spring Java Style
<https://spring.io/projects/spring-core>
 - Java Code Conventions
<https://www.oracle.com/technetwork/java/javase/codeconvtoc-136057.html>
1. **Welches Ziel wird mit der Richtlinie verfolgt?**
Konsistenz im Team, Konsistenz mit anderen Teams, Benutzung einer bisher verwendeten Richtlinie und bisheriger Werkzeuge dafür?
 2. **Wo wird Quellcode angeschaut?**
spezielles Werkzeug? spezielle Plattform? Desktop? Mobil?
 3. **Wie wird die Einhaltung der Programmierrichtlinie unterstützt oder kontrolliert?**
spezielles Werkzeug/Plugin? reicht Formatter der aktuellen IDE? welche XML-Dateien habe ich schon für Richtlinienimporte in meine IDE?
 4. **Wie wichtig sind Richtlinien für moderne Elemente (Streams, Lambdas)?**

Check: Alles ordentlich und kommentiert?

„Checkstyle“:

- Entwicklungswerkzeug zum Sicherstellen eines „Coding Standards“
- gut konfigurierbar, um vielfältige Standards abbilden zu können
- mitgelieferte Beispielkonfigurationen:
„Sun Code Conventions“ und „Google Java Style“
- gibt es als Plugin für Eclipse:
<https://checkstyle.org/eclipse-cs/#!/>



Vielen Dank für Ihre
Aufmerksamkeit.

