Konsequent agile Entwicklung mit funktionaler Programmierung

Michael Sperber

- Individualsoftware
- branchenunabhängig
- Scala, Clojure, Erlang, Haskell, F#
- Schulungen, Coaching

www.active-group.de

funktionale-programmierung.de

@ctive group

Agile Manifesto

Liefere funktionierende Software
regelmäßig innerhalb
weniger Wochen oder Monate und
bevorzuge dabei die kürzere Zeitspanne.

Warum nicht Tage oder Stunden?

Modellgetriebene Entwicklung

"MDD is an approach to software development where extensive models are created before source code is written."

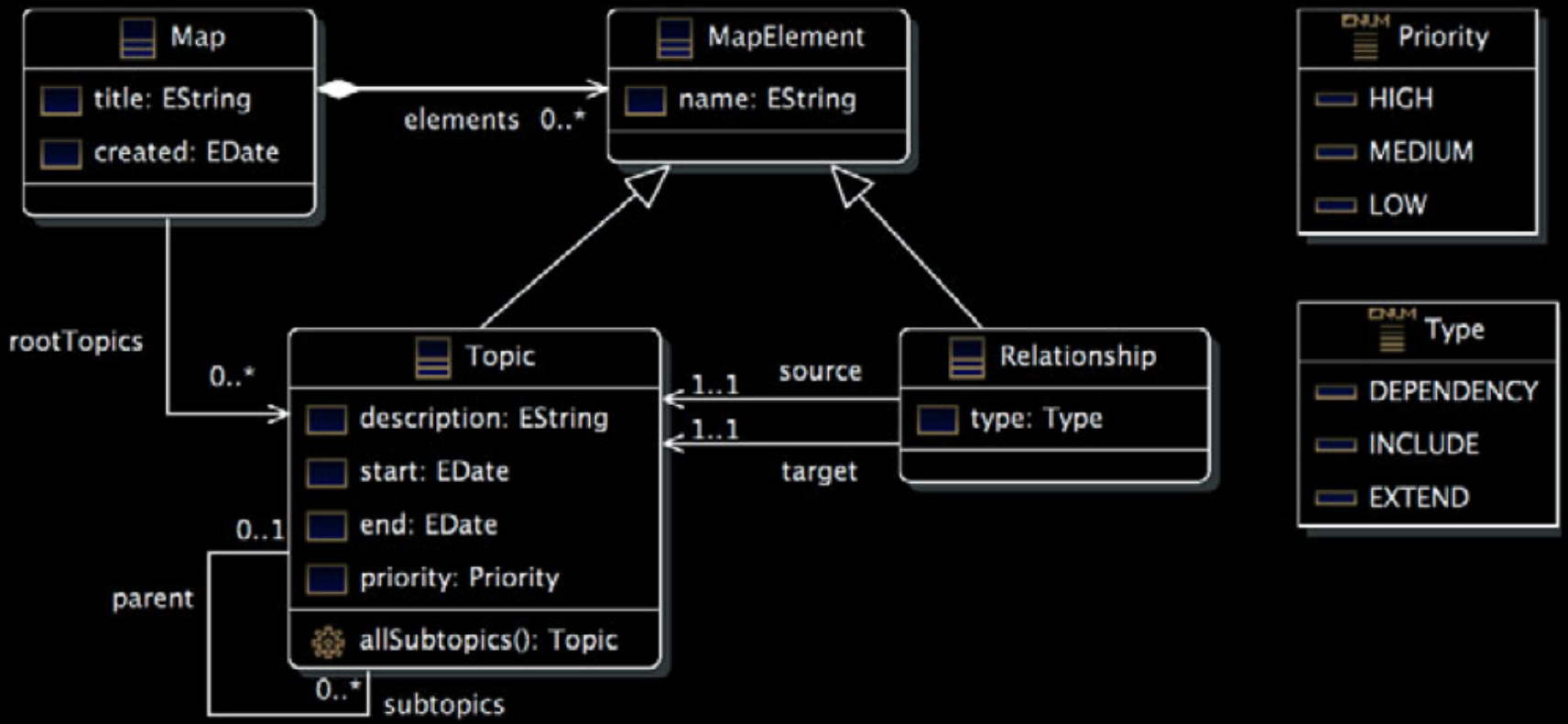
http://www.agilemodeling.com/essays/amdd.htm

Modellgetriebene Entwicklung

"In Model-Driven Development the model of a software application is specified on a higher abstraction level than traditional programming languages."

http://www.theenterprisearchitect.eu/blog/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development/

EMF



Code-Generierung

```
public String getDescription() {
    return description;
}
public void setDescription
    (String newDescription) {
     String oldDescription = description;
     description = newDescription;
     if (eNotificationRequired())
         eNotify(...));
}
```

