technische universität
dortmund

Software- Engineering für langlebige Systeme

Thomas Ruhroth | Dortmund
SoSe 2014

Softwarewartung auf Code-Ebene

VL3

- Grundlagen der Wahrnehmungspsychologie
  - Phänomene
  - Auswirkungen auf die Maintainer von Code
  - Programmieren, so dass negative wahrnehmungspsychologische Phänomene minimiert werden.
- Coding Guidelines

- Ziele:
  - Die Phänomene der Wahrnehmungspsychologie kennenlernen.
  - Erkennen, dass sich niemand den Auswirkungen der Wahrnehmungspsychologie entziehen kann.
  - Wichtigkeit von Coding Guidelines erkennen.
  - Prinzip der Coding Guidelines verstehen.

Psychologie des Programmierens

Psychologie ist eine empirische Wissenschaft. Sie beschreibt und erklärt das Erleben und Verhalten des Menschen, seine Entwicklung im Laufe des Lebens und alle dafür maßgeblichen inneren und äußeren Ursachen und Bedingungen. (Wikipedia)

- Beziehung vom Programmierer zu „seinem Code“
- Denken und Verstehen von fremden Code
- Lernen aus Fehlern
- Fehlschlüsse

Wahrnehmung

http://www.youtube.com/watch?v=lGQmdoK_ZfY

Aufgabe beim Video

Sie werden gleich zwei Mannschaften (weiß und schwarz) sehen. Zählen Sie bitte die Passe der **weißen** Mannschaft untereinander.

Wahrnehmung

http://www.youtube.com/watch?v=IGQmdoK_ZfY

Lösung

Es sind 16 Würfe

Was ist noch passiert?

Was ist noch passiert?

- Ein Gorilla ist durch das Bild gelaufen
- Ein schwarzer Spieler hat das Feld verlassen
- Die Hintergrundfarbe wechselte von rot nach gelb

Zum Beweis

Nochmal der Film

Monkey Business Experiment

- Beispiel für den „Scheinwerferprinzip“
- Um das Gehirn nicht zu überlasten, werden „unwichtige“ Details ausgeblendet.
- Wie bei einem Scheinwerfer werden die „wichtigen“ Details beleuchtet.

Scheinwerferprinzip/ Wahrnehmungsselektion

Was wahrgenommen wird hängt ab von:

- Aufmerksamkeit
- Gewöhnung
- Erwartung

Persönliche Einflüsse

Die Wahrnehmungsselektion wird beeinflusst durch:

- Reizintensität
- Reizkontext
- Reizeindeutigkeit

Reizeigenschaften

Probleme mit dem Scheinwerferprinzip

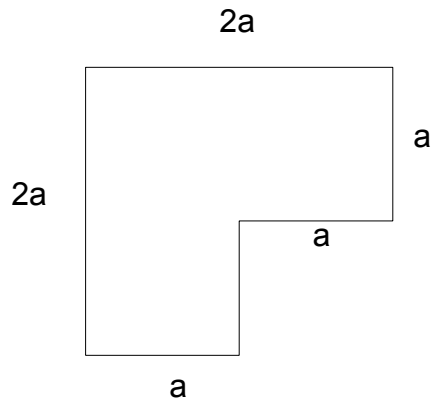
- Einteilung von „wichtigen“ und „unwichtigen“ Informationen ist meist unbewusst
- Häufig sind „unwichtige“ Informationen doch relevant für die Problemlösung
  - Kein vollständiger Überblick

Neues Experiment

- Halten Sie bitte, wenn sie die Lösung gefunden haben, diese geheim!

Aufgabe

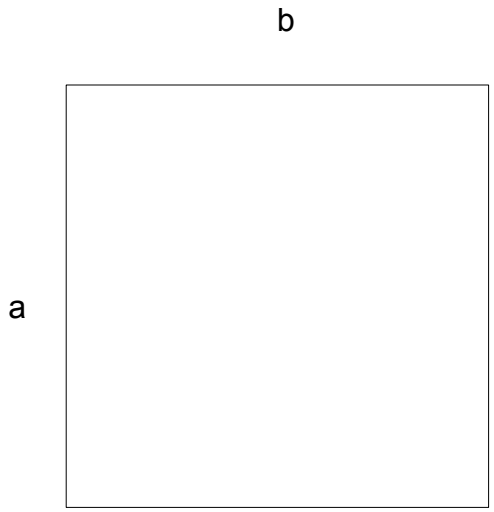
- Teilen Sie bitte die Figur in vier deckungsgleiche Teile:



Nun wird es schwieriger!

