

1

Einführung in die Programmierung

Bertrand Meyer

Letzte Überarbeitung 22. Januar 2005

Chair of Software Engineering Introduction to Programming - Lecture 26

2

Vorlesung 26:

Von Programmierung zu Software Engineering

Chair of Software Engineering Introduction to Programming - Lecture 26

3

Software Engineering (1)

- Die Prozesse, Methoden, Techniken, Werkzeuge und Sprachen für die Entwicklung von **Qualitätssoftware**.

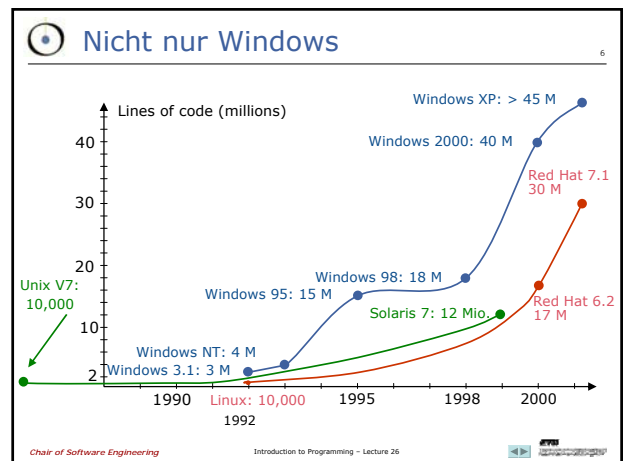
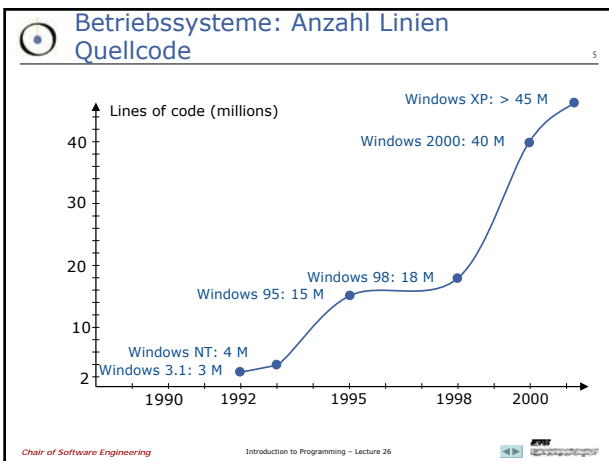
Chair of Software Engineering Introduction to Programming - Lecture 26

4

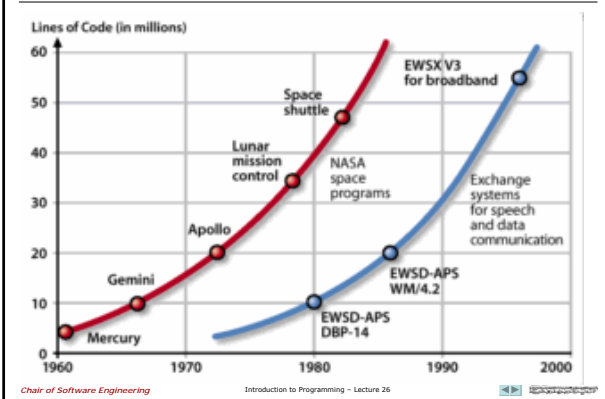
Software Engineering (2)

- Die Prozesse, Methoden, Techniken, Werkzeuge und Sprachen für die Entwicklung von **Qualitätssoftware**, die
 - den Umfang betreffend sehr gross ist
 - während einer langen Zeit entwickelt und benutzt wird
 - von vielen Entwicklern bearbeitet wird
 - viele Änderungen und Revisionen erfährt

Chair of Software Engineering Introduction to Programming - Lecture 26



Nicht nur Betriebssysteme



Das Hauptthema

Softwaresysteme entwickeln, die

- pünktlich und innerhalb des Budgets fertig gestellt werden
- von hoher Qualität sind
- eventuell umfangreich und komplex sind
- erweiterbar sind

US Software Industrie, 1998

Standish-Gruppe: "Chaos"-Bericht

250,000 Software Projekte, \$275 Milliarden

- Projektergebnisse:
 - 28% aufgegeben (1994: 31%)
 - 27% erfolgreich (1994: 16%)

Der Rest: "herausgefordert"

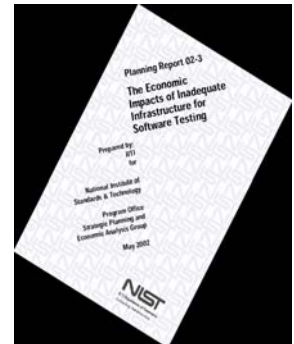
- Kleinere Projekte haben höhere Chancen auf Erfolg

NIST Bericht übers Testen (Mai 2002)

- Die finanziellen Konsequenzen einer "ungenügenden Test-Infrastruktur" für Entwickler und Benutzer

\$ 59.5 M.

