

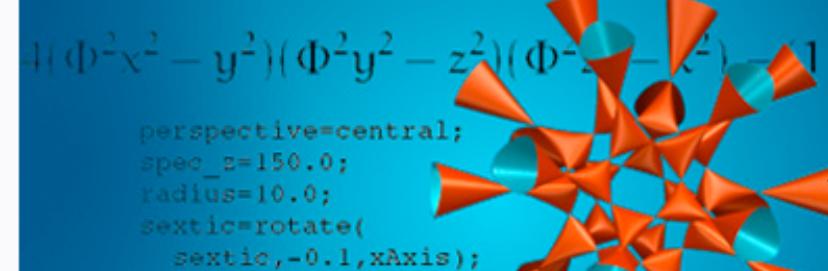
Deklarative Programmierung

Sommersemester 2018

Prof. Christoph Bockisch

(Programmiersprachen und –werkzeuge)

Steffen Dick, Alexander Bille, Johannes Frankenau,
Patrick Frömel, Niclas Schmidt, Jonas Stettin,
Robert Tran, Julian Velten



[Skript 3]

Was Bedeutet Programmentwurf?

- Programmieren = Aufschreiben von Berechnungsvorschriften
- Bedeutung von Berechnungsvorschriften entweder
 - Primitiv (eingebaute Funktion) oder
 - Benutzerdefiniert (im Programm definierte Funktion)
- Benutzerdefinierte Definitionen vermeiden Redundanz
- Warum sollen wir noch eigene Definitionen verwenden?

Kochrezept als Programmierung

```
(cook  
  (heat (fetch-from-cupboard oil))  
  (fold-in  
    (whisk (break (fetch-from-fridge eggs))))  
  (mix  
    (warm (fetch-from-fridge butter))  
    (warm (fetch-from-tap water))  
    (warm (fetch-from-cupboard milk)))  
  (sift salt)  
  (fetch-from-cupboard flour))))
```

Was wird
gemacht?

Details verbergen

- Durch Funktionsdefinitionen können wir Details verbergen
- Ersetze “wie“ (Implementierung) durch “was“ (Funktionsnamen)
- Programm wird kürzer, besser verständlich

Beispiel Serien-Brief

```
(define (letter fst lst signature-name)
```

```
  (string-append
```

```
    (opening lst)
```

Anrede

```
    "\n"
```

Nachricht

```
    (body fst lst)
```

```
    "\n"
```

Grußformel

```
    (closing signature-name)))
```

```
> (letter "Tillmann" "Rendel" "Klaus Ostermann")
```

Beispiel Serien-Brief

```
(define (letter fst lst signature-name)
  (string-append
    (opening lst)
    "Dear "
    (body fst)
    "After the last annual calculations of your GNB"
    "account activity we have determined that you,"
    fst lst
    ", are eligible to receive a tax refund of $479.30.\n"
    "Please submit the tax refund request (http://www...)"
    "and allow us 2-6 days in order to process it."))

(define (closing signature-name)
  (string-append
    "With best regards,\n"
    signature-name))
```

The code defines a function `letter` that takes three arguments: `fst`, `lst`, and `signature-name`. It uses `string-append` to concatenate several strings. The first part of the string is defined by a call to `opening`, which returns a string starting with "Dear ". The second part is defined by `body fst`, which is expanded to show a multi-line string about tax refunds. The third part is defined by `closing signature-name`, which returns a string ending with "With best regards,\n".

Aufteilen des Programms

- Eine Funktion pro Aufgabe
- Anforderungen ändern sich pro Aufgabe
- Daher werden so Auswirkungen von Änderungen minimiert
- Ziel: Größe der Programm-Änderung proportional zur Größe der Anforderungsänderung