- Teilen Sie bitte die Figur in **fünf** deckungsgleiche Teile:

a=b

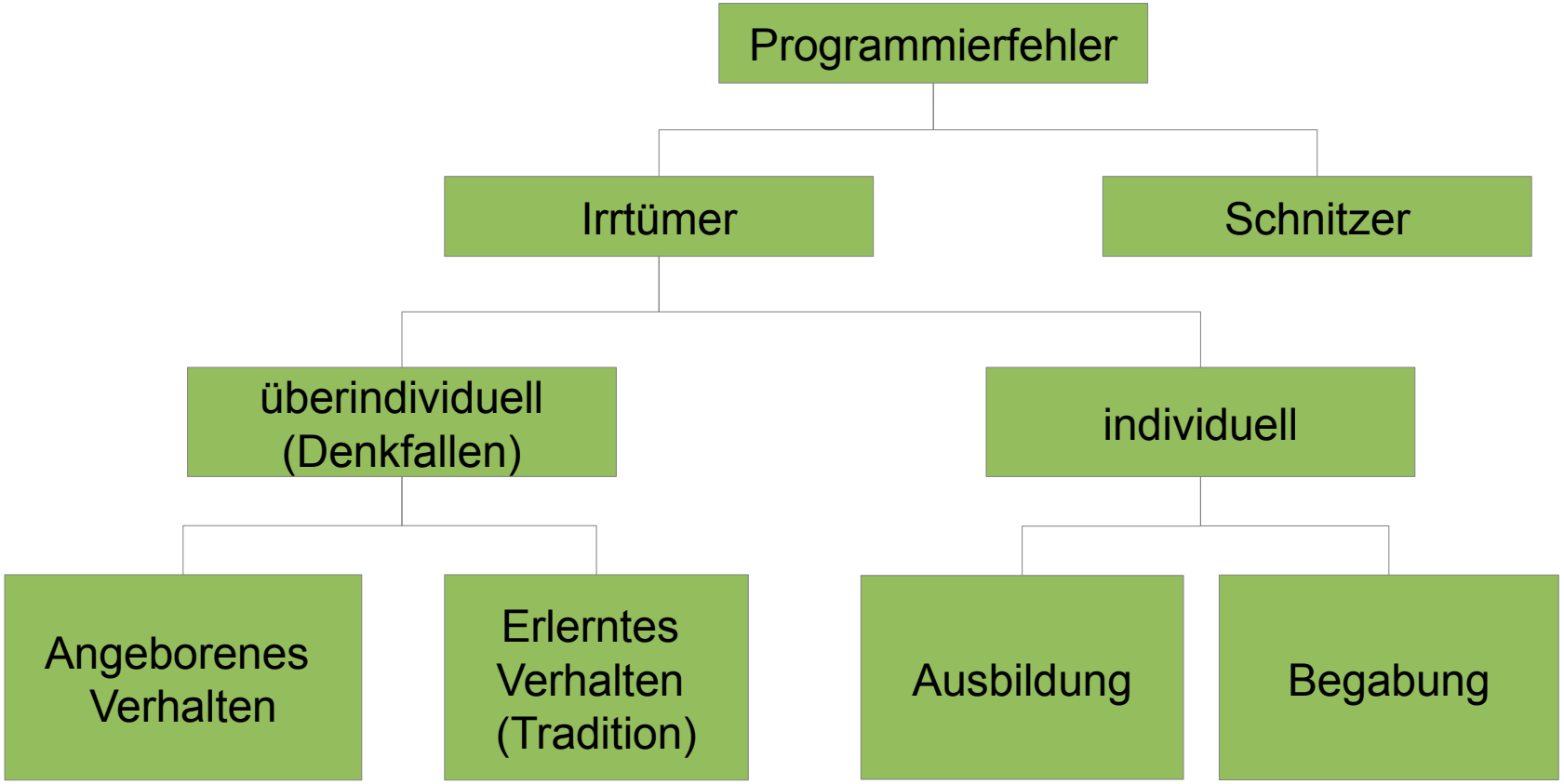


Priming

- Denken wird von den Ereignissen vorher beeinflusst
- Kompliziertes Denken wird weitergeführt