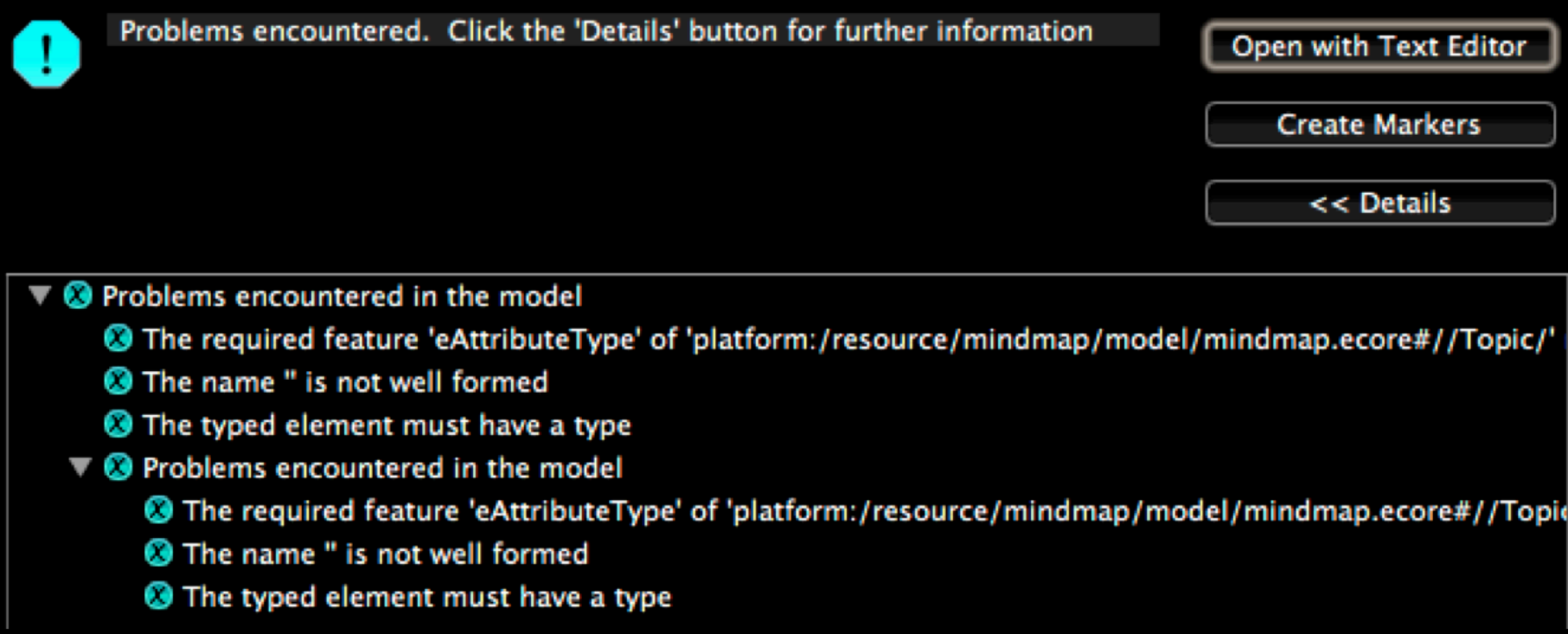
Höherstehende Funktionalität

```
public EList<Topic> allSubtopics() {
        // TODO: implement this method
        // Ensure that you remove
        // @generated or mark it
        // @generated NOT
   throw new
     UnsupportedOperationException();
}
```

