

programmierung und datenbanken

Joern Ploennigs

Exceptions

MIDJOURNEY: EXCEPTION IN TIME, REF. SALVADOR DALÍ

WIEDERHOLUNG: HÖRSAALFRAGE

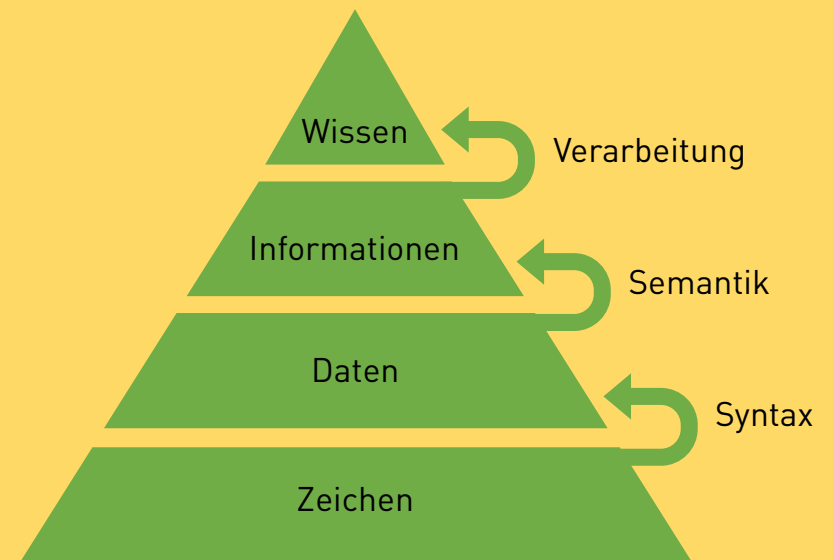
Welche Ebenen hat die Wissenspyramide?



Midjourney: Tower of Babel

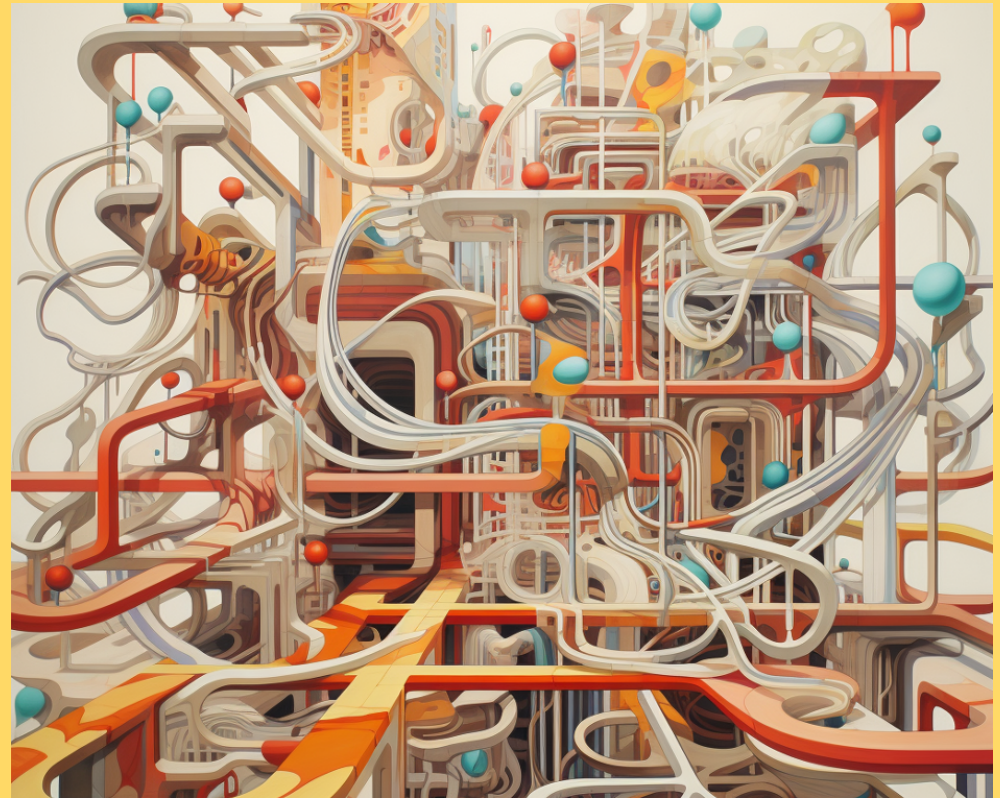
WIEDERHOLUNG: WISSENSPYRAMIDE

Die Wissenspyramide ist ein Modell zur Darstellung der Entstehung von Wissen. Die vier Elementtypen: *Zeichen*, *Daten*, *Informationen* und *Wissen* werden pyramidenförmig als 4 Ebenen dargestellt. Zeichen formen die Basis und das Wissen die Spitze der Pyramide.