- Fehlschlüsse aus der Erfahrung

Programmierfehler [Gram90]



Hintergrundwissen [Balz]

Jeder Mensch benutzt bei seinen Handlungen bewusst oder unbewusst angeborenes oder erlerntes Hintergrundwissen.

Dieses Hintergrundwissen ist Teil des Wissen, das einer Bevölkerungsgruppe, z.B. den Programmierern, gemeinsam ist.

Programmierfehler lassen sich nun danach klassifizieren, ob das Hintergrundwissen für die Erledigung der Aufgabe angemessen ist oder nicht.

Schnitzer [Balz]

„Schnitzer`` sind Tippfehler, Versprecher usw.

Ursachen:

- Fehlende Konzentration
- Schlechte Oberfläche
-

Irrtümer [Balz]

Fehlern, die auf Irrtümer beruhen, ist gemeinsam, dass der Programmierer den Fehler auch beim wiederholten Durchlesen seines Programms nicht sieht.

Irrtümer sind der Kategorie ,,Wissen`` zuzuordnen.

Denkfallen (Überindividuelle Irrtümer) [Balz]

- Hintergrundwissen der Aufgabenstellung nicht angemessen
- Prinzipien und Strukturen zusammen:
  - Übergeordnete Prinzipien
    - Scheinwerferprinzip
    - Sparsamkeits- bzw. Ökonomieprinzip
  - Die ,,angeborenen Lehrmeister``
    - Strukturerwartung (Prägnanzprinzip, kategorisches Denken)
    - Kausaltitätserwartung (lineares Ursache-Wirkungs-Denken)
    - Die Anlage zur Induktion (Überschätzung bestätigender Informationen)
  - Bedingungen des Denkens
    - Assoziationen
    - Einstellungen

Denkfallen (Überindividuelle Irrtümer) [Balz]

Denkfallen treten auf, wenn das Hintergrundwissen der Aufgabenstellung nicht angemessen ist. Denkfallen ergeben sich aus bestimmten Denkstrukturen und -prinzipien. Diese führen zu einem Verhaltens- und Denkmodell, dass das Verhalten eines Programmierers beeinflusst. Das Verhaltens- und Denkmodell setzt sich aus folgenden Prinzipien und Strukturen zusammen:

- Übergeordnete Prinzipien
  - Scheinwerferprinzip
  - Sparsamkeits- bzw. Ökonomieprinzip
- Die ,,angeborenen Lehrmeister``
  - Strukturerwartung (Prägnanzprinzip, kategorisches Denken)
  - Kausaltitätserwartung (lineares Ursache-Wirkungs-Denken)
  - Die Anlage zur Induktion (Überschätzung bestätigender Informationen)
- Bedingungen des Denkens
  - Assoziationen
  - Einstellungen

Scheinwerferprinzip [Balz]

Das ,,Scheinwerferpinzip`` besagt, dass aus den Unmengen an Informationen, die der Mensch ständig aus seiner Umwelt erhält, stets nur relativ kleine Portionen ausgewählt und bewusst verarbeitet werden. Leider sind es oft wichtige Dinge, die unbeachtet bleiben. Viele Programmierfehler zeigen dies: Ausnahme- und Grenzfälle werden übersehen; die Initialisierung von Variablen wird vergessen; Funktionen besitzen unübersehbare Nebenwirkungen.

Sparsamkeitsprinzip [Balz]

Das Sparsamkeits- oder Ökonomieprinzip besagt, dass es darauf ankommt, ein Ziel mit möglichst geringem Aufwand zu erreichen. Der Trend zum sparsamen Einsatz der verfügbaren Mittel birgt allerdings auch Gefahren in sich. Denk- und Verhaltensmechanismen, die unter normalen Umständen vorteilhaft sind, können dem Programmieren zum Verhängnis werden.