(Finanzwesen \$3.3 Mia., Auto-, Luft- und Raumfahrt \$1.8 Mia. usw.)



Qualitätsfaktoren für Software

- **Externe:** interessant für Kunden
 - Beispiele: Zuverlässigkeit, Erweiterbarkeit
- **Interne:** interessant für Entwickler
 - Beispiele: Modularität, Stil

Einige interne Faktoren

- Modularität
- Beachtung von Stilregeln
- Konsistenz
- Struktur
- ...

Einige externe Faktoren

13

Produktqualität (unmittelbar):

- Zuverlässigkeit
- Effizienz
- Leichtigkeit der Benutzung
- Leichtigkeit des Erlernens

Prozessqualität:

- Pünktlichkeit
- Kosteneffektivität

Produktqualität (langfristig):

- Erweiterbarkeit
- Wiederverwendbarkeit
- Portabilität

Chair of Software Engineering Introduction to Programming - Lecture 26

Externe Faktoren: Zuverlässigkeit

14

Zuverlässigkeit = Korrektheit + Robustheit + Integrität

Chair of Software Engineering Introduction to Programming - Lecture 26

Zuverlässigkeit

15

- **Korrektheit**
Die Fähigkeit des Systems ihre Aufgaben gemäss der Spezifikation, für die Fälle die durch die Spezifikation abgedeckt werden, auszuführen
- **Robustheit**
Die Fähigkeit des Systems fehlerhafte Fälle sicher zu behandeln
- **Integrität**
Die Fähigkeit des Systems seine Benutzer, seine Daten und sich selbst gegenüber feindlicher Benutzung zu schützen

Chair of Software Engineering Introduction to Programming - Lecture 26

Externe Faktoren

16

Produktqualität (unmittelbar):

- Zuverlässigkeit
- Effizienz
- Leichtigkeit der Benutzung
- Leichtigkeit des Erlernens

Prozessqualität:

- Pünktlichkeit
- Kosteneffektivität

Produktqualität (langfristig):

- Erweiterbarkeit
- Wiederverwendbarkeit
- Portabilität

Chair of Software Engineering Introduction to Programming - Lecture 26

Aufgaben für die Entwicklung von Software

17

- Anforderungsanalyse
- Spezifikation
- Design
- Implementierung
- Validierung & Verifikation (V&V)
- Management
- Planung und Abschätzung
- Messung

Chair of Software Engineering Introduction to Programming - Lecture 26

Anforderungsanalyse

18

- Benutzerwünsche verstehen
- Beschränkungen des Systems verstehen
 - Interne Beschränkungen: Klassen-Invarianten
 - Externe Beschränkungen

Chair of Software Engineering Introduction to Programming - Lecture 26

Validierung & Verifikation

19

- Verifikation: Überprüfung, ob das System richtig konstruiert wurde
(alle Regeln befolgt wurden)
- Validierung: Überprüfung, ob das richtige System konstruiert wurde
(alle Benutzerwünsche zufrieden gestellt wurden)

Lebenszyklusmodelle von Software

20

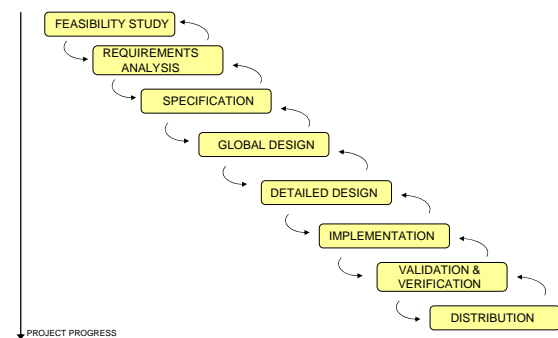
Beschreiben eine Grobverteilung der Aufgaben für Softwareentwicklung, und die Reihenfolge dieser Aufgaben