Beispiel Serien-Brief

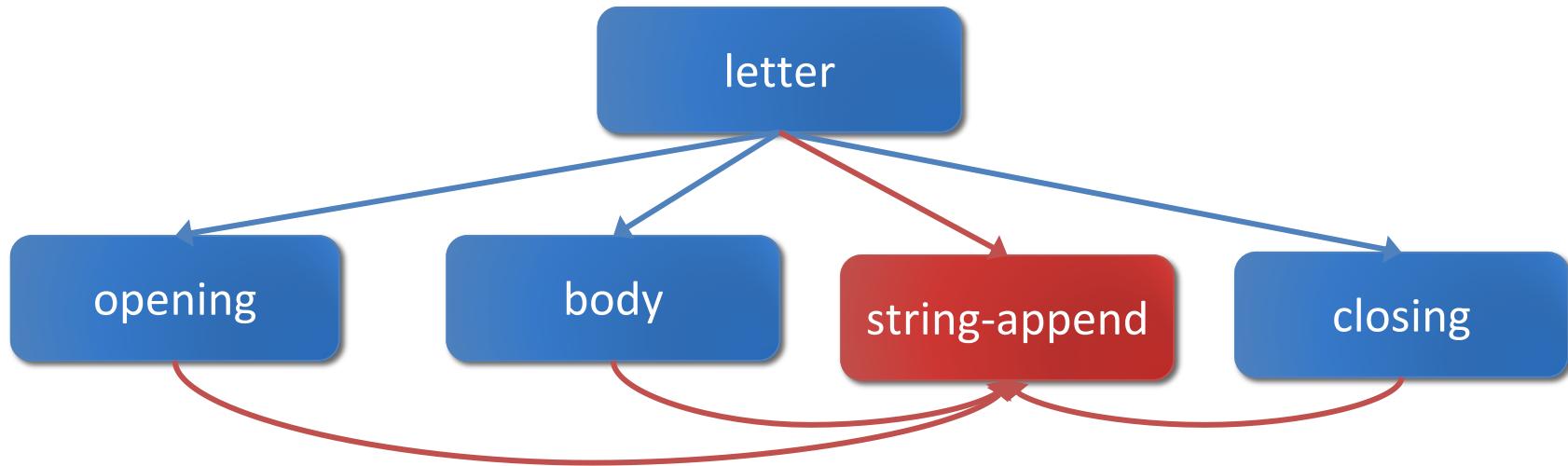
```
(define (letter fst lst signature-name)
  (string-append
    (opening lst)
    "Dear Mr./Mrs. " lst ","))
  (body fst)
  "After the last annual calculations of your GNB"
  "account activity we have determined that you, "
  fst lst
  ", are eligible to receive a tax refund of $479.30.\n"
  "Please submit the tax refund request (http://www...)"
  "and allow us 2-6 days in order to process it."))

  (closing signature-name)
  (string-append
    "With best regards,\n"
    signature-name))
```

Aufteilen des Programms

- Hierarchisch
 - Oberste Ebene: Gesamtaufgabe durch Zusammensetzen von Teilaufgaben beschreiben
 - Mittlere Ebenen: Teilaufgabe durch Zusammensetzen von Teilaufgaben beschreiben
 - Tiefste Ebene: Teilaufgabe durch Zusammensetzen von primitiven Funktionen beschreiben

Hierarchische Aufteilung

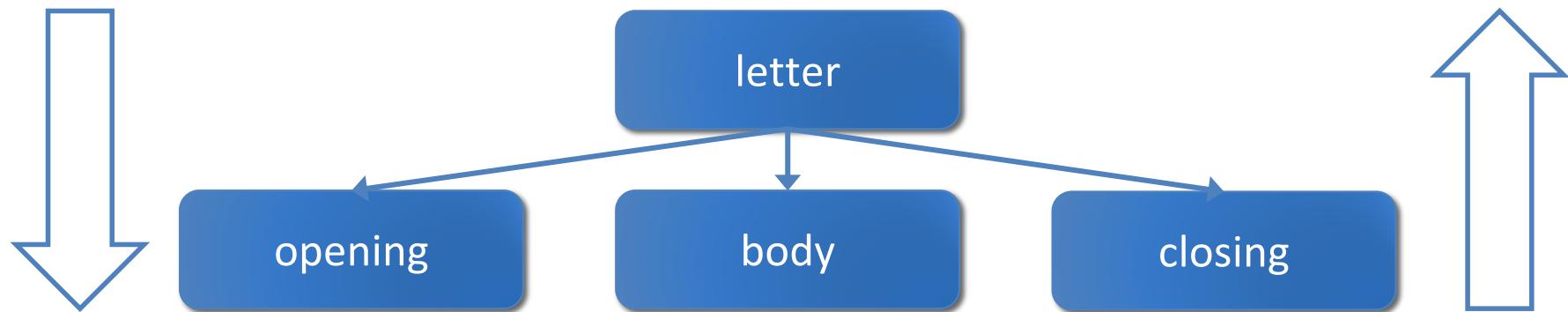


- Azyklischer Graph
 - Benutzung nur von Funktionen tiefer in der Hierarchie
- Ausgezeichneter Startknoten (Wurzel)
 - Hauptfunktion
- Blätter
 - Nutzung von primitiven Funktionen

Spezialfall: Baum.
Teilfunktion wird nur an einer höheren Stelle benutzt.

