



Objektorientiertes Programmieren

Machen wir irgendwas falsch?

Jürgen Börstler

Department of Computing Science

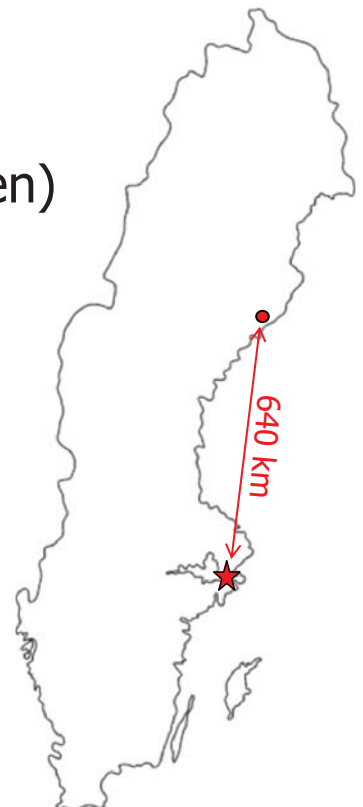
Umeå University, Sweden

jubo@cs.umu.se

Umeå University



- ◆ Gegründet 1965
- ◆ Campus in Stadtnähe
- ◆ 29 000 Studierende (1 300 DoktorandInnen)
- ◆ 3 900 Angestellte
- ◆ 300 ProfessorInnen
- ◆ 320 Mio € Umsatz
- ◆ 70 Institute/Centren/etc.
- ◆ 180 Ausbildungsprogramme
- ◆ 2300 Kurse
- ◆ Bologna angepasst



Machen wir irgendwas falsch?



Drei Antworten

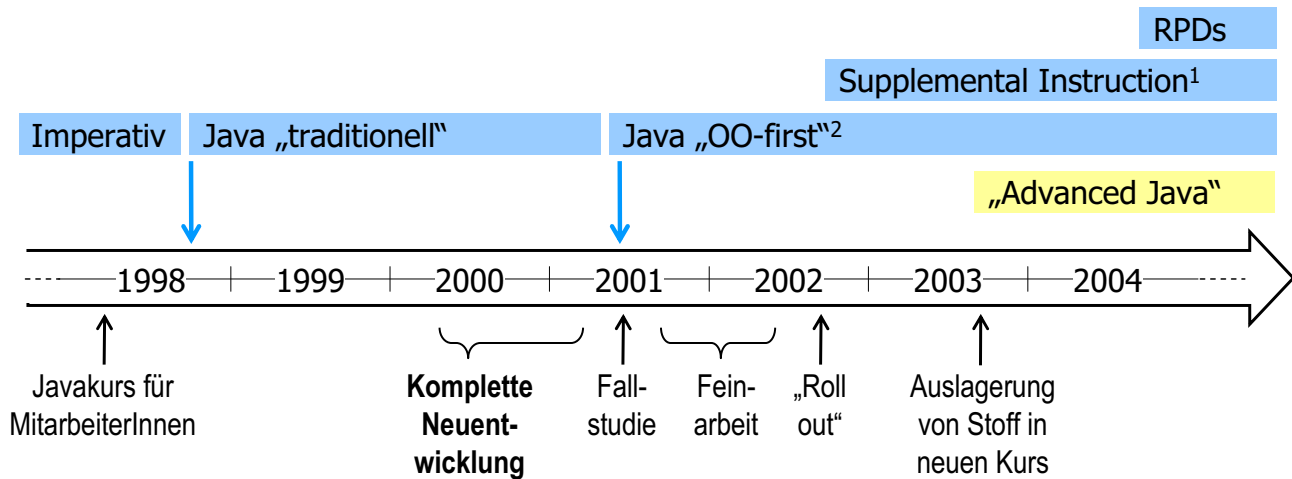
1. Eine Kurze
2. Eine Lokale
3. Eine Lange

Die kurze Antwort



- ◆ Ja, offensichtlich machen wir etwas falsch
 - ❑ Flut von Berichten über Probleme
 - ❑ Dieselben Probleme wie vor 10 Jahren
 - ◆ Aber
 - ❑ Eine Menge von positiven Erfahrungen
 - ❑ Wenig Forschung um mögliche Ursachen
 - ❑ Sehr sehr wenige generalisierbare Ergebnissen
- Wir wissen nur daß es möglich ist, aber nicht wie (falls wir den positiven Studien glauben schenken)