Es gibt zwei Arten von Modellen:

- Darstellung einer abstrakten Version der Realität
- Darstellung des idealen Schemas; wird in der Praxis dann nicht immer eingehalten

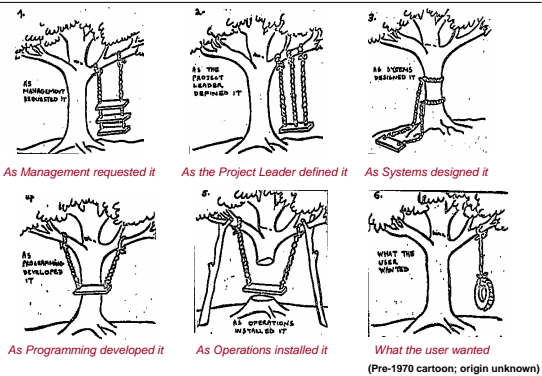
Das Wasserfallmodell (Royce, 1970)

21



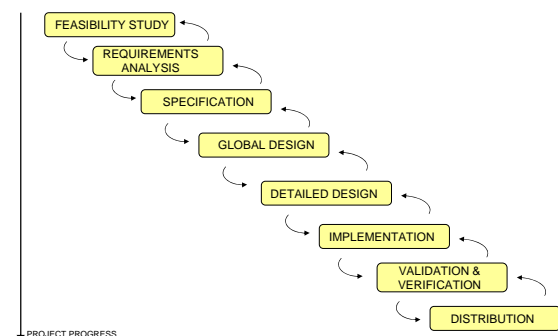
Lebenszyklus: was nicht passieren sollte

22



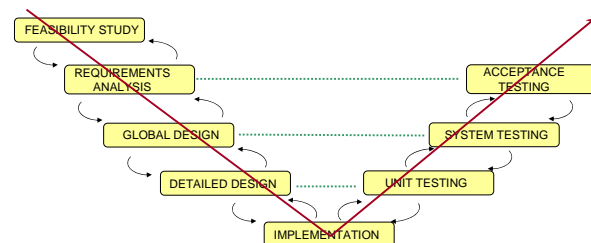
Das Wasserfallmodell

23



Eine realistischere Version

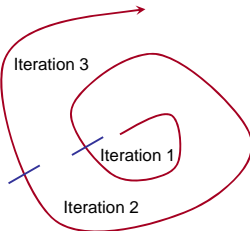
24



Das Spiralmodell

25

- Wende einen dem Wasserfallmodell ähnlichen Ansatz auf sukzessive Prototypen an



Das Problem mit Prototypen

26

- Softwareentwicklung ist schwierig wegen der Notwendigkeit der Einbindung von widersprüchlichen Kriterien, z.B. Portabilität und Effizienz
- Ein Prototyp opfert typischer Weise einige dieser Kriterien
- Risiko, dass der Prototyp an Kunden ausgeliefert wird

Nahtlose, inkrementelle Entwicklung

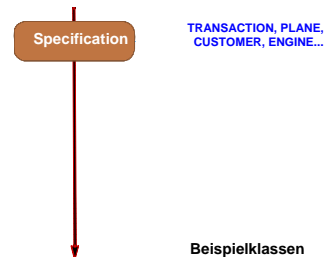
27

Die Eiffelsicht:

- Durchwegs eine einheitliche Menge von Notationen, Werkzeugen, Konzepten und Prinzipien
- Eiffel ist ebenso wertvoll für die Analyse und das Design wie für die Implementierung und Wartung
- Kontinuierliche, inkrementelle Entwicklung
- Modell, Implementierung und Dokumentation konsistent halten
- Umkehrbarkeit: vorwärts als wie auch rückwärts

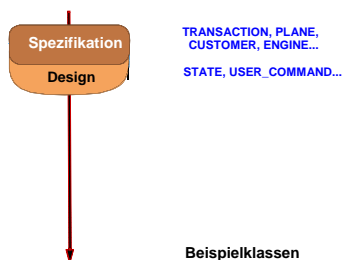
Nahtlose Entwicklung (1)

28



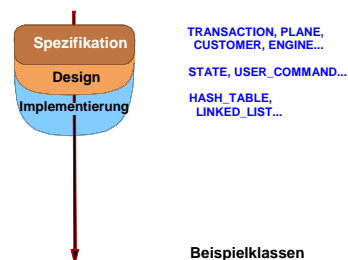
Nahtlose Entwicklung (2)