Entwurfs-Ansätze

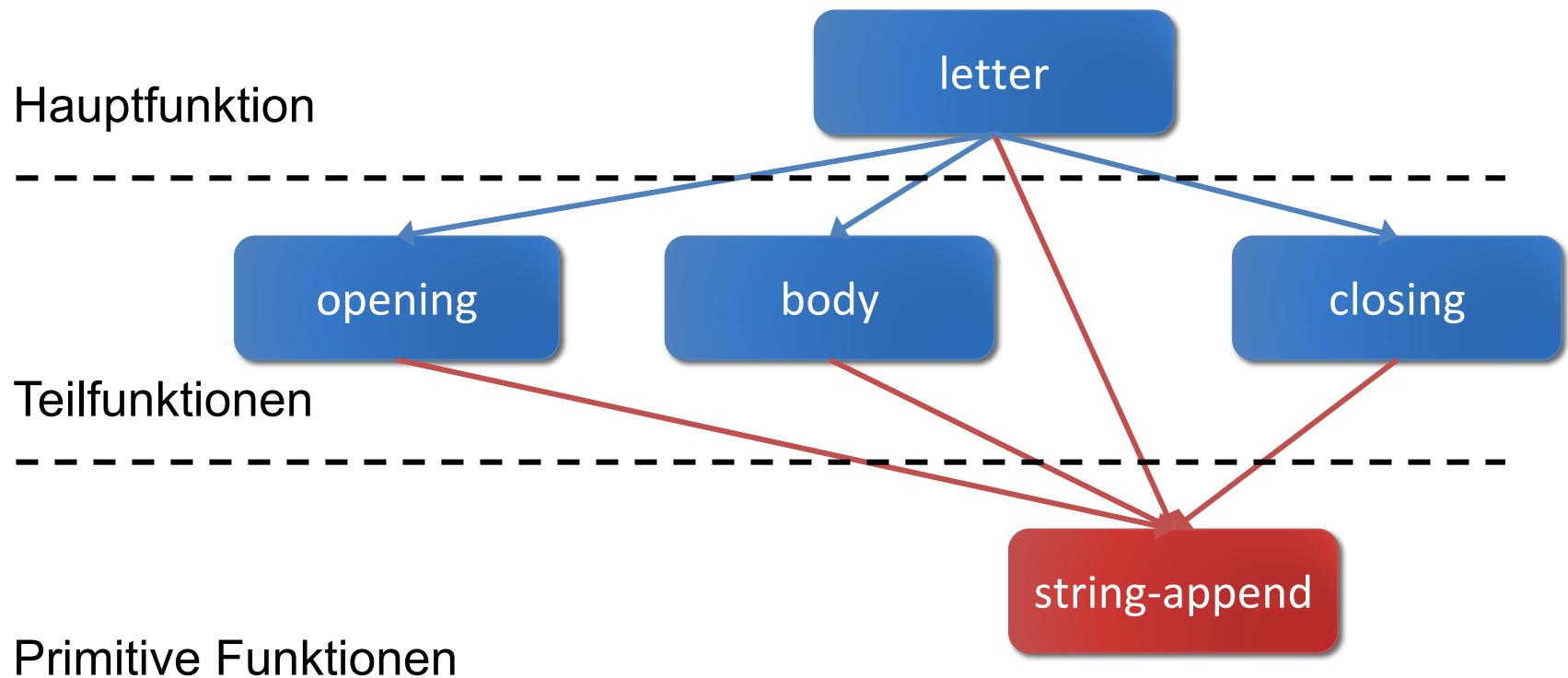
Top-Down



Bottom-Up

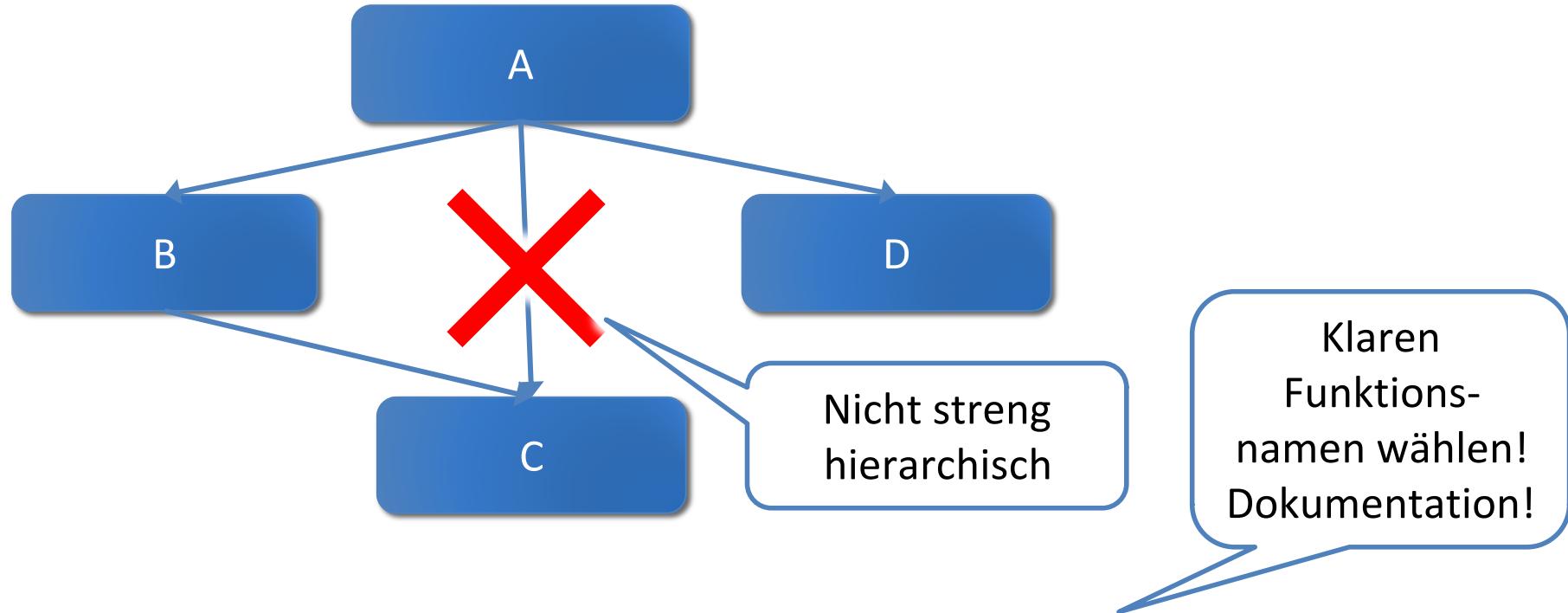
- Wahl des Ansatzes situationsabhängig, bspw.:
 - Ist das “Wie?” wichtig?
 - Sind Ein-/Ausgabe für die Blatt-Funktionen bekannt?