Object Constraint Language

```
self->closure(subtopics)
```

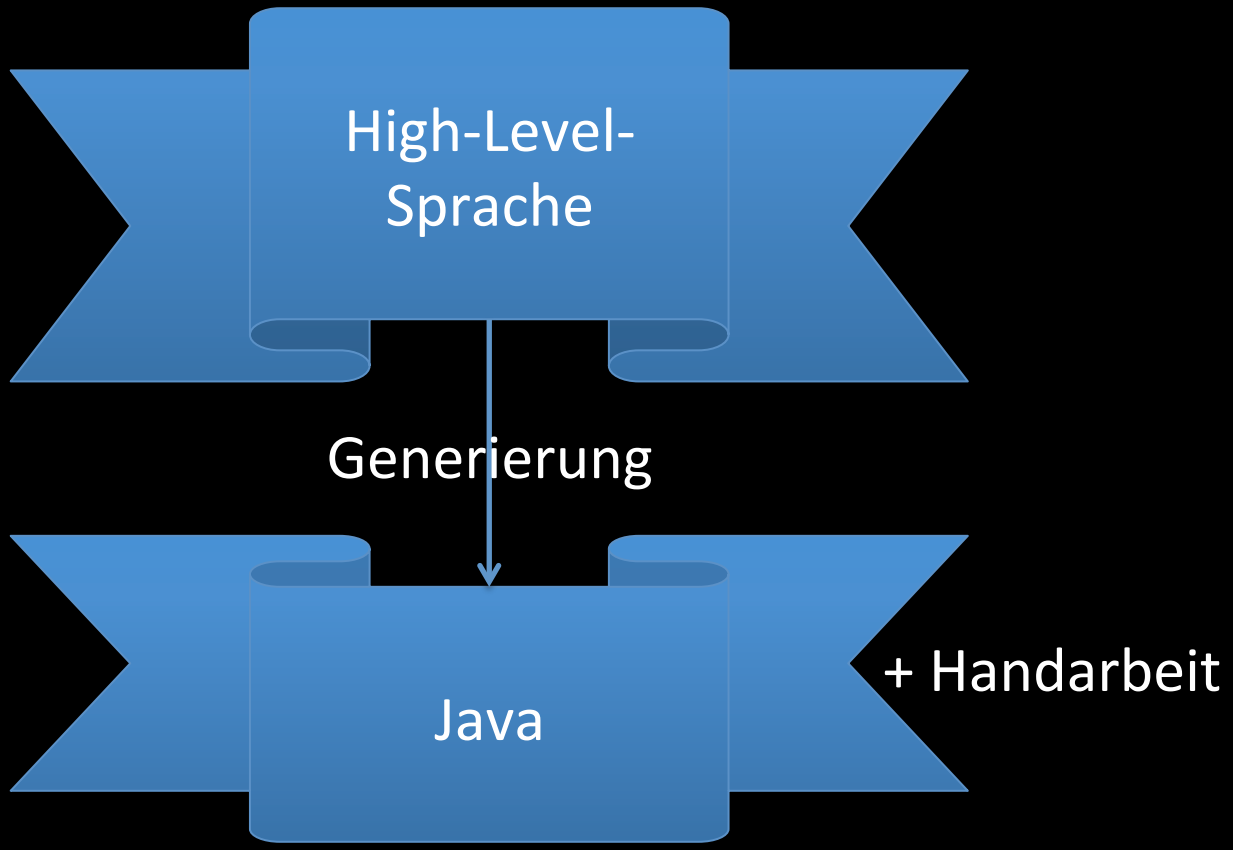
EMF in der Praxis



Sprachen im EMF-Umfeld

- OCL
- Xtext
- OTL
- QVT
- EMF Query
- Xpand
- [Xtend]
- …

Modeling-Stufen



Modellgetriebene Entwicklung

"In Model-Driven Development the model of a software application is specified on a higher abstraction level than traditional programming languages."

http://www.theenterprisearchitect.eu/blog/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development/

Mindmap in Racket

```
(define-record-procedures topic
  make-topic topic?
  (topic-description
   topic-start topic-end
   topic-priority
   topic-subtopics))

(: make-topic
   (string date date priority
    (list-of topic) -> topic))
```

Mindmap in Racket

```
(define fp
  (make-topic "Functional Programming"
              "1990-01-01" "9999-99-99"
              "high"
              empty))

(define oop
  (make-topic "Object-Oriented Programming"
              "1989-01-01" "2001-01-01"
              "low"
              empty))
```

Mindmap in Racket

```
(define-record-procedures mind-map
  make-mind-map mind-map?
  (mind-map-title
   mind-map-created
   mind-map-elements))

(: make-mind-map
   (string date (list-of map-element)
         -> mind-map))
```

Mindmap in Racket

```
(define-record-procedures relationship
  make-relationship relationship?
  (relationship-type
   relationship-source
   relationship-target))

(: make-relationship
   (type topic topic -> relationship))
```

Mindmap in Racket

```
(define map-element
  (signature
   (mixed topic relationship)))
```

Mindmap in Racket

```
(define priority
  (signature
   (one-of "high" "medium" "low")))

(define type
  (signature
   (one-of "dependency”
           "include"
           "extend")))
```

Mindmap in Racket

```
(define programming
  (make-topic "Programming"
              "0000-00-00" "9999-99-99"
              "high"
              (list fp oop)))

(define map1
  (make-mind-map "Mike's idle thoughts"
                 "2014-05-19"
                 (list programming)))
```

Mindmap in Racket

```
(: topic-all-subtopics
   (topic -> (list-of topic)))
(define topic-all-subtopics
  (lambda (t)
    (append
      (topic-subtopics t)
      (concatenate
            (map topic-all-subtopics
                 (topic-subtopics t))))))
```

Object Constraint Language

```
self->closure(subtopics)
```

Abstraktion

```
(: closure
   ((%a -> (list-of %a))
      -> (%a -> (list-of %a))))
(define closure
  (lambda (subthings)
    (lambda (thing)
      (append (subthings thing)
              (concatenate
                (map (closure subthings)
                     (subthings thing)))))))
```

Agile Manifesto

Liefere funktionierende Software
regelmäßig innerhalb
weniger Wochen oder Monate und
bevorzuge dabei die kürzere Zeitspanne.

Stunden!

Ist das DDD?