29



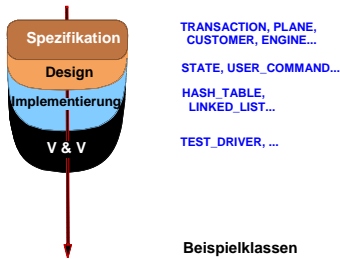
Nahtlose Entwicklung (3)

30



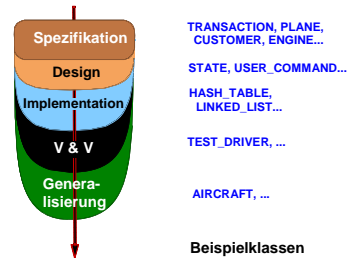
Nahtlose Entwicklung (4)

31



Nahtlose Entwicklung (5)

32



Generalisierung

33

- Für die Wiederverwendung vorbereiten
- Zum Beispiel:
 - Eingebaute Einschränkungen entfernen
 - Spezifische Abhängigkeiten vom Projekt entfernen
 - Dokumentation, Verträge verbessern...
 - Gemeinsamkeiten extrahieren und Vererbungshierarchie restaurieren
- Wenige Firmen haben den Mut das Budget für dies zur Verfügung zu stellen

Antoine de Saint-Exupéry

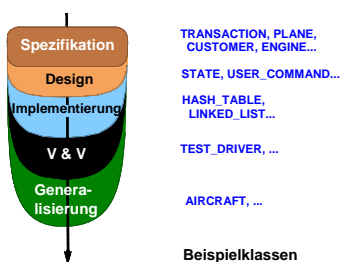
34

It seems that the sole purpose of the work of engineers, designers, and calculators in drawing offices and research institutes is to polish and smooth out, lighten this seam, balance that wing until it is no longer noticed, until it is no longer a wing attached to a fuselage, but a form fully unfolded, finally freed from the ore, a sort of mysteriously joined whole, and of the same quality as that of a poem. It seems that perfection is reached, not when there is nothing more to add, but when there is no longer anything to remove.

(Terre des Hommes, 1937)

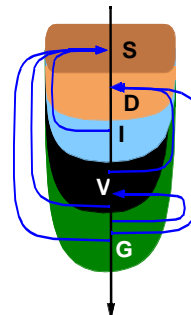
Nahtlose Entwicklung

35



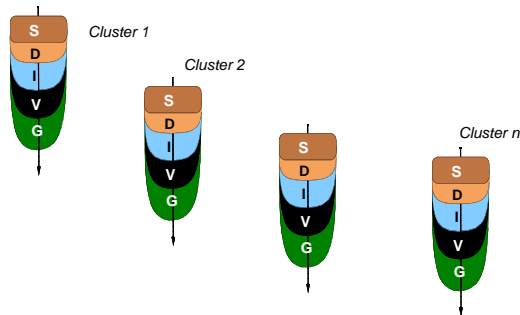
Umkehrbarkeit

36



Das Clustermodell

37



Chair of Software Engineering

Introduction to Programming - Lecture 26



"Agile Methods" und "Extreme Programming"

38

- Beim Prozess und Wiederverwendung zurückstecken
- Betone den Einfluss vom Testen auf den Entwicklungsablauf
- Betone die Rolle des "Refactoring"

Chair of Software Engineering

Introduction to Programming - Lecture 26



Validierung und Verifikation

39

- Nicht nur Testen:

Statische Analyse untersucht Code nach möglichen Mängeln, z.B. nicht initialisierten Variablen

- Sollte während des ganzen Prozesses angewendet werden, nicht nur am Ende

Chair of Software Engineering

Introduction to Programming - Lecture 26



Software Engineering Werkzeuge

40

- Entwicklungsumgebungen (Compiler, Browser, Debugger, ...): "IDE"
- Dokumentations-Werkzeuge
- Werkzeuge zur Aufstellung von Anforderungen
- Analyse- und Designwerkzeuge
- Konfiguration- und Versions-Verwaltung (CVS, Source Safe...) (auch "make" usw.)
- Formale Entwicklung und Beweiswerkzeuge
- Integrierte CASE- (Computerunterstützte Software Engineering) Umgebungen