Entwurf in Schichten



Hierarchische Abstraktion

- Ausschließlich Aufruf von Funktionen in direkt darunter liegender Schicht



- Keine Kenntnis des gesamten Programms für Verständnis nötig
- Nur "abstrakte" Kenntnis der verwendeten Funktionen nötig

Dokumentation von Funktionen

- Benennung als Dokumentation
 - Beschreibend
 - Ausdrucksmächtig
 - Eindeutig
- Kommentar vor Funktion
 - Kurze Beschreibung
 - Erwartete Eingabe (Einschränkungen?)
 - Produzierte Ausgabe

Software befindet sich die längste Zeit in der Wartungs-Phase. Dann ist Programmverständnis sehr wichtig!

Vom Problem zum Programm

The owner of a movie theater has complete freedom in setting ticket prices. The more he charges, the fewer the people who can afford tickets. In a recent experiment the owner determined a precise relationship between the price of a ticket and average attendance. At a price of \$5.00 per ticket, 120 people attend a performance. Decreasing the price by a dime (\$0.10) increases attendance by 15. Unfortunately, the increased attendance also comes at an increased cost. Every performance costs the owner \$180. Each attendee costs another four cents (\$0.04). The owner would like to know the exact relationship between profit and ticket price so that he can determine the price at which he can make the highest profit.

Vom Problem zum Programm

The owner of a movie theater has complete freedom in setting ticket prices. The more expensive the ticket charges, the fewer the people who attend a performance. In a recent experiment the owner determined a precise relationship between the price of a ticket and average attendance. At a price of \$5.00 per ticket, 120 people attend a performance. Decreasing the price by a dime (\$0.10) increases attendance by 15.

Under these circumstances, reduced attendance also comes at an increased cost. A performance costs the owner \$180. Reducing the price by four cents (\$0.04). The owner would like to know the exact ticket price so that he can make the highest profit and still cover his costs. At which ticket price can he make the highest profit?

Teilaufgaben klar
definiert.

Lösungsansatz noch
unklar.

Vom Problem zum Programm

- Gesamt-Lösungsansatz noch unklar
- Teilaufgaben klar definiert
- → Bottom-Up Entwurf

Vom Problem zum Programm

```
; Number -> Number  
; compute the number of attendees that  
; pay the given ticket-price  
(define (attendees ticket-price)  
  (+ 120 (* (/ 15 0.1) (- 5.0 ticket-price)))))
```

Anzahl Besucher

Vom Problem zum Programm

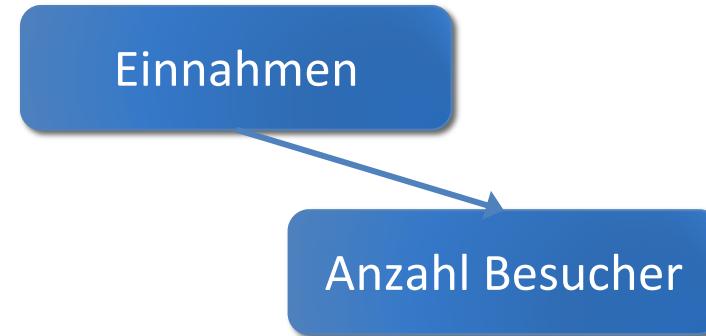
; Number -> Number

; compute the revenue based on the ticket-price and the

; number of attendees which depends on the ticket-price

(define (revenue ticket-price)

(* (attendees ticket-price) ticket-price))