WIEDERHOLUNG: HÖRSAALFRAGE

Was sind die vier Grundelemente eines Programms?

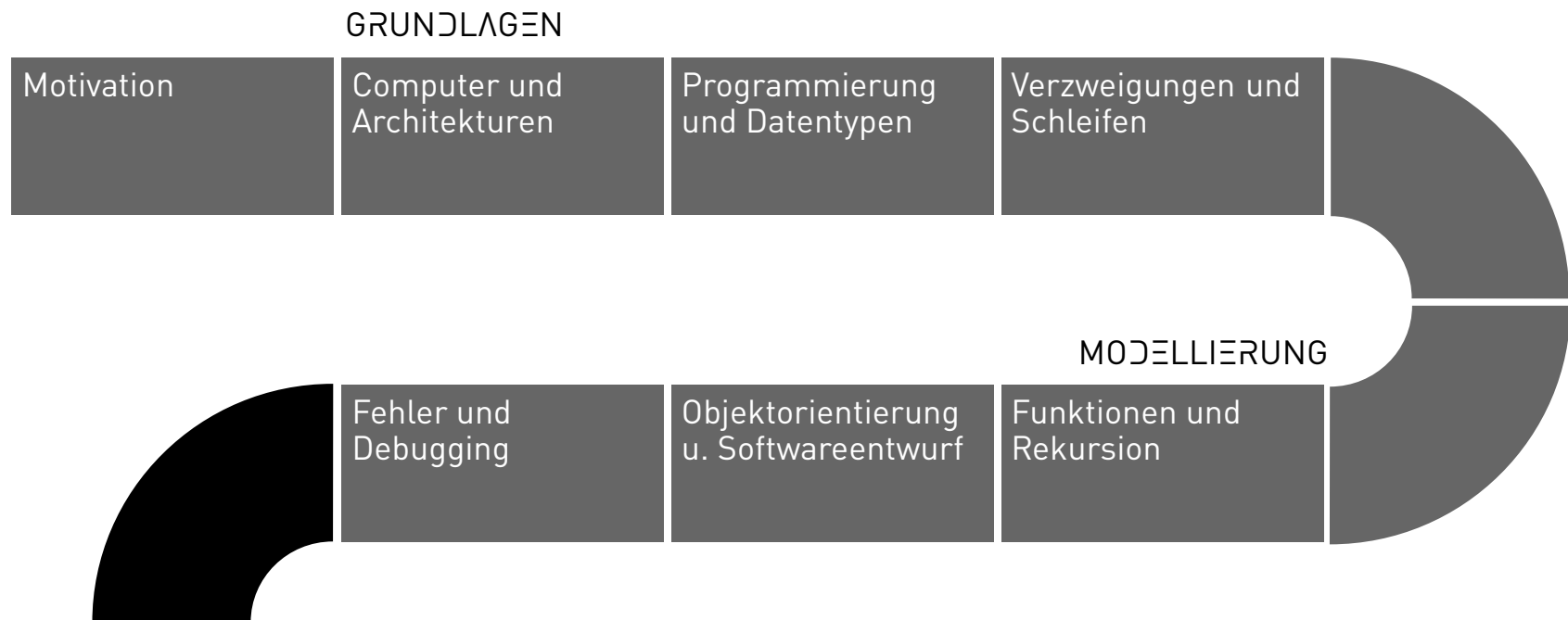


Midjourney: Fundamentals of a Programming Language

WIEDERHOLUNG: PROGRAMMELEMENTE

- Wir kennen alle grundlegenden Grundelemente eines Programms:
 - Statements
 - Funktionen
 - Verzweigungen
 - Schleifen & Rekursion
- Wir wissen wie man Algorithmen plant und diese Elemente kombinieren kann.

ABLAUF



WARUM MACHEN WIR FEHLER?