Typische Fehler gehen auf falsche Hypothesen über die Arbeitsweise des Computers zurück. Insbesondere mit der Maschinenarithmetik kommt der Programmierer oft aufgrund zu einfacher Modellvorstellungen in Schwierigkeiten.

Angeborene Lehrmeister [Balz]

Die ,,angeborenen Lehrmeister`` (K.Lorenz) stellen höheres Wissen dar, das den weiteren Wissenserwerb steuert. Dieses höhere Wissen spiegelt den ,,Normalfall`` wider. Mit ungewöhnlichen Situationen wird es nicht so gut fertig.

Strukturerwartung [Balz]

Die Strukturerwartung erleichtert dem Menschen die Abstraktion. Daraus folgt das Prägnanzprinzip, das besagt, dass es sich lohnt, Ordnung aufzuspüren und auszunutzen.

Eine wirkungsvolle Methode, Ordnung in die zunächst unübersichtlich erscheinende Welt zu bringen, ist das Bilden von Kategorien.

Prägnanzprinzip und kategorisches Denken führen in außergewöhnlchen Situationen gelegentlich zu Irrtümern, zu unpassenden Vorurteilen oder zu Fehlverhalten. Eine Reihe von Programmierfehlern lässt sich darauf zurückführen. Beispielsweise gilt das assoziative Gesetz für reelle Zahlen, aber nicht für die Maschinenarithmetik.

Allgemein gilt: Fehler, die darauf beruhen, dass der Ordnungsgehalt der Dinge überschätzt wird oder den Dingen zu viele Gesetze auferlegt werden, lassen sich auf das Prägnanzprinzip zurückführen.

Kausalitätserwartung [Balz]

Die Kausalitätserwartung führt zu einem linearen Ursache-Wirkungs-Denken und zu der übermäßigen Vereinfachung komplexer Sachverhalte: Der Mensch neigt dazu, Erscheinungen vorzugsweise nur einer einzigen Ursache zuzuschreiben. Er macht oft die Erfahrung, dass die gleichen Erscheinungen dieselbe Ursache haben, so dass dieses Vorurteil zum festen Bestandteil des menschlichen Denkens gehört und nur mit Anstrengung überwunden werden kann.

Das lineare Ursache-Wirkungs-Denken kann in komplexen Entscheidungssituationen „fürchterlich“ [Dör89] versagen. Gerade die Umwelt eines Programmierers ist aber ein vernetztes System.

Die Anlage zur Induktion [Balz]

Der Mensch neigt zu induktiven Hypothesen. Die Fähigkeit, im Besonderen das Allgemeine, im Vergangenen das Zukünftige erkennen zu können, verleitet zur Überschätzung bestätigender Informationen. Beispielsweise wird ein Test, der das erwartete Ergebnis liefert, in seiner Aussagekraft überschätzt. Im Gegensatz zur Deduktion ist die Induktion kein zwingendes Schließen. Gesetzmäßgikeiten werden vorschnell als gesichert angesehen. Dadurch werden Dinge klargemacht, die eigentlich verwickelt und undurchsichtig sind.

In der Programmierung stößt man oft auf die Denkfalle, dass bestätigende Informationen überschätzt werden. Der Programmierer gibt sich oft schon mit einer schwachen Lösung zufrieden. Nach einer besseren wird nicht gesucht. Das Programm wird für optimal gehalten, nur weil es offensichtlich funktioniert.

Bedingungen des Denkens, Assoziationen [Balz]

Die Bedingungen des Denkens betreffen die parallele Darstellung der Denkinhalte und den sequentiellen Ablauf der bewussten Denkvorgänge, d.h., die raum-zeitliche Organisation des Denkens. Neue Denkinhalte werden in ein Netz von miteinander assozierten Informationen eingebettet. Bei der Aktivierung eines solchen Denkinhalts wird das assoziative Umfeld mit aktiviert.

Geht es beim Programmieren um die Zuordnung von Bezeichnern und Bedeutungen, dann spielen Assoziationen eine Rolle3.1. Mnemotechnische Namen können irreführend sein. Es wird eher an den Namen als an die Bedeutung geglaubt. Eine sorglose Bezeichnerwahl kann zur Verwirrung führen.