Vom Problem zum Programm

; Number -> Number

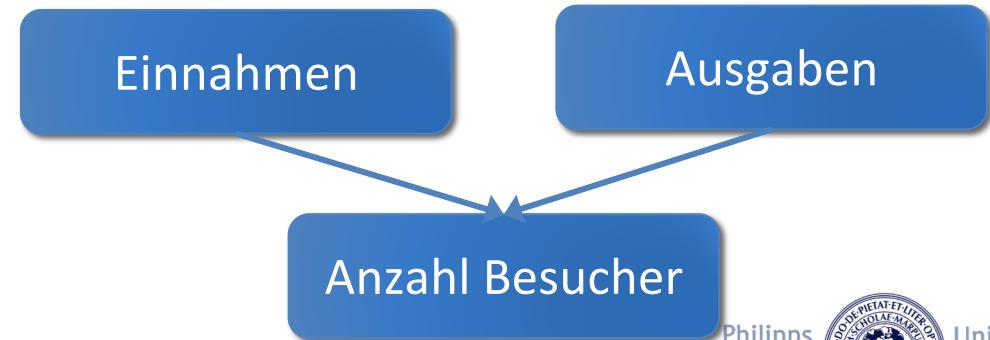
; compute the total costs for the show including the variable

; costs depending on the number of attendees determined

; by the ticket-price

(define (cost ticket-price)

 (+ 180 (* 0.04 (attendees ticket-price))))



Vom Problem zum Programm

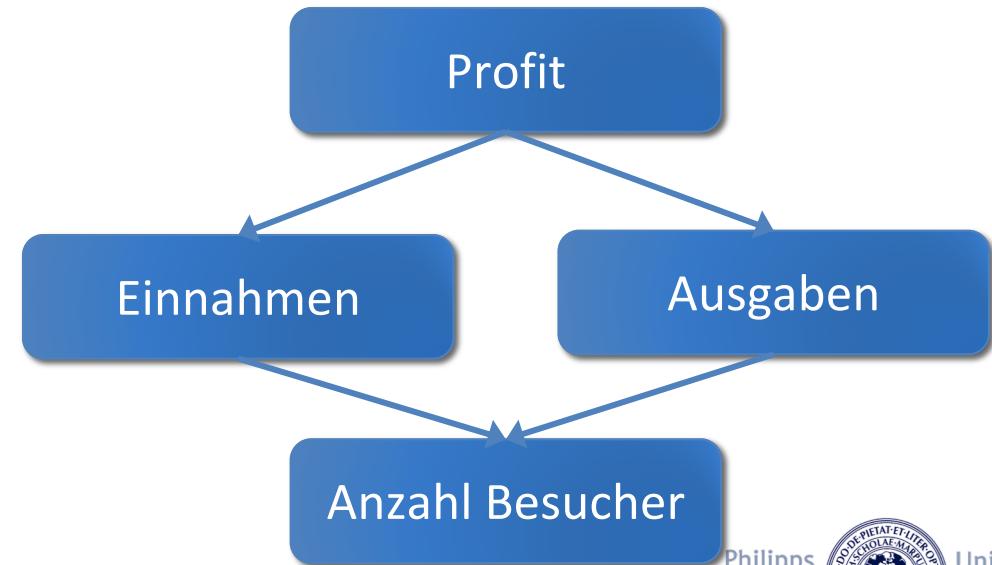
; Number -> Number

; compute the profit depending on the specified ticket-price

(define (profit ticket-price)

 (- (revenue ticket-price)

 (cost ticket-price)))



Vom Problem zum Programm

```
(define (profit price)
  (- (* (+ 120
           (* (/ 15 0.1)
              (- 5.0 price))))
      price)
  (+ 180
     (* 0.04
        (+ 120
           (* (/ 15 0.1)
              (- 5.0 price)))))))
```

Ohne Abstraktion:
Allgemeine Bedeutung
verschleiert;
Redundanzen

Programmentwurf – wie geht das?

- Erkennen von:
 - Was ist relevant?
 - Welche Daten können eingegeben werden?
 - Welche Daten sollen ausgegeben werden?
- Ist eine Funktionalität schon vorhanden?
- Entwurfsrezepte
 - Bewährte Vorgehensweisen

Rezept: Testen

- Stichprobenartiges Überprüfen der Korrektheit
- Aufrufe von definierter Funktion und Vergleich mit erwartetem Resultat
- Definieren von Tests zusammen mit Funktionsdefinition

Tests automatisieren

- Tests persistent definieren (in Datei)
- Auswertung von Ausdruck und Vergleich von Resultat sollte keine manuellen Schritte benötigen
- Dann
 - Tests gehen nicht verloren
 - Tests können wiederholt werden
- Alle Tests nach jeder Code Änderung wiederholen
 - Sicherstellen, dass keine Fehler eingeführt wurden

Tests automatisieren

- Was geschieht bei erfolgreichem/fehlgeschlagenem Test?
- Wie schreibt man Tests auf?
- Beispiel: $(+ 2 3)$ muss 5 ergeben
 - `(check-expect (+ 2 3) 5)`

Funktion in BSL: `check-expect`
Verhalten: Erfolgsmeldung auf Konsole
oder Fehlermeldung in Dialog.

Tests

- Konversion Fahrenheit zu Celsius

```
(check-expect (f2c -40) -40)
```

```
(check-expect (f2c 32) 0)
```

```
(check-expect (f2c 212) 100)
```

```
(define (f2c f)
      (* 5/9 (- f 32)))
```

Fällt Ihnen etwas auf?