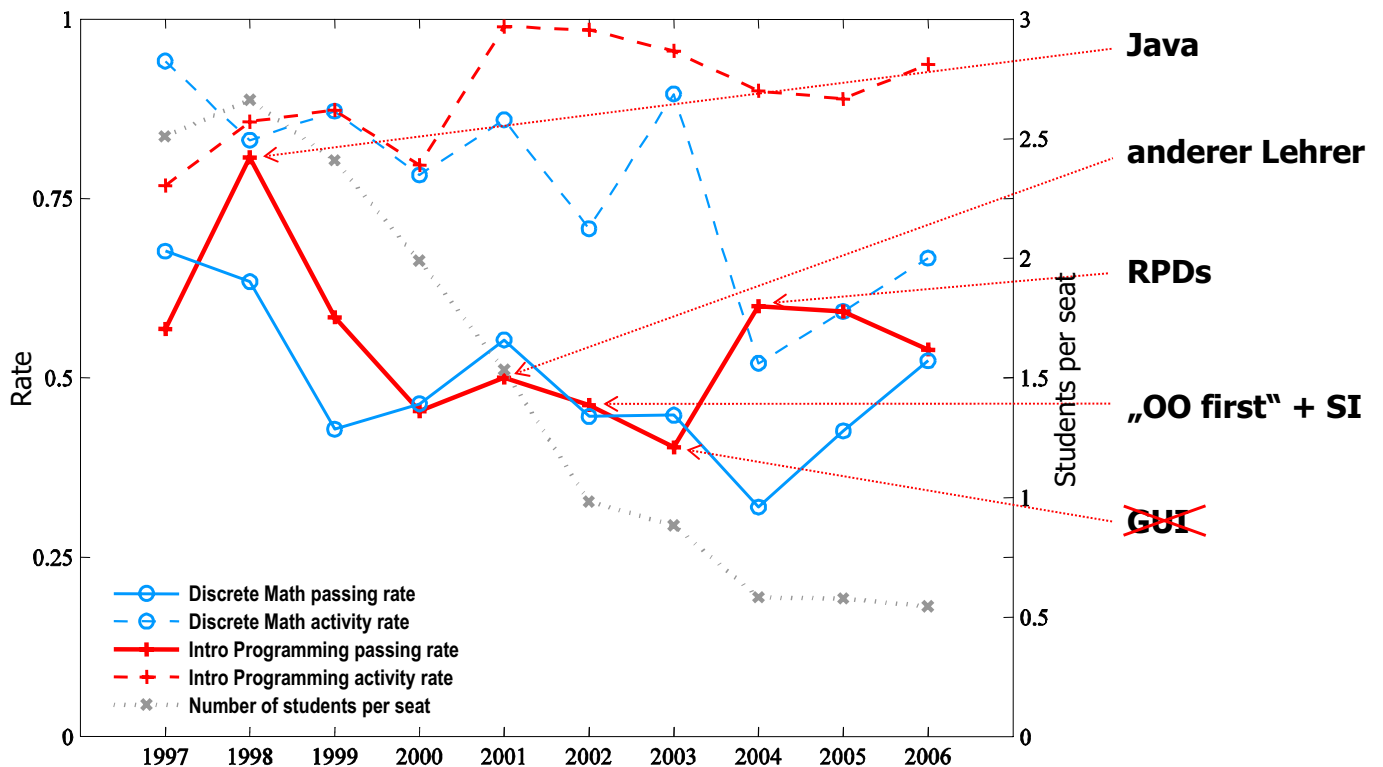
Die lokale Antwort: Java/OO in Umeå



¹ SI: Regelmässige Selbsthilfegruppen ≠ traditionelle Übungen siehe <http://www.umkc.edu/cad/si>

² Vom ersten Tag an; BlueJ, CRC-Karten, gesteuerter Übungsbetrieb, separate Theorie- und Praxis-Klausuren

Hat es geholfen?



Analyse einiger Daten



- ◆ Erfolg in der Klausur korreliert nicht mit
 - ❑ Leistungen in Mathematik (weder Oberstufe noch Uni)
 - ❑ Abgabe von obligatorischen Übungen
 - ❑ Aufbau und Inhalt des Kurses

- ◆ Positive Indikatoren (aber unzureichende Datenmenge)
 - ❑ Mind. 2 Programmierkurse in der Oberstufe
 - ❑ Einführung von CRC/RPD

- ◆ Statistisch signifikante Korrelation mit Klausurnote
 - ❑ Anwesenheit in Vorlesung/Übung (alle Studierende)
 - ❑ Teilnahme an SI (nur Studierende ohne Vorkenntnisse)

Die lokale Antwort



- ◆ Ja, wir machen irgendwas falsch
 - ❑ Wir haben eine Menge versucht
 - ❑ Immer noch dieselben Probleme

- ◆ Aber
 - ❑ Eine Menge von Erfahrungen
 - ❑ Eine Menge möglicher Ursachen
 - ❑ Sehr wenige greifbare Ergebnissen
 - ❑ Viele Probleme haben nichts mit der Objektorientierung zu tun

Die lange Antwort



- ◆ Gibt es überhaupt ein Problem?
- ◆ Ist die Objektorientierung schwieriger?
- ◆ Unsere Rahmenbedingungen
- ◆ Ursachenforschung
- ◆ Erfolgsfaktoren
- ◆ Zusammenfassung

Gibt es überhaupt ein Problem?



Studierende im 1.-2. Semester

- ◆ McCracken et al.
 - Allgemeine Defizite im Problemlösen
- ◆ Lister et al.
 - Schwierigkeiten im Verfolgen von Kod
- ◆ Dehnadi/Bornat, Ma et al.
 - Schwierigkeiten mit Semantik der Zuweisung, insb. für Referenzen

Studierende im letzten Semester

- ◆ Eckerdal et al.
 - Große Mängel im Design von Software

McCracken et al. (2001)



- ◆ Was: Programmierung, Praxis
- ◆ Wann: Im 2. Semester

- ◆ Wer: 216 Studierende von 4 Universitäten
- ◆ Wie: Konkrete [Programmieraufgabe](#)
(einfacher Taschenrechner; Kommandozeile)

- ➔ Ergebnis: Im Schnitt knapp 23 von 110 Punkten

- ➔ Analyse: Hauptproblem war die Umsetzung der Aufgabenbeschreibung in eine konkret zu lösende Programmieraufgabe

Lister et al. (2004)



- ◆ Was: Programmierung, Theorie
- ◆ Wann: Ende des 1. Semesters (oder direkt danach)

- ◆ Wer: 556 (941) Studierende von 12 Universitäten
- ◆ Wie: 12 Fragen (MC); verfolgen von Kod

Question 3.

Consider the following code fragment:

```
int [] x = {1, 2, 3, 3, 3};
boolean b[] = new boolean[x.length];

for ( int i = 0; i < b.length; ++i )
    b[i] = false;

for ( int i = 0; i < x.length; ++i )
    b[ x[i] ] = true;

int count = 0;

for ( int i = 0; i < b.length; ++i )
{
    if ( b[i] == true ) ++count;
}
```

After this code is executed , "count" contains:

a) 1 b) 2 c) 3 d) 4 e) 5

Lister et al. (2004)

- ◆ Was: Programmierung, Theorie
- ◆ Wann: Ende des 1. Semesters (oder direkt danach)
- ◆ Wer: 556 (941) Studierende von 12 Universitäten
- ◆ Wie: 12 Fragen (MC); verfolgen von Kod

→ Ergebnis:

Anzahl richtige Antworten	0-4	5-7	8-9	10-12
Anzahl Studierende in %	23	25	24	27
Ohne größte Institution	29	26	25	20

- Analyse: Die Probleme der Studierenden liegen auf grundlegenderem Nivå (als McCracken et al.)

Dehnadi/Bornat (2006), Ma et al. (2007), ...