Gewollte Fehler passieren, wenn:

- wir bewusst Fakten ignorieren und lieber unserem Bauch vertrauen
- wenn wir Tests durchführen, um zu prüfen ob ein Programm diese Fehler behandelt
- durch gezielte Sabotage (Insider Threat in der Datensicherheit)

Ungewollte Fehler passieren u.a. durch:

- falsche Annahmen da z.B. die Anforderungen nicht hinreichend analysiert worden oder man nicht alle Fehlerfälle durchdacht hat
- Unaufmerksamkeit (Flüchtigkeitsfehler) durch Müdigkeit, Zeitdruck oder Desinteresse
- weil wir an kognitive/körperliche Limits stoßen (z.B. optische Täuschungen)

FEHLER SIND MENSCHLICH

- *Computer sind von Menschen gemacht*
- Die mentalen Modelle anderer Menschen haben (oft subtile) Unterschiede zu unseren eigenen, z.B. wegen:
 - Alter
 - Herkunft
 - Ausbildung
 - Erfahrung
 - Kultur
- Die meisten Fehler im Alltag passieren an der *Schnittstelle zwischen uns und den internen Vorgängen*: Nutzeroberflächen, Hardware, etc.

HÖRSAALFRAGE

Was bedeutet dieses Zeichen?



icon-icons.com

WAS BEDEUTET DIESES ZEICHEN?

- Das Diskettensymbol wird häufig zum Abspeichern benutzt.
- Es stammt ursprünglich aus den 90er wo Daten noch auf Disketten gespeichert wurden.
- Heutzutage haben viele diesen Kontext nicht und können das Symbol nicht intuitiv deuten.
- Beispiel für unterschiedliche mentale Modelle zwischen Generationen



icon-icons.com



www.pxfuel.com

WANN MACHEN COMPUTER FEHLER?

- Computer machen nur sehr selten Fehler (z.B. Bitfehler durch kosmische Strahlung, Fehler bei Berechnungen mit Fließkommazahlen, Überläufe von Integer, Fehler bei der Datenübertragung oder Speicherung)
- Meistens ist es allerdings der Nutzer oder Programmierer der die Fehler erst ermöglicht hat (Falsche Datentypen, keine gute Ausnahmen- oder Fehlerbehandlung)
- Die eigentliche Frage ist: Gibt es Fehler im System abseits von unserem eigenen Programmcode?

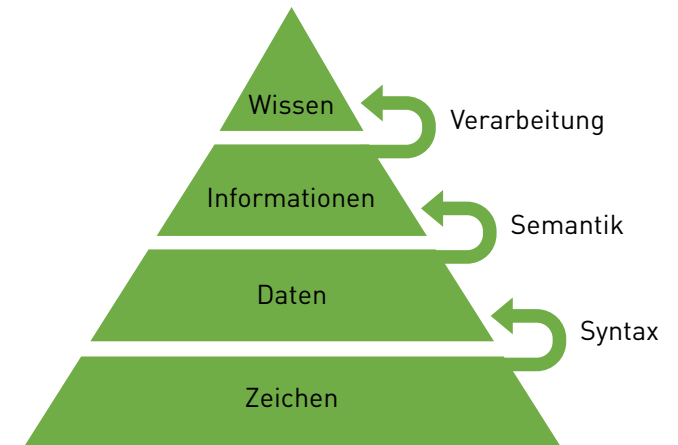
WIE BEUGT MAN FEHLERN VOR?

- *Nutzen einer integrierten Entwicklungsumgebung (IDE)*
 - Automatische Fehlererkennung von Syntaxfehlern
 - Code-Vervollständigung
 - Code-Generierung (z.B. aus Diagrammen)
- *Umfangreiches Planen* der Software mit Anforderungsanalyse, Softwareentwurf, Testentwurf, etc.
- *TESTEN, TESTEN, TESTEN* mittels spezieller Testfälle (Unit-Tests!)
 - Nicht nur reiner Funktionstest (macht es was es soll?)