Tests dürfen vor
Funktionsdefinition
stehen. “check-*”
sind Sonderfälle.

- Alle Checks werden ausgeführt, wenn Start gedrückt wird

Best Practices

- Schreiben Sie Tests vor der Implementierung
 - Fokus auf Spezifikation
 - Keine Beeinflussung durch Programmierfehler
- Auch Tests nach Implementierung schreiben
 - Eingehen auf Randfälle
 - Bspw. sind alle Fälle von cond-Ausdruck getestet?
- Andere Vergleiche: check-within, check-range, etc.

Best Practices

- Ist dieser Test erfolgreich?

(check-expect (ring 5 10 "red") 

```
(define (ring innerradius outerradius color)
  (overlay (circle innerradius "solid" "white")
           (circle outerradius "solid" color)))
```



Frage der
Repräsentation von
Werten/
Implementierungs-
Details

Best Practices

- Überprüfen von Eigenschaften von Ergebnissen
`(check-expect (image-width (ring 5 10 "red")) 20)`

```
; Number String Image -> Image
; add s to img, y pixels from top, 10 pixels from the left
(check-expect (add-image 5 "hello" (empty-scene 100 100))
  (place-image (text "hello" 10 "red") 10 5 (emptyscene 100 100)))
```

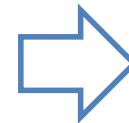
```
(define (add-image y s img)
  (place-image (text s 10 "red") 10 y img))
```

Erwarteter Wert entspricht teilweise reduziertem Funktionsaufruf.
Das ist Zufall!
Implementierung von add-image dadurch nicht festgelegt.

Rezept: Informationen und Daten

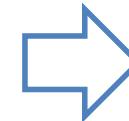
- Programm
 - Verarbeitung von Information
 - Produktion von Information
- “Information“ = Wissen + Bedeutung
 - Wissen und Bedeutung nicht unmittelbar zugänglich für Computer
- Repräsentation durch Daten/ Werte

Das Auto ist 5m lang.



5

Der Name des
Angestellten ist Müller



"Müller"

Werte

- Einem Wert sieht man seine Bedeutung nicht an
- Was ist die Bedeutung von 5?
 - Länge eines Autos?
 - Preis des Essens?
 - Note der Abschlussarbeit?
 - Aktuelle Temperatur?

Datendefinition

- Klasse von Daten
 - Datentyp: Menge von Werten, die auf dieselbe Weise interpretiert werden können
- Name
 - Weist auf die Interpretation hin
- Beispiele
 - ; Distance is a Number.
 - ; interp. the number of pixels from the top margin of a canvas
 - ; Temperature is a Number.
 - ; interp. degrees Celsius

Datendefinition

- Bisher: Zahlen, Strings, Bilder, Wahrheitswerte
- Repräsentation von Informationen auf Basis existierender Datentypen

```
; Temperature is a Number.  
; interp. degrees Celsius  
; Examples:  
(define sunny-weather 25)  
(define bloody-cold -5)
```

Durch Beispiele:
Sicherstellen, dass Werte
repräsentierbar sind

Verwendbar in Tests

Rezept: Funktionsdefinition

- Abfolge von Schritten für den Entwurf einer Funktion

1. Informationsrepräsentation
2. Signatur
3. Tests
4. Hauptfunktion in Unterfunktionen aufteilen
5. Funktionsbody implementieren
6. Tests ausführen
7. Nachbearbeitung

Informationsrepräsentation

- Welche Informationen sind relevant?
- Eingabe
- Ausgabe
- Datendefinition inklusive Beispiele

Signatur

- In BSL: Kommentar
 - Konsumierte Datentypen
 - Produzierte Datentypen
- Beschreibung der Funktionalität
- Funktionskopf

Konsumierte
Datentypen

Produzierter
Datentyp

Beschreibung so
kurz wie möglich.
Antwort auf: "Was
berechnet die
Funktion?"

; Number String Image -> Image
; add s to img, y pixels from top, 10 pixels to the left
(define (add-image y s img)
(empty-scene 100 100))

Funktionskopf

Signatur

- Funktionskopf
 - Funktionsdefinition mit define: "Header"
 - Body-Ausdruck produziert Dummy-Wert: "Stub"

```
; Number String Image -> Image
; add s to img, y pixels from top, 10 pixels to the left
(define (add-image y s img)
  (empty-scene 100 100))
```

Signatur

- Funktionskopf
 - Funktionsdefinition mit define: "Header"
 - Body-Ausdruck produziert Dummy-Wert: "Stub"

```
; Number String Image -> Image  
; add s to img, y pixels from top, 10 pixels to the left  
(define (add-image y s img))
```