- ◆ Was: Zuweisung (und Sequenz)
- ◆ Wann: Im/nach dem 1. Programmierkurs
- ◆ Wer: 60–100 Studierende (mehrfach repliziert)
- ◆ Wie: 12+ MC Fragen + 1 offene Frage; Antworten korrespondieren zu möglichen Semantiken der Zuweisung

Beispielfrage



<p>1. Read the following statements and tick the box next to the correct answer in the next column.</p> <pre>int a = 10; int b = 20; a = b;</pre>	<p>The new values of a and b are:</p> <ul style="list-style-type: none"><input type="checkbox"/> a = 10 b = 10<input type="checkbox"/> a = 30 b = 20<input type="checkbox"/> a = 0 b = 10<input type="checkbox"/> a = 20 b = 20<input type="checkbox"/> a = 0 b = 30<input type="checkbox"/> a = 10 b = 20<input type="checkbox"/> a = 20 b = 10<input type="checkbox"/> a = 20 b = 0<input type="checkbox"/> a = 10 b = 30<input type="checkbox"/> a = 30 b = 0 <p>Any other values for a and b:</p> <p>a = b = a = b = a = b =</p>	<p>Use this column for your rough notes please</p>
---	--	---

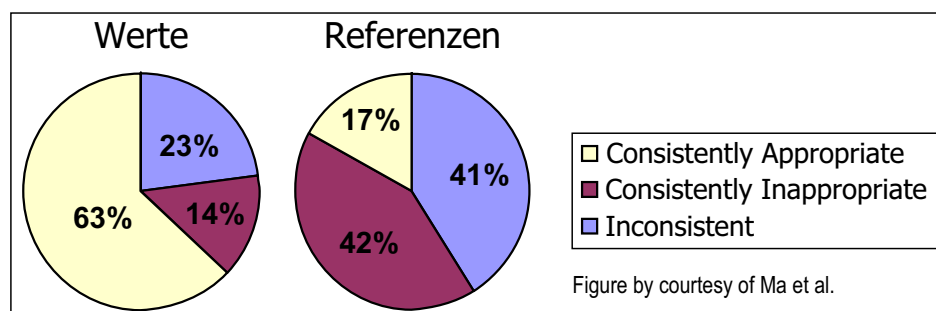
Question	Answer/s	Model/s
1. <pre>int a = 10; int b = 20; a = b;</pre>	a = 20 b = 0	M1 (a ← b & b ← 0)
	a = 20 b = 20	M2 (a ← b & b unchanged)
	a = 0 b = 10	M3 (b ← a & a ← 0)
	a = 10 b = 10	M4 (b ← a & a unchanged)
	a = 30 b = 20	M5 (a ← (a+b) & b unchanged)
	a = 30 b = 0	M6 (a ← (a+b) & b ← 0)
	a = 10 b = 30	M7 (b ← (a+b) & a unchanged)
	a = 0 b = 30	M8 (b ← (a+b) & a ← 0)
	a = 10 b = 20	M9 (a unchanged & b unchanged)
	a = 20 b = 10	M11 (a, b swap values simultaneously)
	a = 20 b = 20 a = 10 b = 10	M10 (a = b)

Table by courtesy of Dehnadi/Bornat

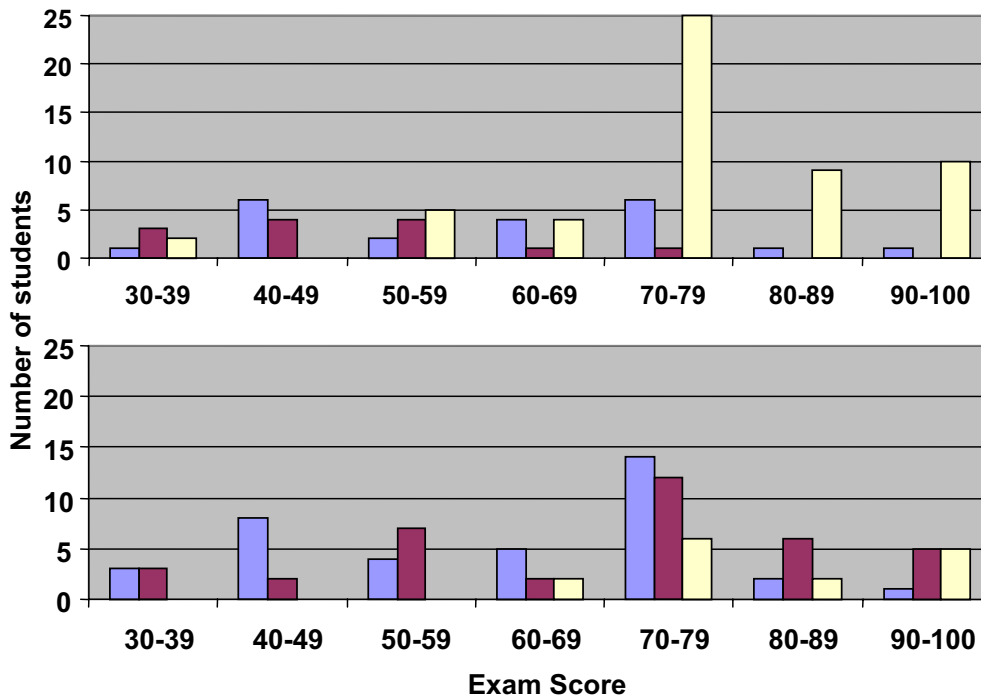
Dehnadi/Bornat (2006), Ma et al. (2007), ...

- ◆ Was: Zuweisung (und Sequenz)
- ◆ Wann: Im/nach dem 1. Programmierkurs
- ◆ Wer: 60–100 Studierende (mehrfach repliziert)
- ◆ Wie: 12+ MC Fragen + 1 offene Frage; Antworten korrespondieren zu möglichen Semantiken der Zuweisung

→ Ergebnis:
(Ma et al.)



Analyse der Kursresultate



Zuweisung mit Werten

Zuweisung mit Referenzen

Consistently Appropriate
Consistently Inappropriate
Inconsistent

Figure by courtesy of Ma et al.

Eckerdal et al. (2006)



- ◆ Was: Softwareentwurf
- ◆ Wann: Ende des Studiums
- ◆ Wer: 100+ Studierende unterschiedlicher Univ.
- ◆ Wie: Entwurf eines „Superweckers“
- ➔ Ergebnis: Partiiell korrekte Lösungen – 9%
Kein anwendbarer Lösungsansatz – 62%
- ➔ Analyse: Je mehr abgeschlossene Kurse in Informatik, desto bessere Ergebnisse

Gibt es überhaupt ein Problem?



- ◆ Ja
- ◆ Größer als vermutet
- ◆ In Theorie und Praxis
- ◆ Grundlegende Konzepte, Programmierung, Entwurf und Problemlösung
- ◆ Nicht auf AnfängerInnen beschränkt

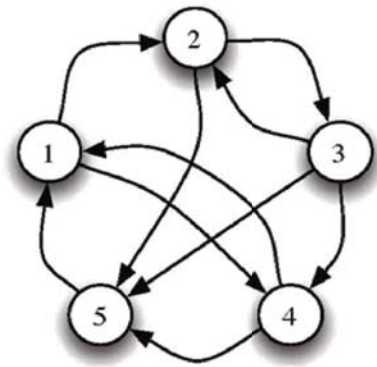
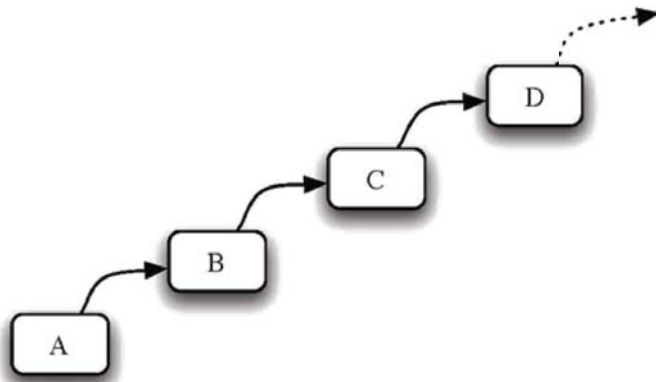
Nicht auf Objektorientierung beschränkt!?