Die bisherigen Fehler bezogen sich auf das Wissen. Im Wesentlichen ging es um die Gewinnung von Informationen (Filterung und Abstraktion) und um Mechanismen, wie Informationen intern repräsentiert und abgerufen werden (Assoziationen).

Problemlösen-Fallen [Balz]

Es gibt jedoch auch Denkfallen auf dem Gebiet des produktiven Denkens, d.h. des Problemlösens. Fehler im Problemlösungsprozess führen dazu, dass

- entweder gar keine Lösung gefunden wird, obwohl sie existiert, oder dass
- nur eine mangelhafte Lösung erkannt wird, die noch weit vom Optimum entfernt ist.

Einstellungen [Balz]

Einstellungen führen zu einer vorgeprägten Ausrichtung des Denkens. Sie können zurückgehen auf

- die Erfahrung aus früheren Problemlöseversuchen in ähnlichen Situationen,
- die Gewöhnung und Mechanisierung durch wiederholte Anwendung eins Denkschemas,
- die Gebundenheit von Methoden und Werkzeugen an bestimmte Verwendungszwecke,
- die Vermutung von Vorschriften, wo es keine gibt (Verbotsirrtum).

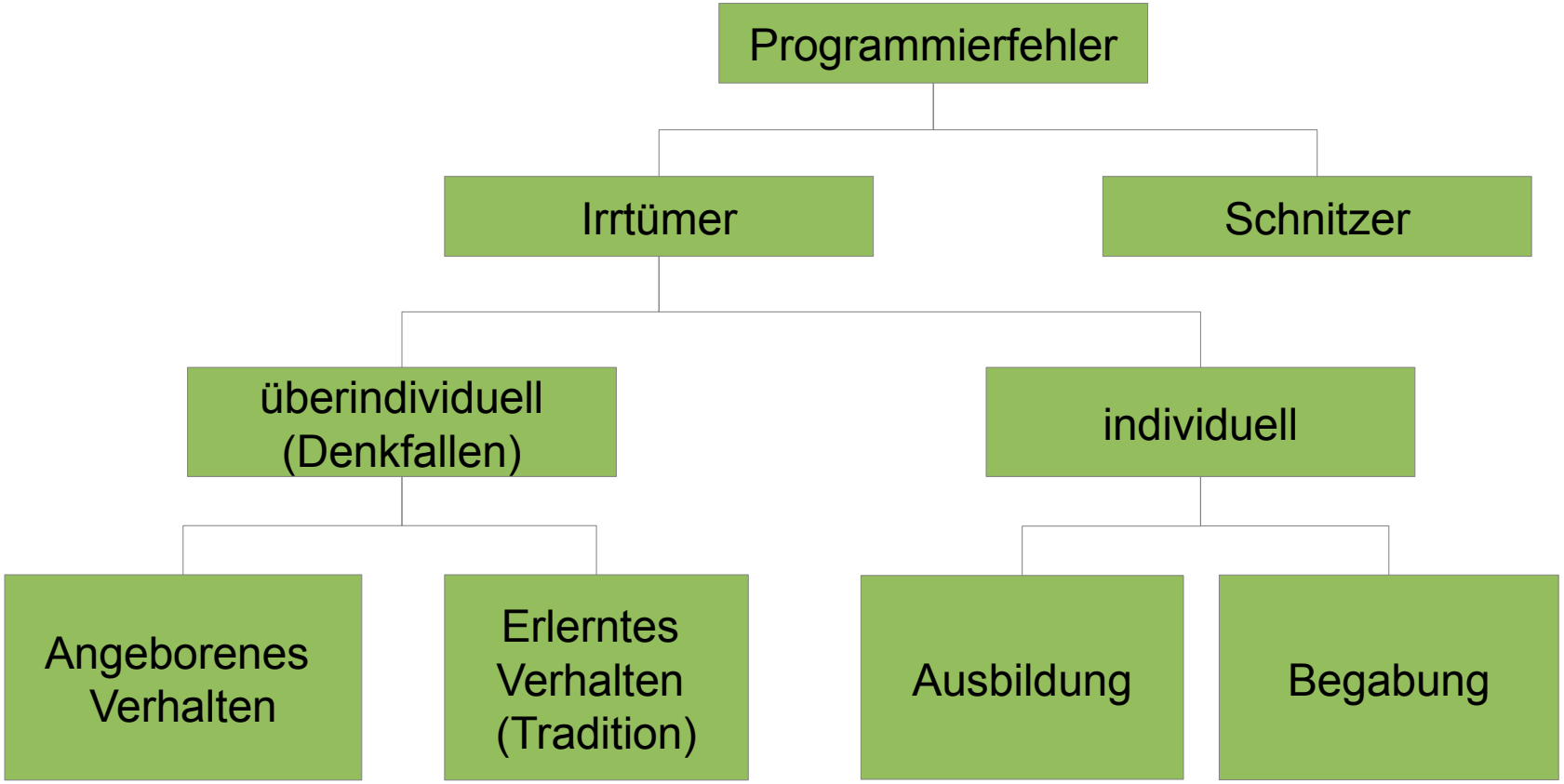
Um Fehler durch Einstellungen zu verhindern, sollte das Aufzeigen von Lösungsalternativen und eine Begründung der Methodenwahl zum festen Bestandteil der Problembearbeitung gemacht werden.