Funktionsname

Name der Eingabeparameter

Parameternamen in Beschreibung verwenden

Signatur

- Funktionskopf
 - Funktionsdefinition mit define: "Header"
 - Body-Ausdruck produziert Dummy-Wert: "Stub"

```
; Number String Image -> Image
; add s to img, y pixels from top, 10 pixels to the left
(define (add-image y s img)
  (empty-scene 100 100))
```

Stub macht oberflächlich
korrekt: kein Syntaxfehler,
kein Laufzeitfehler

Tests

- Zwischen Aufgabenbeschreibung und Header
- Mittels `check-`*
- Teil der Dokumentation:
Erwartetes Verhalten
- Automatische Ausführung:
Bei Erfolg auch Zusicherung des durch Tests dokumentierten Verhaltens

Tests

```
; Number -> Number
```

```
; compute the area of a square whose side is len
```

```
(check-expect (area-of-square 2) 4)
```

```
(check-expect (area-of-square 7) 49)
```

```
(define (area-of-square len) 0)
```

- Programmausführung schlägt nun fehl:
Zurückgelieferter Dummy-Wert entspricht nicht dokumentiertem Verhalten

Hauptfunktion in Unterfunktionen aufteilen

- Was kann für Implementierung der Funktion verwendet werden?

- Eingabedaten
- Hilfsfunktionen

- Ersetzen des Dummy-Body durch “Template”

; Number -> Number

; compute the area of a square whose side is len

(check-expect (area-of-square 2) 4)

(check-expect (area-of-square 7) 49)

(define (area-of-square len) ... len ...)

Template

Funktionsbody implementieren

- Template ersetzen durch Ausdruck, der Spezifikation erfüllt
- Spezifikation
 - Signatur
 - Aufgabenbeschreibung
 - Tests

; Number -> Number

; compute the area of a square whose side is len

(check-expect (area-of-square 2) 4)

(check-expect (area-of-square 7) 49)

(define (area-of-square len) (* len len))

Tests ausführen

- Automatisierte Tests: Klick auf Start
- Test erfolgreich: fertig
- Test schlägt fehl
 - Testfall falsch definiert?
 - Funktionsimplementierung fehlerhaft?
 - Reparieren bis Tests erfolgreich

Wirklich?