Die lange Antwort

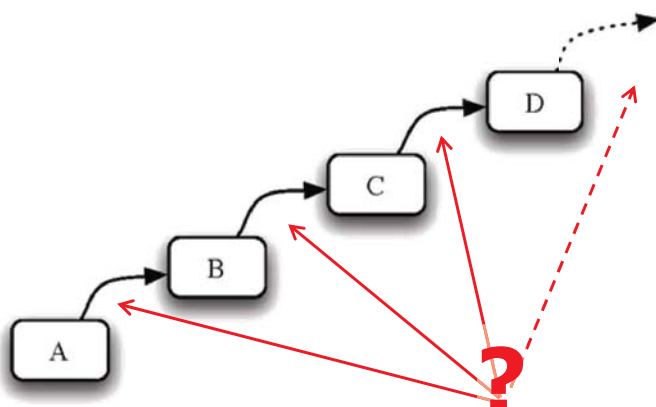


- ◆ Gibt es überhaupt ein Problem? → **Ja**
- ◆ Ist die Objektorientierung schwieriger?
- ◆ Unsere Rahmenbedingungen
- ◆ Ursachenforschung
- ◆ Erfolgsfaktoren
- ◆ Zusammenfassung

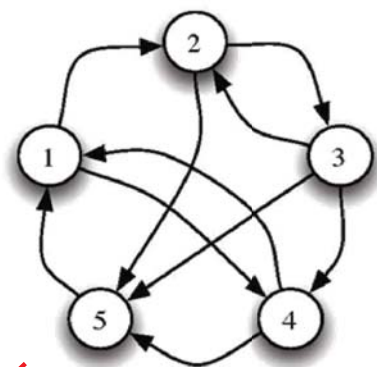
Objektorientierung ist anders



Trotzdem sehen die meisten Kurse/Bücher wie vorher aus



Wo „passt“ OO
am besten rein



Welcher Ansatz ist schwieriger



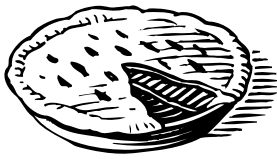
- ◆ OO ist einfacher, weil natürlicher
- ◆ OO ist schwieriger
 - Mehr Stoff
 - Steilere „Lernkurve“
- ◆ Kein Unterschied
 - C++/prozedural verglichen mit Java/OO
 - BefürworterInnen von OO

Ist OO natürlicher?

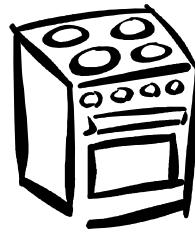


- ◆ Wer kennt die aufgelaufenen Verspätungsgebühren?

Ist OO natürlicher?



Backwerk
...
...
bake()?
...



Backofen
...
...
bake()?
...

- ◆ Wer ist verantwortlich für den Backvorgang?

Ist die Objektorientierung schwieriger?



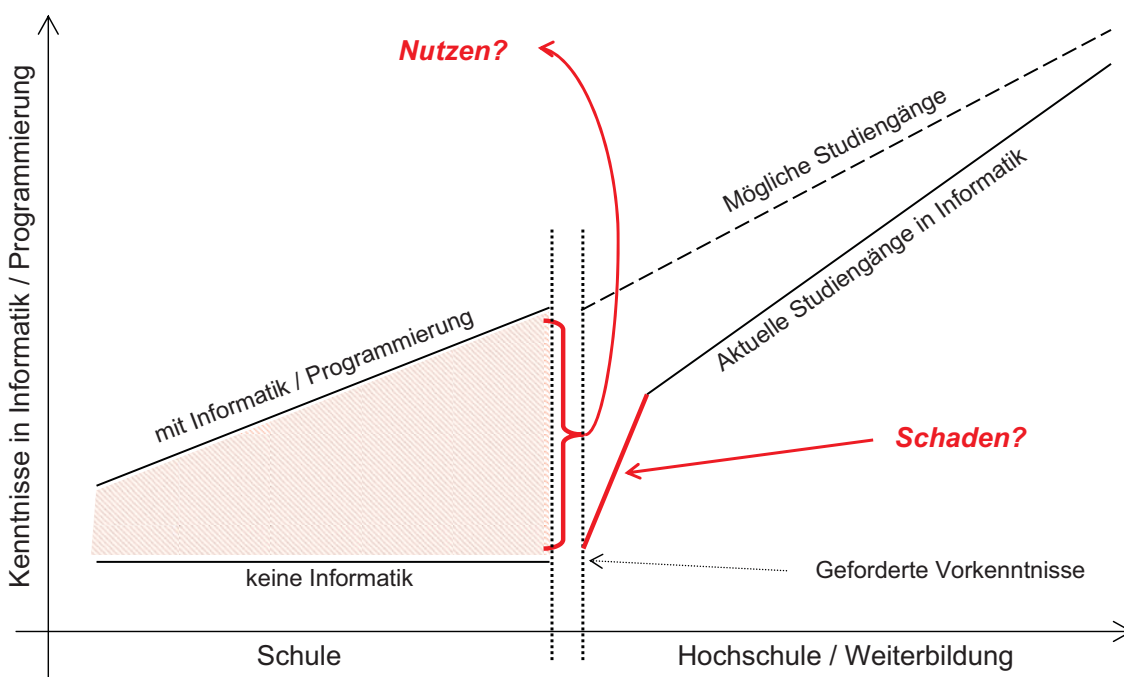
- ◆ Auf jeden Fall nicht einfacher ([PPiG'07])
- ◆ Schwieriger zu lernen/verstehen
 - Ja (mit Einschränkung)
- ◆ Schwieriger zu unterrichten
 - Davon bin ich überzeugt
- ◆ Sollte man deshalb warten
 - Nein, die Konzepte brauchen Zeit zum „reifen“