KLASSEN VON FEHLERN

Entlang der Wissenspyramide:

- *Lexikalisch* – Code enthält ungültige Zeichen oder Zeichenketten
- *Syntaktisch* – Code kombiniert lexikalisch gültige Wörter in ungültiger Weise
- *Semantisch* – Code ist syntaktisch gültig, aber liefert keine Ergebnisse/stürzt ab
- *Logisch* – Code ist semantisch gültig, aber löst das gegebene Problem nicht



ANDERE KLASSIFIZIERUNGEN VON FEHLERN

Nach Häufigkeit und Reproduzierbarkeit:

- *Deterministische Fehler* - treten immer auf und lassen sich reproduzieren
- *Sporadische Fehler* - treten nur selten und in besonderen Zuständen auf. Sind schwer zu reproduzieren

Nach Auftreten:

- *Statische Fehler* - Direkt im geschriebenen Code erkennbar
- *Dynamische Fehler* - Treten erst beim Ausführen des Programms auf

LEXIKALISCHE & SYNTAKTISCHE FEHLER

- Sind in den meisten Fällen *statisch* und *deterministisch*
- Erzeugen *immer* eine Fehlermeldung
 - Schon in IDE, beim Compilieren oder nach dem Ausführen
- *Diese Fehlermeldungen können allerdings trügerisch sein!*

LEXIKALISCHE & SYNTAKTISCHE FEHLER - BEISPIELE

- Ein *Schlüsselwort* ist falsch geschrieben
- *Nicht geschlossene* Anführungszeichen oder Klammern
- Einen *Doppelpunkt* nach einer Bedingung oder Funktion vergessen
- *Falsche Einrückung* nach einer Bedingung, Schleife oder Funktion
- Ggf. auch falsches Einrücken durch *Tabs* vs. *Leerzeichen*

LEXIKALISCHE & SYNTAKTISCHE FEHLER - BEHANDLUNG

- *Erster Blick*: In welcher Zeile ist der Fehler aufgetreten?
- Gibt der *Text der Fehlermeldung* einen Hinweis?
- Liegt der Fehler in der Zeile oder ist der Fehler nur ein *Symptom* und der Fehler liegt in einer Zeile davor?
- Normalerweise *einfach zu beheben* und von der Fehlerzeile den Code Zeile für Zeile rückwärts durchgehen
- Falls nicht: *Refactoring* - Code zur besseren Lesbarkeit umstrukturieren ohne das Verhalten zu verändern

SEMANTISCHE FEHLER - GRUNDLAGEN

- Entstehen durch die *fehlerhafte Anwendung* von syntaktisch korrekten Programmstrukturen auf ein bestimmtes Problem
- Semantische Fehler können *erkennbar* oder *unerkenbar* sein. Sie sind immer deterministisch, aber können statisch oder dynamisch auftreten
- *Erkennbare Fehler* sind Fehler von denen man weiß, dass sie auftreten können (z.B. Division durch Null, keine Internetverbindung, etc.)
 - Sollten *immer* abgefangen und behandelt werden
- *Unerkannte Fehler* sind beim Erstellen des Programms unbekannt. Deutlich schwieriger abzufangen

SEMANTISCHE FEHLER - BEISPIELE

- Eine *Variable* wird als *Funktion* aufgerufen
(*statisch, deterministisch*)
- Ein *Operator* wird mit einem Wert des falschen Typs genutzt
(*statisch oder dynamisch, deterministisch*)
- Ein Wert des falschen Typs wird in eine Funktion übergeben
(*statisch oder dynamisch, deterministisch*)
- Es geschieht eine *Division durch 0*
(*dynamisch, deterministisch*)

SEMANTISCHE FEHLER - BEHANDLUNG

Erster Schritt: Verstehen der Fehlermeldung

- *Idealfall*: Gibt das exakte Problem an
- *Meistens*: Gibt Hinweise auf ein Problem
 - Ort
 - Art des Problems
 - Die betroffene Variable/Operation
- *Oft*: Die Fehlermeldung ist nur ein Symptom des eigentlichen Problems

SEMANTISCHE FEHLER - BEHANDLUNG (2)

Falls die Fehlermeldung nicht genug Informationen gibt:

- *Programmablauf loggen* und so besser nachvollziehen:
 - Weitere Ausgaben hinzufügen (z.B. über `print()` -Funktion)
 - Variablenzustände prüfen oder prüfen ob gewisse Zeilen erreicht werden
- *Debugging-Tools* nutzen:
 - Programmablauf an bestimmten Punkten (*Breakpoints*) pausieren
 - Programmablauf *Schritt für Schritt* nachvollziehen
 - *Variablen beobachten* in diesem Moment

LOGISCHE FEHLER - GRUNDLAGEN

Entstehen durch:

- Fehler in der *Programmlogik*
- Ein *falsches mentales Modell*
- Einen grundlegenden *Fehler im Programmentwurf*
- *Flüchtigkeitsfehler* bei der Übersetzung vom Entwurf zur Implementierung