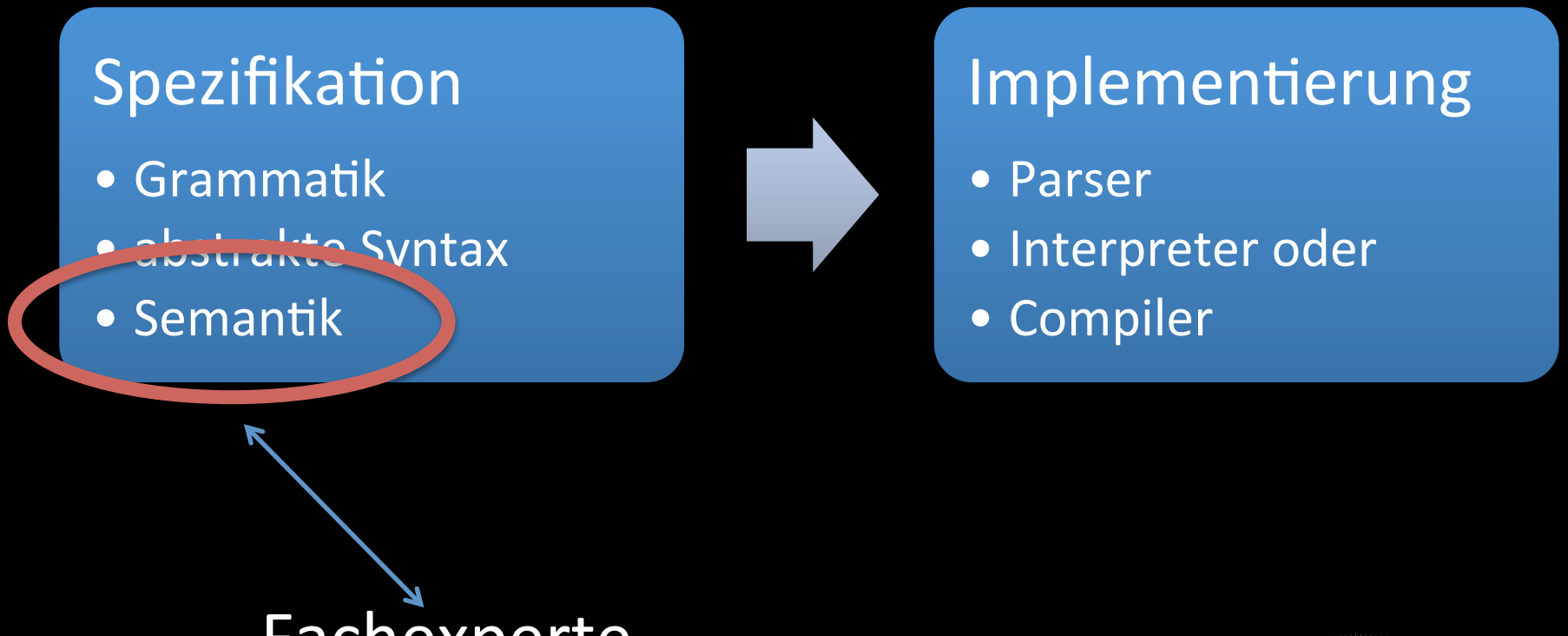
Aber:

Modelle meist blutleer

Agile Manifesto

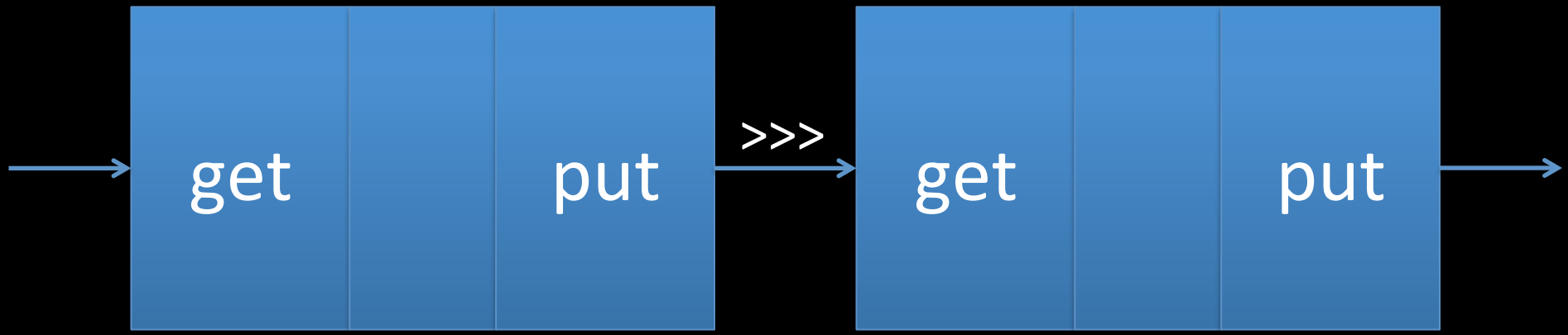
Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.