Chair of Software Engineering

Introduction to Programming - Lecture 26



Konfigurations-Verwaltung

41

- Ziel: sicherstellen, dass die Versionen, die für die verschiedenen Komponenten eines Systems benutzt worden sind, kompatibel sind
- Eine der grössten Quellen von Software-Katastrophen, wenn dies schlecht gemacht wird
- Gute Werkzeuge existieren heutzutage, z.B. CVS, Source Safe
- Projekte, die keines dieser Werkzeuge benutzen, werden von Idioten geleitet

Chair of Software Engineering

Introduction to Programming - Lecture 26



Formale Methoden

42

- Benutze Mathematik als Basis für Softwareentwicklung
- Ein Softwaresystem wird als mathematische Theorie betrachtet, die stufenweise verfeinert wird, bis sie direkt implementierbar ist
- Jede Variante der Theorie und jede Verfeinerung wird **bewiesen**
- Beweis wird durch Computerwerkzeuge unterstützt
- Beispiele: *Atelier B*, Sicherheitssystem der neuesten Pariser U-Bahn

Chair of Software Engineering

Introduction to Programming - Lecture 26



Messungen

43

Sachen, die man messen kann:

- Produktattribute: Anzahl der Codezeilen, Anzahl von Klassen, Komplexität der Kontrollstrukturen ("cyclomatic number"), Komplexität und Tiefe der Vererbungshierarchie, Präsenz von Verträgen ...
- Projektattribute: Anzahl von Personen, Personenmonate, Kosten, Vervollständigungszeit, Zeit für verschiedene Aktivitäten (Analyse, Design, Implementierung, V&V usw.)

Kosten-Modelle

44

- Versuch, die Kosten der Softwareentwicklung vor der Projektrealisierung anhand der Schätzung von Parametern abzuschätzen
- Beispiel: COCOMO (Constructive Cost Model), Barry Boehm

L: 1000 * Delivered Source Instructions (KDSI)

| Program type | Effort (pm) | Time |
|--------------|------------------|-------------------|
| Application | $2.4 * L * 1.05$ | $2.5 * pm * 0.38$ |
| Utility | $3.0 * L * 1.12$ | $2.5 * pm * 0.35$ |
| System | $3.6 * L * 1.20$ | $2.5 * pm * 0.32$ |

Softwarezuverlässigkeitsmodelle

45

- Anzahl der Fehler durch
 - Eigenschaften des Programms
 - Anzahl der bisher gefundenen Fehler abschätzen
- Variante: "Fault Injektion"

Projekt-Management

46

- Teamspezialitäten: Kundenbeziehungen, Analytiker, Designer, Implementierung, Tester, Manager, Dokumentierung...
- Welche Rolle für den Manager: nur geschäftsführende oder auch technische?
- "Chef der Programmiererteams"

Software Engineering

47

- Am Ende ist es Code
- Unterschätze nicht die Rolle der Werkzeuge, Sprachen und – allgemeiner – der Technologie
- Schlechtes Management vernichtet Projekte
Gute Technologie macht Projekte erfolgreich

Programmiersprachen

48

- Nicht nur um mit deinem Computer zu sprechen!
- Eine Programmiersprache ist ein Werkzeug zum Denken

Ein bisschen Geschichte

49

- "Plankalkül", Konrad Zuse, 1940s
- Fortran (FORmula TRANslator), John Backus, IBM, 1954
- Algol, 1958/1960

Etwas FORTRAN Code

50

```
100 IF (N) 150, 160, 170
150 A(I) = A(I)**2
    READ ("I6") N
    GOTO 100
C   THE NEXT ONE IS THE TOUGH CASE
160 A(I)=A(I)+1
    READ ("I6") N
    GOTO 100
170 STOP
    END
```

Algol

51

- Internationaler Ausschuss, Europäer und Amerikaner; führte zur IFIP. Algol 58, Algol 60.
- Beeinflusst durch (und Reaktion auf) FORTRAN; auch beeinflusst durch LISP (siehe später). Rekursive Prozeduren, dynamische Arrays, Blockstrukturen, dynamisch zugewiesene Variablen
- Neuer Sprachbeschreibungsmechanismus: BNF (für Algol 60).