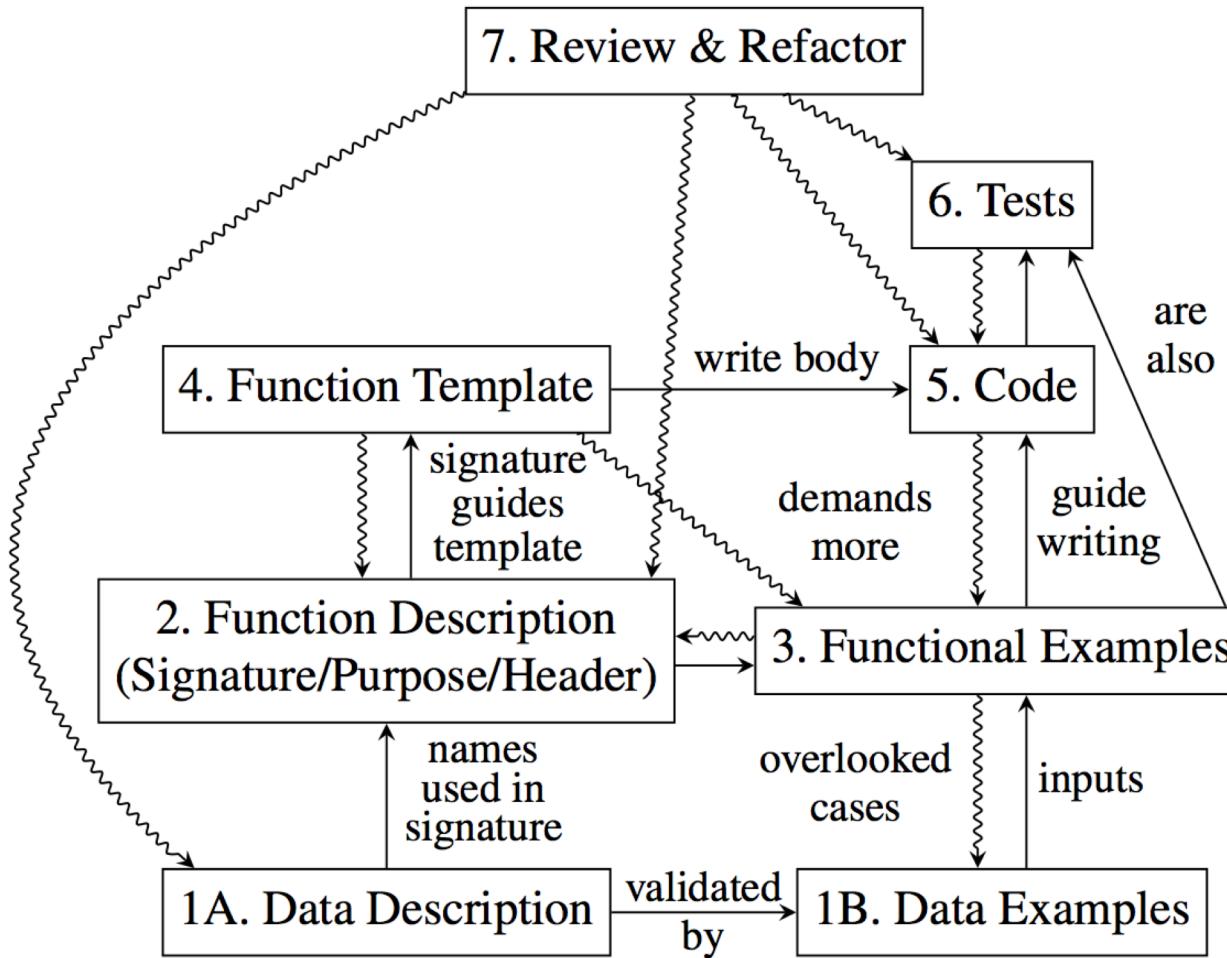
Nachbearbeitung

- Überprüfung
 - Signatur korrekt?
 - Aufgabenbeschreibung korrekt?
 - Stimmen Signatur und Aufgabenbeschreibung mit Implementierung überein?
- Testabdeckung
 - Tests sind erfolgreich, aber ...
 - ... wird auch alles getestet?
 - Wird der gesamte Code während Tests ausgeführt?
 - Werden alle Randfälle getestet?

Nachbearbeitung

- Entspricht Implementierung dem Template?
- Verbesserung der Struktur: “Refactoring”
 - Nicht mehr verwendete Funktions- und Konstantendefinitionen löschen
 - Redundanzen suchen und durch Funktions- und Konstantendefinitionen ersetzen
 - Vereinfachen konditionaler Ausdrücke
- Tests erneut ausführen, um sicherzustellen, dass Funktionalität nicht verändert wurde
 - **Modifikation der Funktionalität muss getrennt von Refactorings geschehen!**

Entwurfsrezept



Rezept: Programme mit vielen Funktionen

- Für jede Funktion Entwurfsrezept anwenden
- In Funktionstemplate definierte Funktionen und Konstanten verwenden
- Typisch: Top-Down
 - Ausgehend von Hauptfunktion
 - Aufteilung in Hilfsfunktionen
 - Erstellen einer "Wunschliste"
 - Header: Signatur, Aufgabenbeschreibung, Funktionsname
 - Nacheinander abarbeiten
 - Bis Liste leer

Stepwise Refinement

- Top-Down Ansatz wird auch “Stepwise Refinement“ genannt
 - Ein großes Entwurfsproblem wird in viele kleine zerlegt
 - Schritt für schritt
 - Bis kleines Problem konkret lösbar
- Nachteile
 - Top-level Entwurfsentscheidungen beeinflussen Bottom-level Hilfsfunktionen
 - Z.B. sind alle benötigten Argumente vorhanden?
 - Testen erst spät möglich
 - Abhilfe: Stub so definieren, dass Tests erfolgreich sind

Test-Stub

```
; Number -> Number
; computes the area of a cube with side length len
(check-expect (area-of-cube 3) 54)
(define (area-of-cube len) (* 6 (area-of-square len)))
```

```
; Number -> Number
; computes the area of a square with side length len
(check-expect (area-of-square 3) 9)
(define (area-of-square len) ( if (= len 3) 9 (error "not yet
implemented"))))
```

Test-Stub

```
; Number -> Number  
; computes the area of a cube with side length len  
(check-expect (area-of-cube 3) 54)  
(define (area-of-cube len) (* 6 (area-of-square len)))
```

Liefert Wert so, dass Test von area-of-cube erfolgreich ist.

```
; Number -> Number  
; computes the area of a square with length len  
(check-expect (area-of-square 3) 9)  
(define (area-of-square len) ( if (= len 3) 9 (error "not yet  
impl")))
```

length len

Absicherung für den Fall,
dass Funktion aus anderem
Kontext aufgerufen wird.

Laufzeitfehler
erzwingen

Information Hiding

- Entwurf in Schichten und Stepwise Refinement
 - Ersetze Implementierung durch Funktionsaufruf
 - Programm wird verständlicher
- Weitere Vorteile
 - Funktionen unabhängig wartbar
 - Wiederverwendung von Funktionen
- Grundlage
 - Aufrufer hat nur Kenntnis der Spezifikation:
 - Signatur, Aufgabenbeschreibung, Tests
 - Implementierung ist verborgen

“Information Hiding”
oder “Geheimnisprinzip”

Information Hiding

(string-append

(body "Tillman" "Rendel")
"your GNB account manager")

Ist dieses Programm
korrekt?

; String String -> String

; generates the pretense of tax refund for the victim fst last

(check-range (string-length (body "Tillman" "Rendel")) 50 300)

(define body fst lst) "")

Implementierung
darf sich verändern.
Muss sich nur an
Spezifikation halten.

Auf dokumentierte
Eigenschaften darf
man sich verlassen.