Die lange Antwort



- ◆ Gibt es überhaupt ein Problem? → **Ja**
- ◆ Ist die Objektorientierung schwieriger? → **Ja**
- ◆ Unsere Rahmenbedingungen
- ◆ Ursachenforschung
- ◆ Erfolgsfaktoren
- ◆ Zusammenfassung

Informatik in der Schule



Mögliche Zielgruppen



- A. Personen mit Vorkenntnissen** in der Informatik/
Programmierung sind unterfordert
- „Das haben wir doch schon in der Schule durchgenommen“
 - „Null Problemo“
 - Böses Erwachen nach der Klausur
- B. Personen ohne Vorkenntnisse** in der Informatik/
Programmierung sind überfordert
- „Viel zu schwierig, viel zu schnell“
 - „Das schaff ich nie“
 - Abschreckender Effekt
- **Situation wird keiner Zielgruppe gerecht**

Die lange Antwort



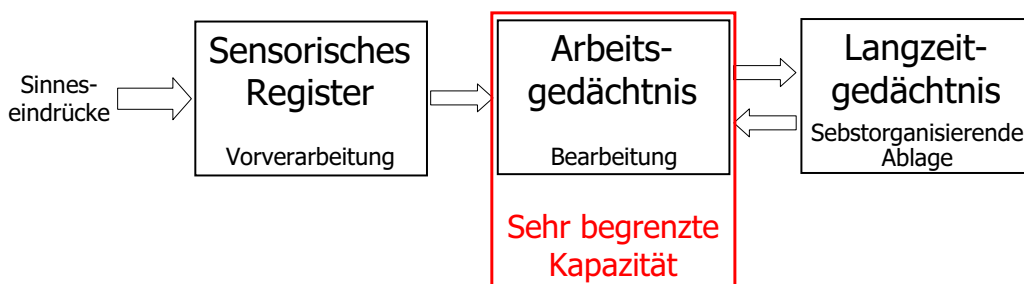
- ◆ Gibt es überhaupt ein Problem? → **Ja**
- ◆ Ist die Objektorientierung schwieriger? → **Ja**
- ◆ Unsere Rahmenbedingungen → **Schwierig**
- ◆ Ursachenforschung
- ◆ Erfolgsfaktoren
- ◆ Zusammenfassung

Die lange Antwort



- ◆ Gibt es überhaupt ein Problem? → **Ja**
- ◆ Ist die Objektorientierung schwieriger? → **Ja**
- ◆ Unsere Rahmenbedingungen → **Schwierig**
- ◆ Ursachenforschung
 - Kognitive Belastung
 - Programmierpläne
 - Sprachen und Werkzeuge
 - Terminologie
 - Beispiele
- ◆ Erfolgsfaktoren
- ◆ Zusammenfassung

Unser Gedächtnis



Chunking



- ◆ Chunk = Informationen, die als Einheit betrachtet wird
- Effektiveres Arbeitsgedächtnis

Hell oni cetom eety ou

+46907866735

Hello nice to meet you

+46 90 786 67 35

- Lernen ⇒ sehen und verarbeiten immer komplexerer Einheiten
- Schemata (im Langzeitgedächtnis)

Schemata

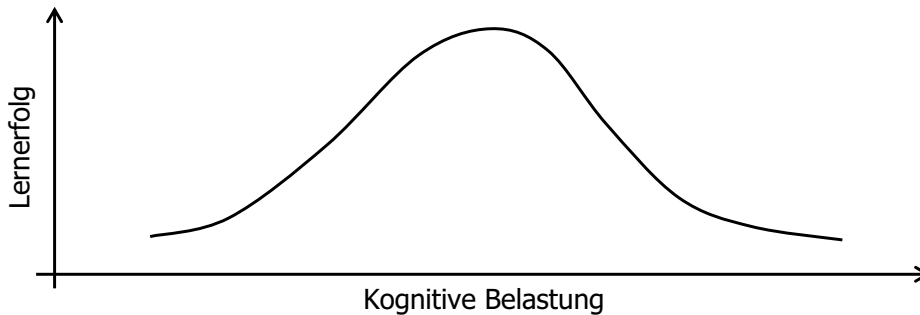


┌=1 □=2 L=3 ▢=4 □=5 □=6 7=7 ▢=8 Γ=9

7 ▢ □ □ ┌ ┌ 7 $\xrightarrow{\text{?}}$ 7 8 6 6 1 2 7

1	2	3
4	5	6
7	8	9

- Neue Information ist einfacher zu lernen, wenn sie „Sinn macht“



◆ Drei additive Faktoren

- Intrinsic: Problembedingt; schwierig zu beeinflussen
- Germaine: Bedingt durch Lernprozesse; gerne hoch
- Extraneous: Irrelevant fürs Lernen; minimieren

Kognitive Belastung und (OO)P

- ◆ Enge Vernetzung grundlegender Konzepte
 - Intrinsic KB ist hoch

- ◆ Unreife didaktische Konzepte
 - Extraneous KB ist hoch

- ◆ Keine Schemata im Langzeitgedächtnis auf die die Studierenden aufbauen können
 - Germaine KB ist niedrig

- **Das kann nicht gut gehen!?**

Ursachenforschung

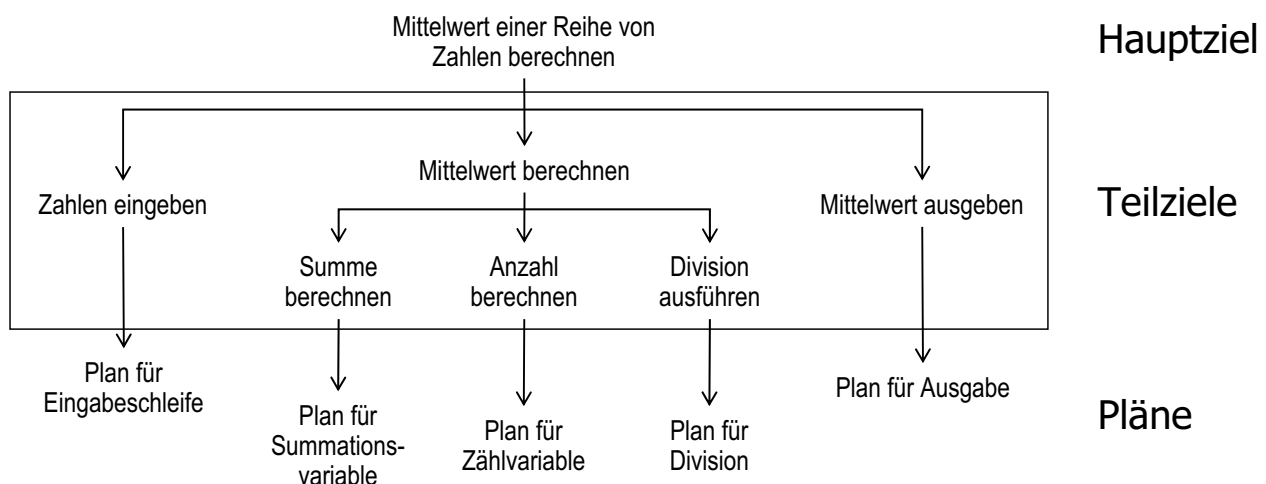


- ◆ Kognitive Belastung
- ◆ Programmierpläne
- ◆ Sprachen und Werkzeuge
- ◆ Terminologie
- ◆ Beispiele