Charakteristika:

- Erzeugen *nur selten Exceptions* sondern falsches Verhalten/Ergebnisse
- Bleiben oft *unentdeckt* und treten erst dynamisch auf
- Können *deterministisch* oder *sporadisch* sein

LOGISCHE FEHLER - BEISPIELE

Deterministische Fehler:

- Verwendung eines *falschen Faktors* in einer Einheitenkonvertierung
- Verwenden der *falschen Art von Schleife* oder falsche Schachtelung
- Verwenden einer *falschen Zeitzone* beim Schreiben von Sensor-Daten

Sporadische Fehler:

- Probleme mit *Sommer-/Winterzeitkonvertierung* bei Sensor-Daten
- Die Bedingung in einer Schleife hat Fälle in der sie *immer True* ist und so endlos wird

LOGISCHE FEHLER - BEHANDLUNG

Deterministische Fehler:

- *Testfälle erstellen* um zu reproduzieren wann und wie unerwartetes Verhalten stattfindet
- *Debugger nutzen* um die Ausführung nachzuvollziehen und Fehlerursache zu isolieren
- Werte *systematisch verfolgen* und Schritt-für-Schritt auf Korrektheit prüfen

Sporadische Fehler:

- Mittels *Logging* (`print()`) den Zustand des Fehlers identifizieren, so dass er reproduzierbar wird
- *Bedingte Breakpoints* im Debugging nutzen (Programm bleibt nur stehen, wenn eine Fehlerbedingung gegeben ist)

ZUSATZ: WARNUNGEN IN DER IDE ODER VOM COMPILER

- Weisen auf Code hin, der *syntaktisch und semantisch korrekt* scheint, seinen Zweck aber wahrscheinlich *nicht erfüllen* wird
- Basieren z.B. auf *Erfahrungswerten*
- Können dabei helfen *semantische und logische Fehler frühzeitig* zu erkennen

LESSON LEARNED

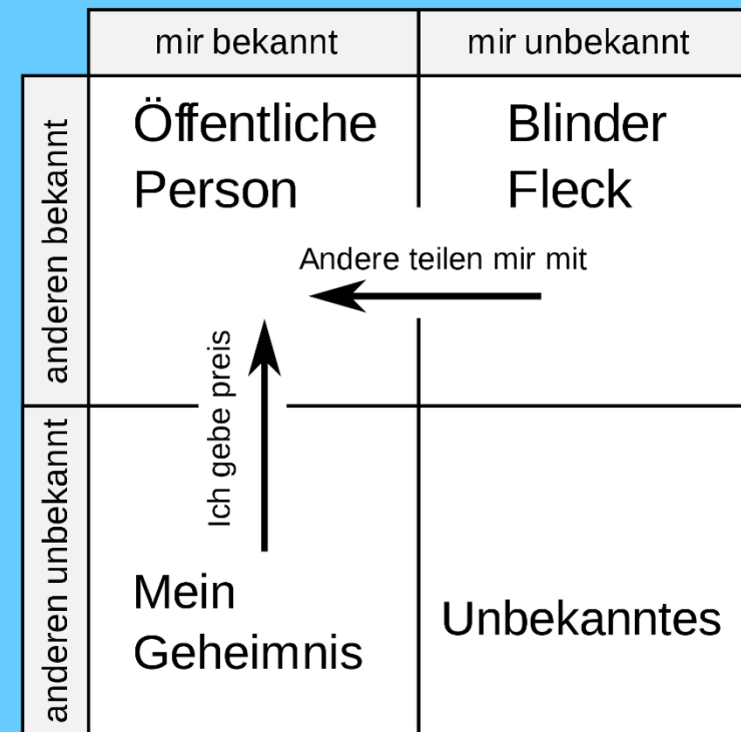
Wofür ist Feedback gut?



Midjourney: Dorian Gray looking at his picture seeing an old monster

LESSON LEARNED - SELBST UND FREMDWARNEHMUNG

- Unser Selbstbild stimmt nie mit dem Fremdbild überein, da wir unsere Erfahrungen und Gedanken mit einbringen
- Das Fremdbild wird nur durch Beobachtung und Schilderungen von unserer Person gebildet
- Feedback ist entscheidend, um das Selbstbild und Fremdbild abzugleichen



fragen?