DSLs



Eingebettete DSL

Stream-Prozessoren



Stream-Programme

```
(get x)
(get y)
(put (* x y))
(stop)
```

Stream-Prozessor in CPS

```
(make-get
  (lambda (x)
    (make-get
      (lambda (y)
        (make-put (+ x y)
          (lambda ()
            (stop)))))))))
```

Neue Syntax mit Makro

```
(define-syntax stream-processor
  (syntax-rules (get put when)
    ((stream-processor) (stop))
    ((stream-processor (get ?v) ?clauses ...)
     (make-get (lambda (?v)
                 (stream-processor ?clauses ...))))
    ((stream-processor (put ?exp) ?clauses ...)
     (make-put ?exp (lambda () (stream-processor ?clauses ...))))
    ((stream-processor (when ?cond ?cons ...) ?clauses ...)
     (if ?cond
         (stream-processor ?cons ...)
         (stream-processor ?clauses ...)))
    ((stream-processor ?clause ?clauses ...) ?clause)))
```

Neue Syntax mit Makro

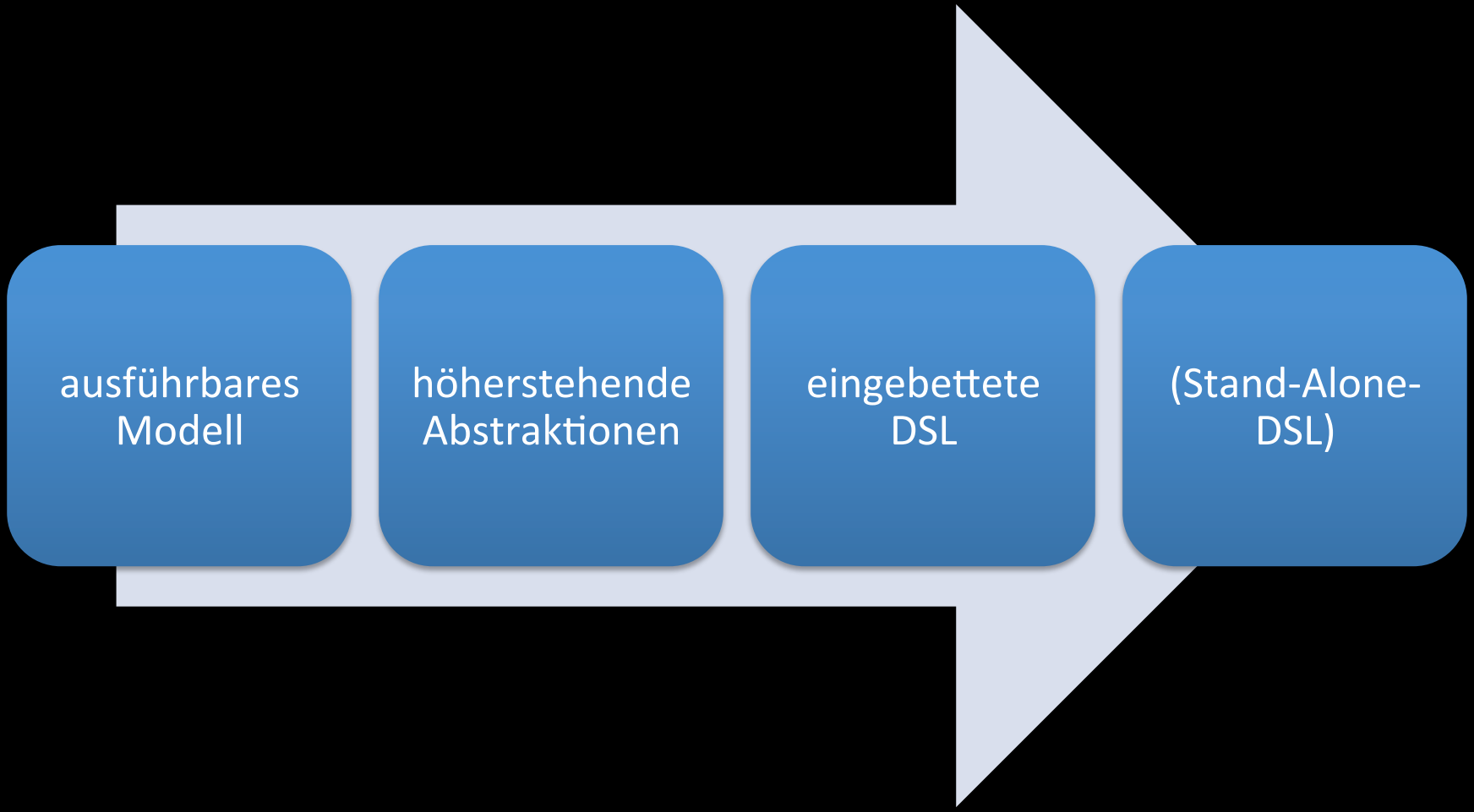
```
...
((stream-processor (get ?v) ?clauses ...)
 (make-get (lambda (?v)
             (stream-processor ?clauses ...))))
...
```