Programmierpläne



- ◆ Programmieren ist zielgerichtetes instantiieren und kombinieren von Programmierplänen



Instantiierung und Kombination



var

summe, anzahl, aktuell: int;
mittelwert: real;

begin

Plan für Summationsvariable (Initialisierung) → summe := 0;
Plan für Zählvariable (Initialisierung) → anzahl := 0;

Plan für Eingabeschleife →

Plan für Summationsvariable (Update) → summe := summe + aktuell;
Plan für Zählvariable (Update) → anzahl := anzahl + 1

repeat

readln(aktuell);
if aktuell <> ABRUCH **then begin**

end

until aktuell = ABRUCH;

Plan für Division → mittelwert := summe / anzahl;

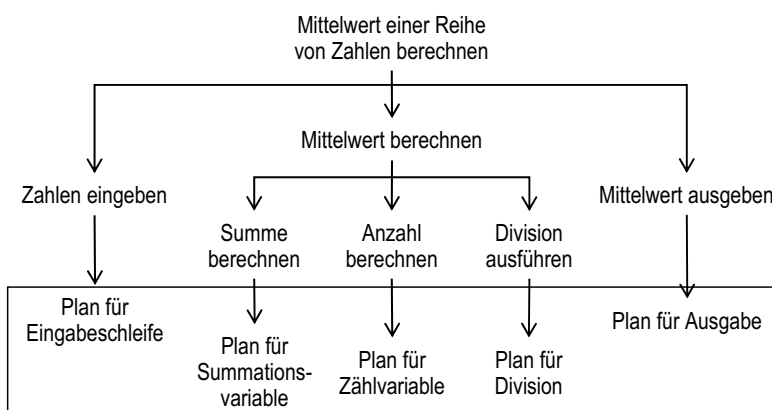
Plan für Ausgabe → writeln(mittelwert)

end

ExpertInnen



- ◆ Viele (abstrakte) Pläne
- ◆ Generelle Lösungsstrategien
- ◆ Einfaches Problem ⇒ „top-down forward expansion“



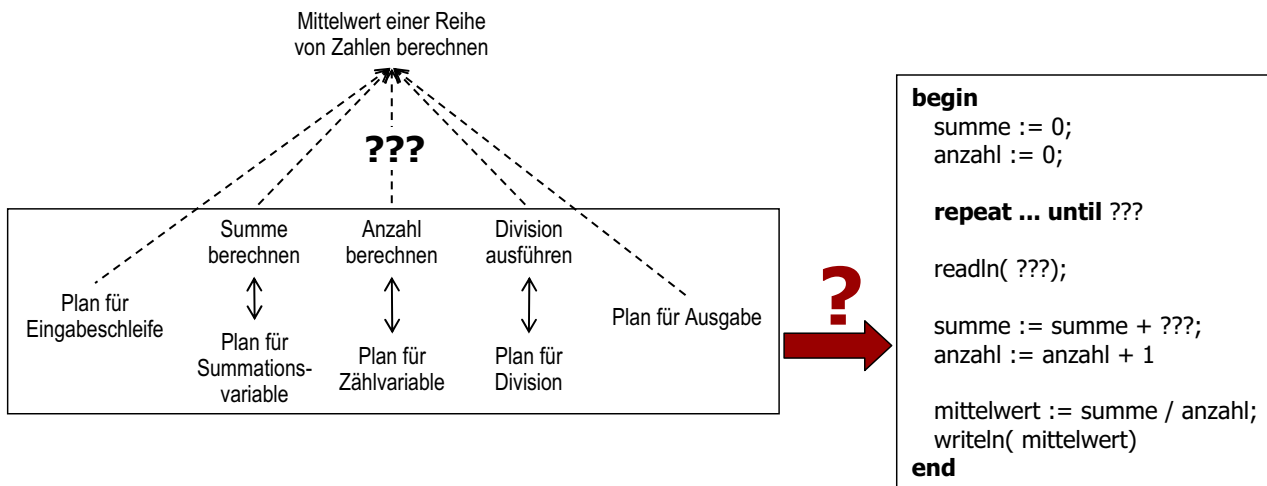
```

var
  summe, anzahl, aktuell: int;
  mittelwert: real;
begin
  summe := 0;
  anzahl := 0;
  repeat
    readln( aktuell);
    if aktuell <> ABRUCH then begin
      summe := summe + aktuell;
      anzahl := anzahl + 1
    end
    until aktuell = ABRUCH;
  mittelwert := summe / anzahl;
  writeln( mittelwert)
end
  
```

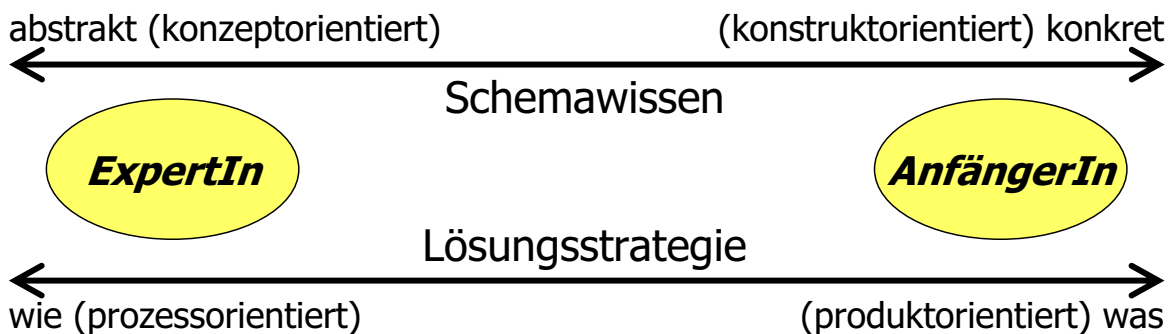
AnfängerInnen



- ◆ Wenig (konkrete) Pläne
- ◆ Keine generelle Lösungsstrategien
- ◆ Problem \Rightarrow „bottom-up backward solution“



