Mythen in der Programmierung

Umgang mit Mythen

- Wahrheitsgehalt von Mythen oft schwer feststellbar
- Mythos $\rightarrow$ Ideologie, religiöse Glaubenswahrheit
- Ideologisierung (Markenbildung) von Industrie gefördert
- Wirkung ähnelt der von Team-Regeln $\rightarrow$ bildet Vertrauen
- „gute Mythen“ förderlich, „schlechte Mythen“ gefährlich
- auch viele Leute können einem falschen „Guru“ folgen
- alle Mythen selbst hinterfragen

Programmierparadigmen und Mythen

Was stimmt an folgenden Aussagen?

- imperative Programmierung effizient
- auf Objekte kommt es an
- objektorientierte Programmierung längst out
- funktionale Programmierung nur für Freaks
- Typüberprüfungen schützen vor Fehlern
- Zukunft ist dynamisch
- Technologien sind entscheidend

Daten und Informationen

Informationsgehalt von Aussagen

Frage: „Was kommt zum 3. Test?“

Antwort A: „Alles was in der Vorlesung gemacht wurde“
(richtig, niedriger Informationsgehalt)

Antwort B: „Generizität und Iteratoren sehr wahrscheinlich“
(richtig, höherer Informationsgehalt)

Antwort C: „Der heutige Wetterbericht sicher nicht“
(richtig, inhaltlich irrelevant, etwas Metainformation)

Wovon hängt Informationsgehalt ab?

- Fragestellung (was will ich wissen, auch auf Metaebene)
- Erwartungshaltung (wie wahrscheinlich ist die Antwort)
  - wahrscheinlich $\Rightarrow$ niedriger Informationsgehalt (A)
  - unwahrscheinlich $\Rightarrow$ hoher Informationsgehalt (B)
- Relevanz hinsichtlich Fragestellung
  - irrelevante Teile der Antwort ausgeblendet (C)
- nicht von Korrektheit der Antwort

Daten

- haben gewisse Größe (n Bits)
- jeder Wert stellt einen von 2^n Möglichkeiten dar
- Bedeutung nicht in den Daten festgelegt
- Verwendung/Darstellungsform/Wissen $\Rightarrow$ Bedeutung
- Typ schränkt Darstellungsform (und Bedeutung) ein

Informationsgehalt von Daten

- Fragestellung von Verwendung der Daten bestimmt
- Erwartungshaltung: Auftrittswahrscheinlichkeit der Werte
- irrelevante Werte treten nicht auf (Annahme)
- Daten korrekt wenn berechnet oder validiert (Annahme)

Informationsbegriff nach Shannon

- Auftrittswahrscheinlichkeit p_w eines Wertes w
- Informationsgehalt $I(p_w) = -\log_2(p_w) = \log_2(1/p_w)$
- Einheit des Informationsgehalts: *bit*
- Variable enthält Wert $w \in W$ mit Wahrscheinlichkeit p_w
- *Entropie* der Variablen $H = \sum_{w \in W} p_w \cdot I(p_w)$

  (wenn Auftrittswahrscheinlichkeiten unabhängig)
- Entropie ist Maß für mittleren Informationsgehalt
  und entspricht (aufgerundet) der Mindestanzahl an Bits

Beispiel für maximale Entropie

- `int x;` wobei alle Zahlen gleich wahrscheinlich
- für jede 32-Bit-Zahl z in `x`: $p_z = 1/2^{32}$
- Informationsgehalt $I(p_z) = \log_2(2^{32}) = 32$ bit
- Entropie $H = \sum_{z \in \text{int}} 1/2^{32} \cdot 32 = 2^{32} \cdot 1/2^{32} \cdot 32 = 32$ bit
- mit 32 Bit optimal codiert (maximal mögliche Entropie)
- kompaktere Darstellung (Komprimierung) nicht möglich

Beispiel für niedrige Entropie

- `boolean x;` wobei $p_{\text{true}} = 0,99$ und $p_{\text{false}} = 0,01$
- Informationsgehalt $I(p_{\text{true}}) = \log_2(1/0,99) = 0,0145$ bit
- Informationsgehalt $I(p_{\text{false}}) = \log_2(1/0,01) = 6,644$ bit
- Entropie $H = 0,99 \cdot 0,0145 + 0,01 \cdot 6,644 = 0,081$ bit
- `x` benötigt trotzdem ein ganzes Bit (aufrunden)
- 100 solcher Werte in nur $100 \cdot 0,081 \approx 9$ Bit darstellbar

Redundanz

- Notwendig: Anzahl der Bits $\geq H$ (H ist Entropie)
- Redundanz = Anzahl der Bits $- H$
- bei Redundanz selbe Information vielleicht mehrfach
- Redundanz zur Fehlererkennung und -korrektur nutzbar
- Redundanz in Programmen kann Lesbarkeit verbessern
- Redundanz kann zu Inkonsistenzen führen
  (Information mehrfach, aber widersprüchlich)
- Komprimierung = Vermeidung von Redundanz