Algol W und Pascal

52

- Nachfolger von Algol 60, entwickelt von Niklaus Wirth (ETH)
- Algol W hat Datensatzstrukturen eingeführt
- Pascal betont Einfachheit, Datenstrukturen (Records, Zeiger).
- Keine umfangreiche Sprache, angepasst für die Lehre
- Hat geholfen die PC Revolution durch Turbo Pascal von Borland (Philippe Kahn) auszulösen

C

53

- 1968: Brian Kernighan und Dennis Richie, AT&T Bell Labs
- Anfänglich eng verbunden mit Unix
- Betonung auf "low-level" Maschinenzugriff: Zeiger, Adressarithmetik, Konversionen
- Rasend übernommen durch die Industrie in den 80er und 90er Jahren

Lisp und funktionale Sprachen

54

- LISP Processing, 1959, John McCarthy, MIT dann Stanford
- Fundamentaler Mechanismus ist die rekursive Funktionsdefinition
- Zahlreiche Nachfolger, z.B. Scheme (MIT)
- Funktionale Sprachen: Haskell, Scheme

LISP "Listen"

55

Eine Liste ist von der Form $(x_1 x_2 \dots)$ wobei jedes x_i eine der folgenden Sachen ist

- Ein Atom (Zahl, Bezeichner usw.)
- (Rekursiv) eine Liste:

Beispiele:

- $()$
- $(x_1 x_2)$
- $(x_1 (x_2 x_3) x_4 (x_5 (x_6 () x_7)))$

$((x_1 x_2))$ ist nicht dasselbe wie $(x_1 (x_2))$

Chair of Software Engineering

Introduction to Programming - Lecture 26



LISP Funktions-Anwendung und Definition

56

Die Anwendung der Funktion f auf die Argumente a, b, c wird geschrieben als $(f a b c)$

Beispiel Funktionsdefinition (Scheme):

```
(define (factorial n)
  (if (eq? n 0)
      1
      (* n (factorial (- n 1)))))
```

Chair of Software Engineering

Introduction to Programming - Lecture 26



Grundfunktionen

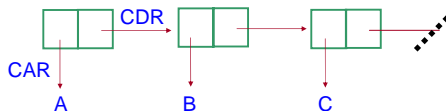
57

Let $my_list = (A B C)$

$(CAR my_list) = A$

$(CDR my_list) = (B C)$

$(CONS A (B C)) = (A B C)$



Chair of Software Engineering

Introduction to Programming - Lecture 26



Funktionen, die mit Listen arbeiten

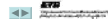
58

```
(define double-all (list)
  (mapcar
   '(lambda (x) (* 2 x)) list))
```

```
(define (mapcar function f)
  (if (null? ls) '()
      (cons
       (function (car ls))
       (mapcar function (cdr ls)))))
```

Chair of Software Engineering

Introduction to Programming - Lecture 26



Objekt-orientierte Programmierung

59

- Simula 67: Algol 60 Erweiterungen für Simulationen, Universität von Oslo, 1967 (nach Simula 1, 1964). Kristen Nygaard, Ole Johan Dahl
- Wuchs zur einer vollständigen Programmiersprache heran
- Smalltalk (Xerox PARC) wurde mit Ideen von Lisp und innovativen Ideen für Benutzerschnittstellen erweitert. Alan Kay, Adele Goldberg, Daniel Bobrow

Chair of Software Engineering

Introduction to Programming - Lecture 26



"Hybride" Sprachen

60

- Objective-C, um 1984: Smalltalk-Schicht oberhalb von C
- C++, um 1985: "C mit Klassen"

O-O wurde durch die Industrie akzeptiert

Schlüsselmoment: erste OOPSLA, 1986

Chair of Software Engineering

Introduction to Programming - Lecture 26



Java und C#

61

- C++ mit genügend Einschränkungen um Typsicherheit und Speicherbereinigung zu erlauben
- Java wurde ursprünglich für Applets in Zusammenhang mit dem Aufstieg des Internets 1995 vermarktet.
- C# fügt "delegates" (ähnlicher Mechanismus wie „Agents“ in Eiffel) hinzu

Eiffel

62

- Erste Version geht zurück auf Mitte-80er, zuerst an der OOPSLA 86 demonstriert
- Betonung auf Software Engineering Prinzipien: Informationskapselung, Design by Contract, statische Typisierung (durch Genericity), vollständige Anwendung von O-O Prinzipien
- Hauptanwendungsgebiet sind unternehmenskritische Anwendungen



63

Ende der Vorlesung 26