Ein möglicher Konflikt



- ◆ Kognitive Belastung
- ◆ Programmierpläne
- ◆ Sprachen und Werkzeuge
- ◆ Terminologie
- ◆ Beispiele

- ◆ C++/Java die populärsten Sprachen (>75% [ICER'06])
- ◆ Probleme (im historischen Vergleich)
 - Viele neue Konzepte (nicht nur OO!)
 - Mehr konkrete Syntax
 - Verwirrende Syntax (z.B. „=" für Zuweisung)
 - Kompliziertere Ein-/Ausgabe
 - Umfangreiche Bibliotheken
 - ...
- ◆ Alternativen vorhanden (Pascal, Python, Scheme, ...)
- ◆ Transfer (prozedural → OO) mehrfach nachgewiesen

- ◆ Spezielle Programmierumgebungen (JGrasp, BlueJ, ...)
- ◆ Mikrowelten/Werkzeugkästen (Greenfoot, Alice, ...)
- ◆ „Kognitive“ Werkzeuge
 - Metaphern (Objekte = „lebende“ Dinge mit speziellen Verantwortlichkeiten)
 - Abstrakte Modelle (mit denen z.B. die Ausführung von Programmen beschrieben werden kann)
 - Syntaxdiagramme/(E)BNF
- ◆ Visualisierungen
 - Algorithmen
 - Abstrakte Maschinen

„A class is a set of objects that share a common structure and a common behaviour.“ [Booch, 91; S. 93]

- ◆ Kleine begriffliche Unterschiede \Rightarrow Große Wirkung
- ◆ Ist eine Fussballmannschaft eine Klasse?
- ◆ Brauche ich für jede neue Objektmenge eine neue Klasse?
- ◆ Schüler verwechseln verschiedene Begriffshierarchien (z.B. Partonomie vs. Taxonomie) [Teif/Hassan, 06]

Schlechte Angewohnheiten verbreiten sich schnell



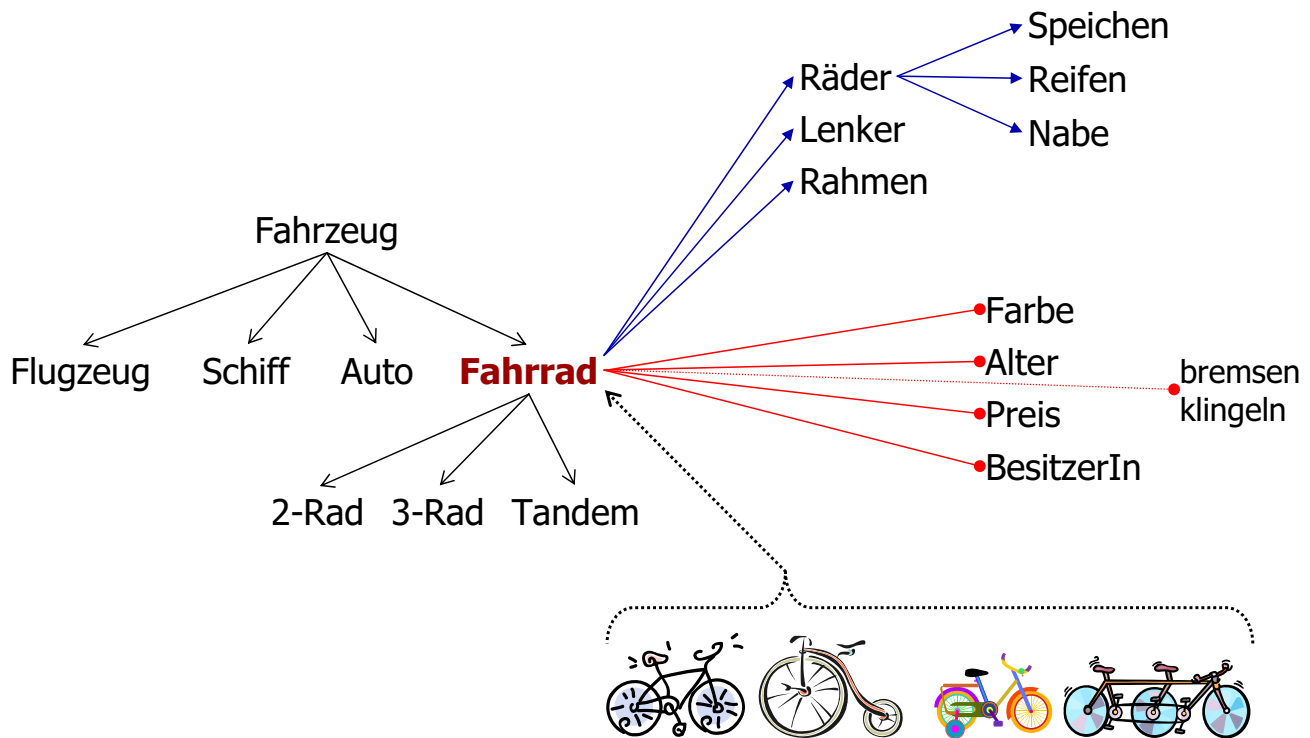
Google Suchbegriff	Anzahl Treffer nach Einschränkung auf <i>object oriented programming</i>	Anzahl Treffer nach nochmaliger Einschränkung auf <i>Java</i>
„a class is a set of objects“	32400	22900
„a class describes a set of objects“	638	413
„a class is a template“	877	610

Ein Dschungel von Begriffen



	UML	Java	C++	Weitere
Attribut	attribute	(field) variable	data member, (instance/reference) variable	
Methode	operation	method	member function	
Eigenschaft	feature	member	member	property
Methodenimplementierung	method	method definition / body	(member) function definition	method implementation
Subklasse	child	subclass	derived class / child class	
Superklasse	parent	superclass	base class	
Vererbung	generalization	inheritance	inheritance	specialization

Die konkrete Syntax liefert noch mehr Vielfalt, z.B. „extends“ in Java



Beispiele

◆ Eigenschaften „guter“ Beispiele

- Korrekt
- Verständlich
- Im Einklang mit den Lernzielen

◆ Beispiele haben Vorbildfunktion

◆ Oberflächliche und grundlegende Eigenschaften wichtig

→ Keine „Regelverletzungen“

Ein OO Beispielprogramm



```
public class Example2_9
{
    public static void main(String[ ] args)
    {
        System.out.println("(int) (7.9) = " + (int) (7.9));
        System.out.println("(int) (3.3) = " + (int) (3.3));
        System.out.println("(double) (25) = " + (double) (25));
        System.out.println("(double) (5 + 3) = " + (double) (5 + 3));
        System.out.println("(double) (15) / 2 = " + ((double) (15) / 2));
        System.out.println("(double) (15 / 2) = " + ((double) (15 / 2)));
        System.out.println("(int) (7.8 + (double) (15) / 2) = "
            + ((int) (7.8 + (double) (15) / 2)));
        System.out.println("(int) (7.8 + (double) (15 / 2)) = "
            + ((int) (7.8 + (double) (15 / 2))));
    }
}
```