DSL-Programm

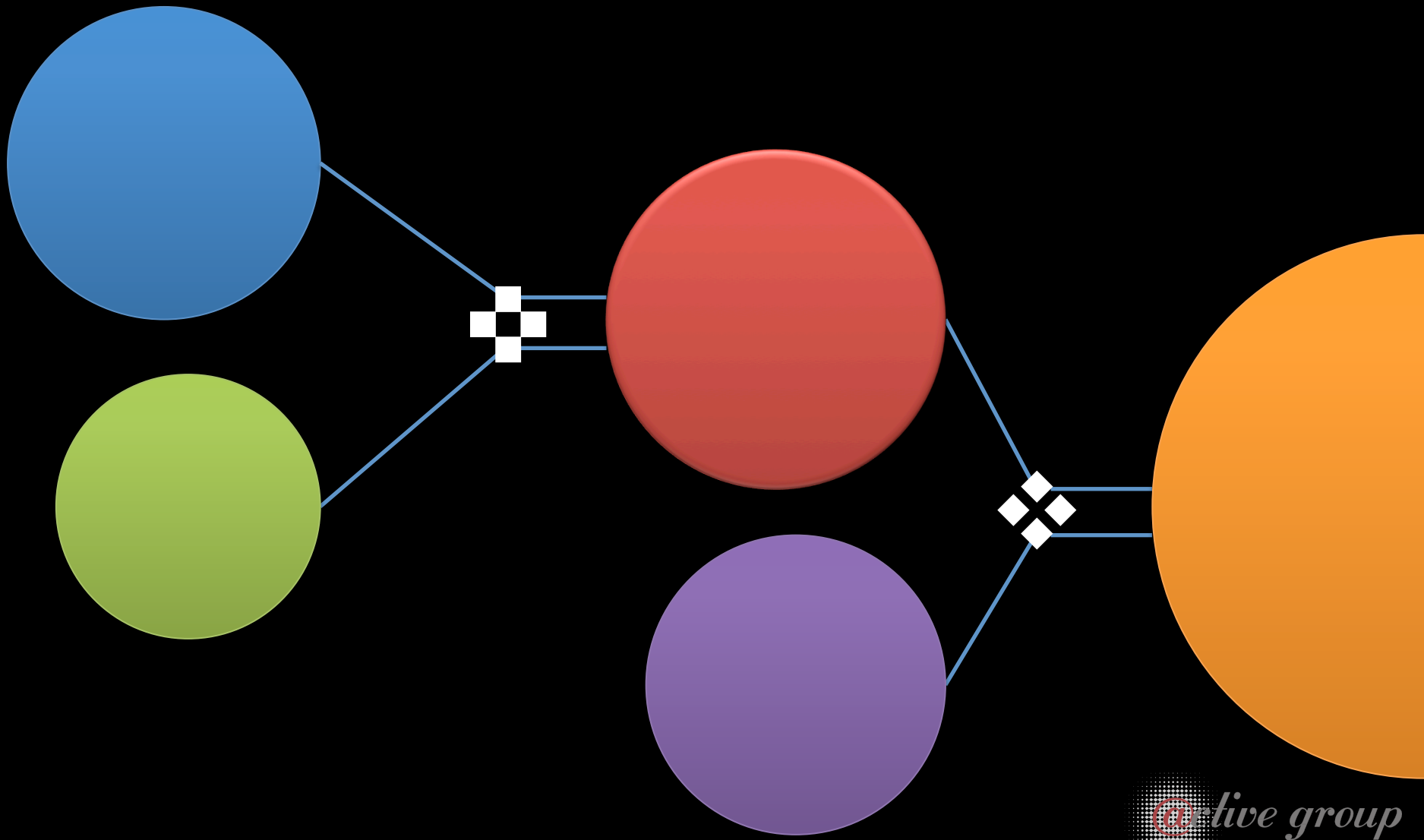
```
(define sp-filter
  (lambda (p)
    (stream-processor
     (get x)
     (when (p x)
       (put x)
       (sp-filter p))
     (sp-filter p))))
```