Einstellungen führen zu determinierenden Tendenzen beim Problemlösen:

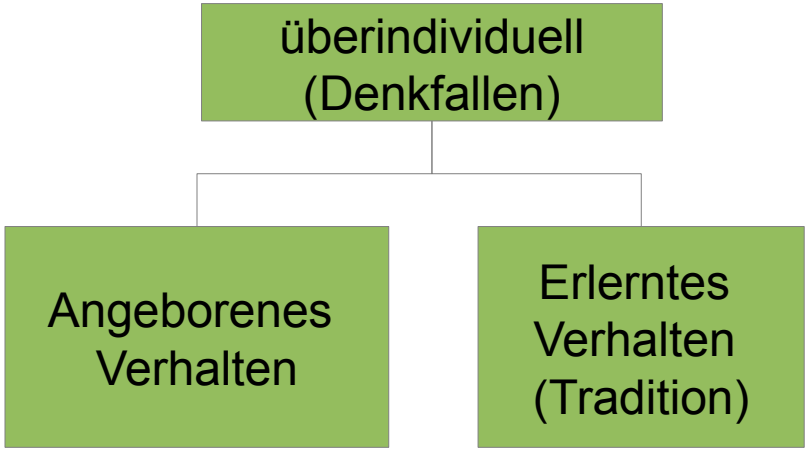
Die Anziehungskraft, die von einem nahen (Teil-)Ziel ausgeht, verhindert das Auffinden eines problemlösenden Umwegs.

Die vorhandenen Strukturen und Teillösungen lenken den weiteren Lösungsprozess in bestimmte Bahnen.

Programmierfehler [Gram90]



Denkfallen und Erosion I



- Übergeordnete Prinzipien
  - Scheinwerferprinzip
  - Sparsamkeits- bzw. Ökonomieprinzip
- Die ,,angeborenen Lehrmeister``
  - Strukturerwartung (Prägnanzprinzip, kategorisches Denken)
  - Kausaltitätserwartung (lineares Ursache-Wirkungs-Denken)
  - Die Anlage zur Induktion (Überschätzung bestätigender Informationen)
- Bedingungen des Denkens
  - Assoziationen
  - Einstellungen

Denkfallen und Erosion II

Ziel:

- Software sollte eine Struktur aufweisen, die Denkfallen vermeidet

Problem:

- Viele Software-Entwickler/Designer etc. sehen ihre Arbeit als „Kunst“
- Individualismus herrscht vor – Keiner möchte zugunsten der Gruppe seine Arbeitsweise anpassen

Denkfallen verhindern I

Coding Conventions

Coding Conventions

- Menge von Richtlinien die den gewünschten Programmierstil beschreiben.
  - Dateiorganisation
  - Einrückungen
  - Kommentare
  - White Spaces
  - Namenskonventionen
  - Best Practices
  - ...