Malik: Java Programming, 3. Auflage

INFOS 2007

Copyright © by jubo@cs.umu.se

53

Ein OO Beispielprogramm



```
public class Die
{
    private final int MAX = 6; // maximum face value

    private int faceValue; // current value showing on the die

    //-----
    // Constructor: Sets the initial face value.
    //-----
    public Die()
    {
        faceValue = 1;
    }

    //-----
    // Rolls the die and returns the result.
    //-----
    public int roll()
    {
        faceValue = (int)(Math.random() * MAX) + 1;

        return faceValue;
    }

    //-----
    // Face value mutator.
    //-----
    public void setFaceValue (int value)
    {
        faceValue = value;
    }

    //-----
    // Face value accessor.
    //-----
    public int getFaceValue()
    {
        return faceValue;
    }

    //-----
    // Returns a string representation of this die.
    //-----
    public String toString()
    {
        String result = Integer.toString(faceValue);

        return result;
    }
}
```

Lewis/Loftus: Java Software Solutions, 5. Auflage

INFOS 2007

Copyright © by jubo@cs.umu.se

54

Die lange Antwort



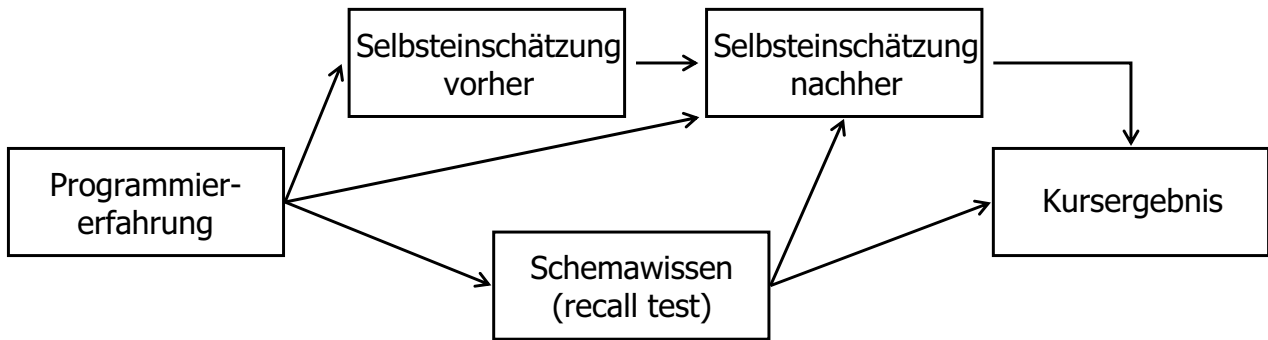
- ◆ Gibt es überhaupt ein Problem? → **Ja**
 - ◆ Ist die Objektorientierung schwieriger? → **Ja**
 - ◆ Unsere Rahmenbedingungen → **Schwierig**
 - ◆ Ursachenforschung
 - Kognitive Belastung
 - Programmierpläne
 - Sprachen und Werkzeuge
 - Terminologie
 - Beispiele
 - ◆ Erfolgsfaktoren
 - ◆ Zusammenfassung
- Noch viel zu tun**

Erfolgsfaktoren



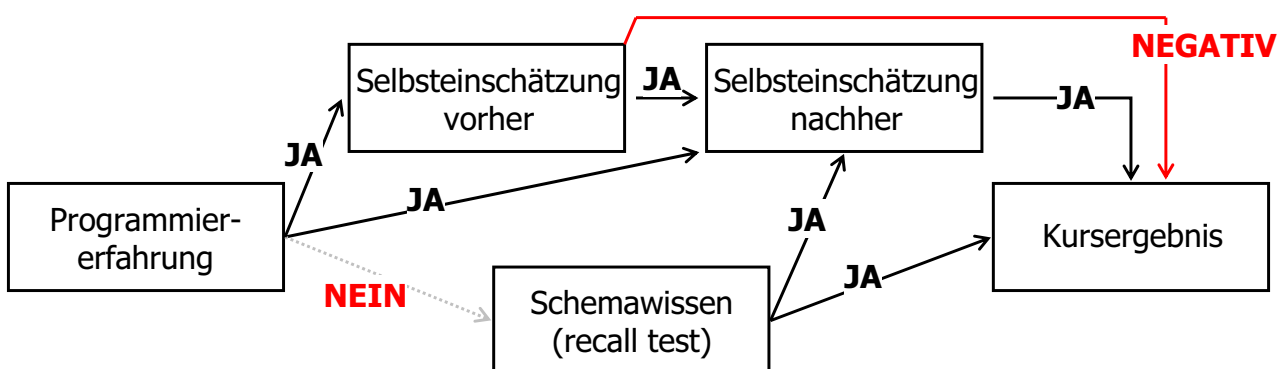
- ◆ Viele Studien aber wenig eindeutige Resultate
 - + Gute Kenntnisse in Mathematik (nicht unsere Daten)
 - + Vorkenntnisse im Programmieren (ja)
 - + Allgemein hoher Notendurchschnitt (ja)
 - + Positive Selbsteinschätzung
 - Intensives Computer-spielen
- ◆ Gegenseitige Beeinflussung von Faktoren kaum untersucht

Wiedenbeck et al. – Annahme



→ positive Wirkung

Wiedenbeck et al. – Resultat



→ positive Wirkung

- ◆ Gleiche Resultate für InformatikerInnen (75 Stud.) und Andere (120 Stud., keine technisch-naturw. Fächer)

Objektorientiertes Programmieren

Machen wir irgendwas falsch?



- ◆ Ja, aber wir verstehen immer noch nicht was
- ◆ Viele Forschungsgebiete führen einen Dornröschenschlaf
 - Psychologie des Programmierens
 - Program Comprehension
 - Kognitive Belastung
 - Worked Examples
 - Programmierpläne und konzeptuelle Modelle
 - ...
- ◆ Gute Fortschritte in den letzten paar Jahren
 - Wachsendes Interesse in Grundlagenforschung
 - Wachsende Anzahl „verwertbarer“ Publikationen



Packen wir's an

DANKE