Alternative: Monaden

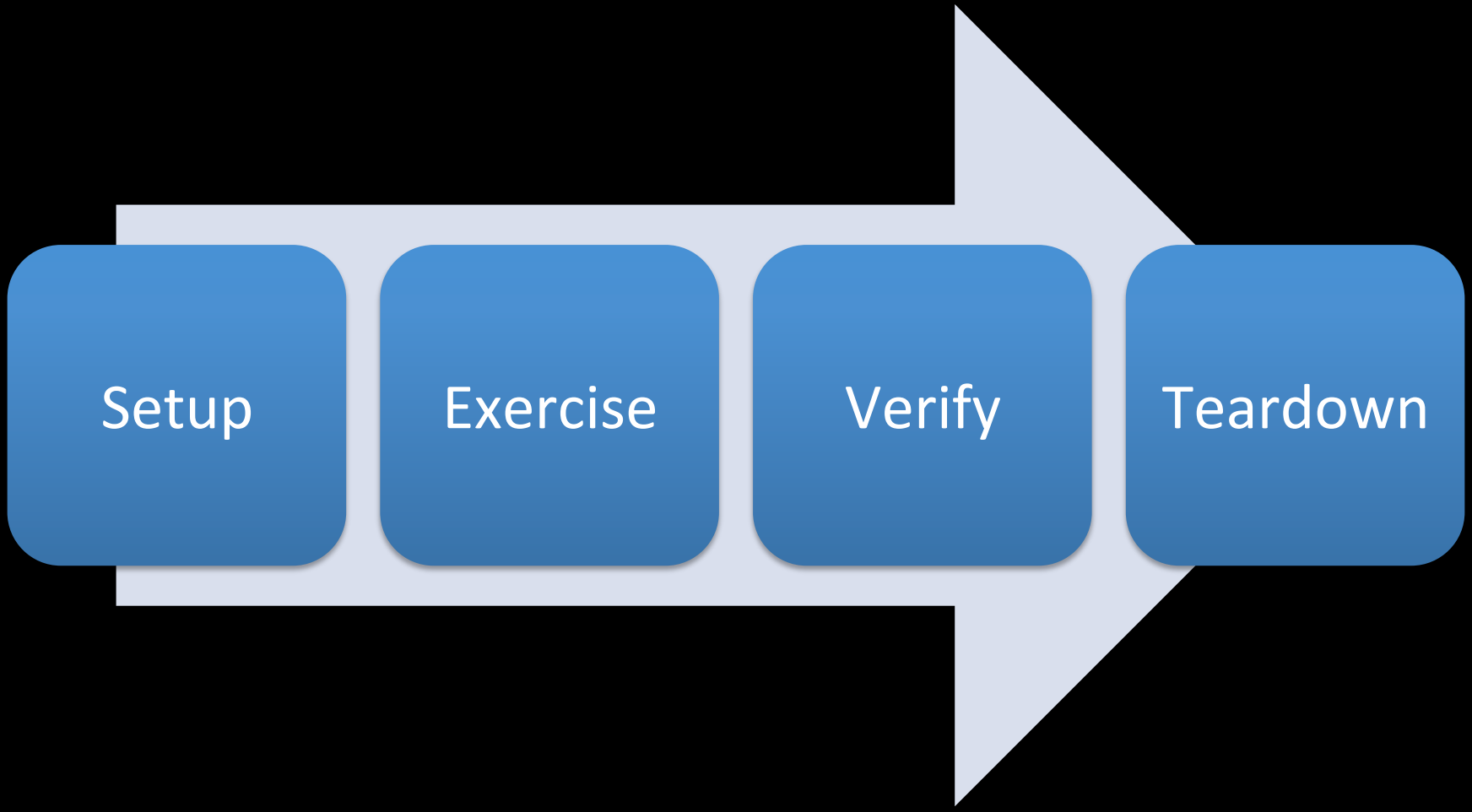
Funktionale Programmierung



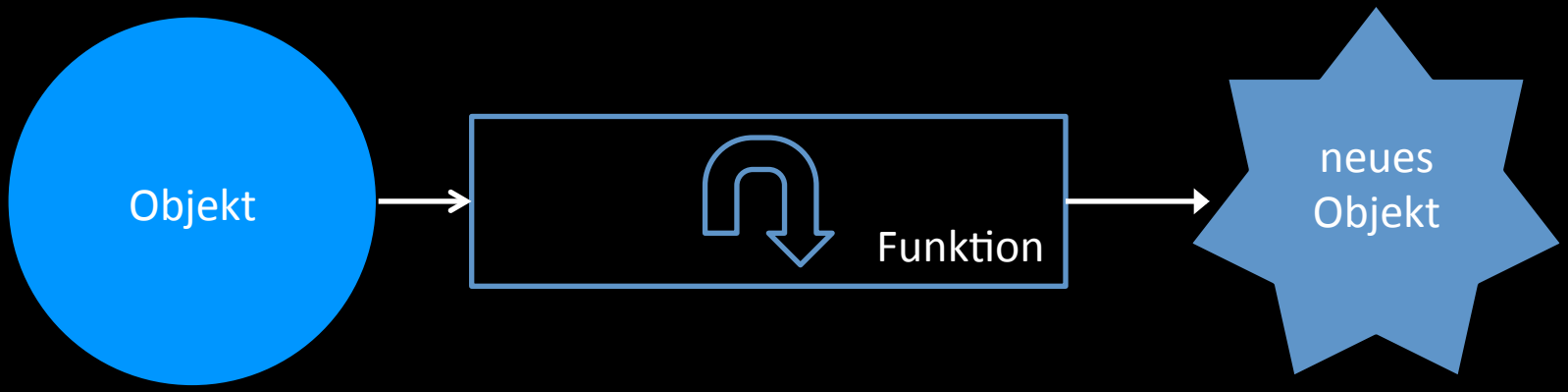
Kompositionale Modelle



xUnit



FP vs. OO



OO:

mutierbares
Objekt

Unveränderbare Werte

http://blogs.msdn.com/b/ericlippert/

Werte statt Effekte

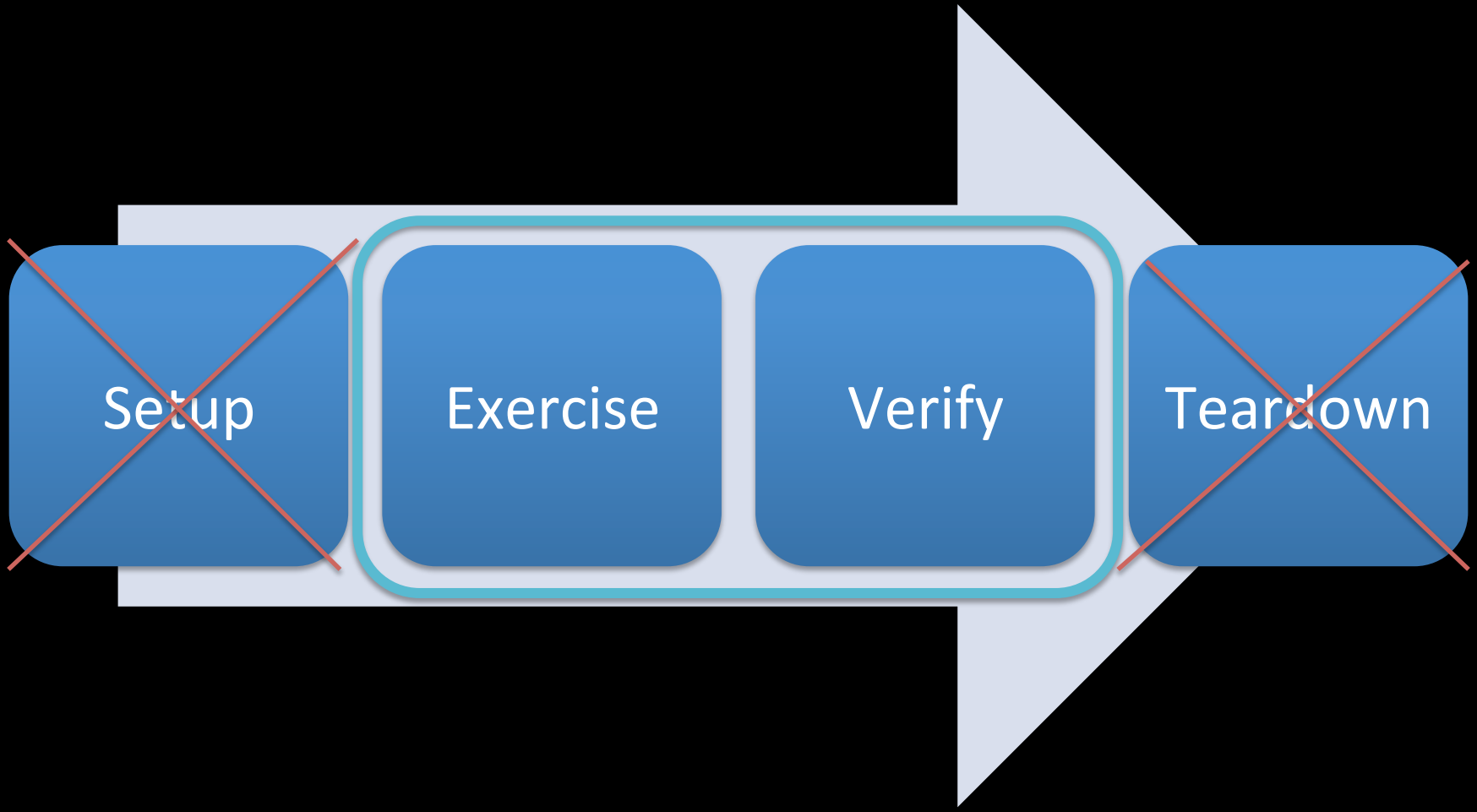
```
(define sp1
  (stream-processor
   (put 5) (put 3)))
(define sp2
  (stream-processor
   (get x) (get y)
   (put (+ x y))))
```

```
Welcome to DrRacket, version 6.0.1 [3m].
Language: racket.
> (take 1 (run (>>> sp1 sp2)))
'(8)
```

RackUnit

```
(check-equal?
 (take 1
   (run (>>> sp1 sp2)))
 '(8))
```

xUnit



Spezifikationsbasiertes Testen

```
(check-property
 (for-all ((a number)
           (b number))
    (= (+ a b)
       (+ b a))))
```

Spezifikationsbasiertes Testen

```
(check-property
 (for-all ((a number)
           (b number)
           (c number))
   (= (+ a (+ b c))
      (+ (+ a b) c))))
```

```
Property falsifiable with a = -4.0 b = 0.6666666666666666 c = 0.6666666666666667
in properties.rkt, line 19, column 0
```

Spezifikationsbasiertes Testen

```
(define associativity
  (lambda (op sig =?)
    (for-all ((a sig)
              (b sig)
              (c sig))
      (=? (op a (op b c))
          (op (op a b) c)))))
```

Funktionale Programmierung und Softwarearchitektur

- Modelle sind Code
- Objekte sind unveränderbare Werte
- eingebettete DSLs sind alltäglich
- Eigenschaften sind Tests

Agile Manifesto

Unsere höchste Priorität ist es,
den Kunden durch frühe und
kontinuierliche Auslieferung
wertvoller Software
zufrieden zu stellen.

Konsequent agile Entwicklung mit funktionaler Programmierung

funktionale-programmierung.de

ENTWICKLERTAG

meet the SPEAKER @speakerlounge

1. OG DIREKT ÜBER DEM EMPFANG