Meta-Bemerkungen

- Es ist wichtiger Richtlinien zu haben und zu befolgen, als dass die Richtlinien perfekt sind.
- Es gibt verschiedene Meinungen was „perfekte“ Coding Guidelines sind.

- Mein Ziel: Vorstellung der
  - Vorteilen
  - Auswirkungen
  - Hilfsmittel zu Coding Guidelines .
- Nicht mein Ziel:
  - Einen verbindlichen Satz von Richtlinien zu erstellen

Coding Guidelines und Wahrnehmungspsychologie

- Erfüllung von Erwartungen (auf Programmstil-Ebene)
  - Strukturerwartung
    - Finden von Deklarationen
    - Änderungsvereinfachung
    - ....
  - Kausalitätserwartung
- Vermindern der Probleme des Scheinwerferprinzips
  - Verdrängen von „unwichtigen“ Coding Details
  - Fokus auf den Code-Inhalt statt auf die Syntaxstruktur

Beispiel: Das Komma bei Arrays

Komma nach dem letzten Element in einem Array-Literal

- Keine Komma-Korrektur beim Umstellen und Kürzen
- Kein Mehraufwand beim Erweitern (Anzahl neuer Zeichen gleich)
- Nicht in allen Sprachen möglich!

```
const char *array[] = {
    "item1",
    "item2",
    "item3",
};
```

Beispiel: Keine Nutzung von impliziten Blöcken

If mit implizitem Block

```
If (x > 0)
  return x;
```

- Bei Erweiterung muss der Block eingefügt werden
- Erstellen des Blockes wird wichtig, dafür fallen andere Details aus dem „Scheinwerfer“

If ohne implizitem Block

```
If (x > 0) {
  return x;
}
```

- Block existiert

Einhaltung der Richtlinien

- Automatisierte Code-Anpassung
- Automatisierte Prüfung der Einhaltung
- Code-Reviews
- Gruppen-Kultur

- Die meisten Richtlinien passen nicht in jede Kategorie!

Automatisierung

Einige Coding Guidelines Richtlinien lassen sich automatisieren:

- Einrückungen
- Auslösung von impliziten Blöcken
- Vorlagen für Kommentare
- Vorziehen von Deklarationen
- Sortierung von Klassenbestandteilen

Nutzen Sie immer diese Automatisierungen

- Einige IDEs bieten die Möglichkeit den Code bei jedem Speichern anzupassen!
- Tools zur Anpassung/Prüfung bei „commit“ der Änderungen einbinden

Automatisierte Prüfung der Einhaltung

- Prüfung als „Pre-Commit“-Hook in Versionierungssystemen
  - Vorteil: Code im System entspricht den geprüften Regeln
  - Nachteil: Verleitet zu weniger Commits, Stört evtl. den Entwicklungsverlauf
- Prüfung im Rahmen der automatisierten Tests
  - Vorteil: Keine Änderung der gewohnten Arbeitsweise
    - Nachteil: Einhaltungsfehler werden häufig nicht gefixed, da andere Fehler wichtiger sind

Prüfung der Einhaltung bei Code-Reviews

- Baut stark auf die freiwillige Mitarbeit der Gruppe auf
- Es werden nur Teile des Codes geprüft
- Bietet sich für nicht automatisierbare Richtlinien an
  - Architekturvorgaben
  - Dokumentierung
  - Semantische Namenskonventionen

Gruppen-Kultur

- Alle Ziehen am gleichen Strang
- Persönliches (freundliches) Hinweisen bei Abweichungen
- Nicht Steuerbar
- Wenn Gruppen-Kultur funktioniert, gute Einhaltung auf allen Ebenen
- Instabil:
  - Neue Gruppenmitglieder können die Kultur zerstören

Ändern von Coding Guidelines

Nie

Ändern von Coding Guidlines

Am besten nie!

- Änderungen müssen an allen Stellen im Code nachgezogen werden
  - Ansonsten Verlust der Vorteile bei der Strukturerwartung
  - Am ehesten bei automatisierbaren Richtlinien möglich
- Die meisten Änderungen an den Richtlinien bergen mehr Gefahren als sie Vorteile bringen

- Führen Sie nach Möglichkeit nur Regeln ein, die erprobt sind.

To be continued....