

Einführung in die Informatik

Vermitteln der Grundlagen für weitere Vorlesungen  
im Grund- und Hauptstudium

Anregungen für das Selbststudium / Anregungen fürs  
Programmieren

Konkrete Vorbereitung auf die Programmierung

## Allgemeines Phänomen in der Softwareentwicklung

- Trotz der substanziellen Abhängigkeit von Unternehmen /  
Branchen / Volkswirtschaften von funktionierenden  
Computersystemen (inkl. Anwendungen):
  - Keine zuverlässige Herstellung von Software im  
industriellen Maßstab
  - Kosten- und Terminüberschreitungen
  - Bei Auslieferung: ungenügende Softwarereife
  - Keine Produktivitätskontrolle wie in anderen industriellen  
Fertigungsbereichen
  - Keine Qualitätskontrolle wie in anderen industriellen  
Fertigungsbereichen

## Ursachen der Phänomene

- Thin Spread of Application Domain Knowledge
  - Projekte in neuer Branche
  - Technologiezentriertheit
- Fluctuating and Conflicting Requirements
  - Markt
  - verschiedene Kunden
  - Erkenntnisprozesse
  - Missverständnisse
- Communication and Coordination Breakdowns
  - unterschiedliche Vorstellungen und Zielsetzungen
  - Inkongruenzen zwischen Kompetenz und Verantwortung
  - Kapitulation

B. Curtis, H. Krasner, N. Iscoe, A Field Study of the Software Design Process for Large Systems, Communications of the ACM, November 1988, Vol. 31, No. 11, pp 1268 - 1287

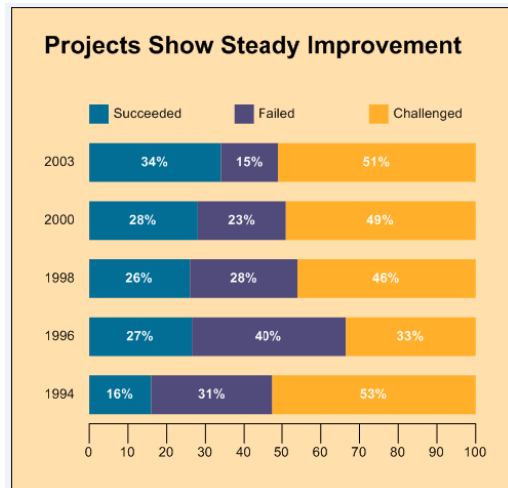
## Phänomene der aktuellen, industriellen Softwareentwicklung im einzelnen

### I Abbruchrate

Anzahl Function Points	Früher als geplant	Termingerecht	Verspätet	Abgebrochen
1 FP	14,86 %	83,16 %	1,92 %	0,25 %
10 FP	11,08 %	81,25 %	5,67 %	2,00 %
100 FP	6,06 %	74,77 %	11,83 %	7,33 %
1.000 FP	1,24 %	60,76 %	17,67 %	20,33 %
10.000 FP	0,14 %	28,03 %	23,83 %	48,00 %
100.000 FP	0,00 %	13,67 %	21,33 %	65,00 %
<b>Durchschnitt</b>	5,53 %	56,94 %	13,71 %	23,82 %

Abbruchrate großer Projekte nach C. Jones, American Programmer, April 1996

# Phänomene der aktuellen, industriellen Softwareentwicklung im einzelnen



The Standish Group: *Latest Standish Group CHAOS Report Shows Project Success Rates Have Improved by 50%*. <http://www.standishgroup.com/press/article.php?id=2>.

# Phänomene der aktuellen, industriellen Softwareentwicklung im einzelnen

## II Termine, Kosten, Qualität



T. de Marco, *Controlling Software Projects*, Yourdon Press, 1982

T. de Marco, *Why does software cost so much*, Dorset House Publishing, 1995

## Was ist Informatik?

engl: *Computer Science* (manchmal: Computing Science)

### Informatik-Duden:

Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von Digitalrechnern

### Studien- und Forschungsführer Informatik:

Wissenschaft, Technik und Anwendung der maschinellen Verarbeitung und Übermittlung von Informationen

### Brockhaus:

Wissenschaft, die sich mit der grundsätzlichen Verfahrensweise der Informationsverarbeitung und allg. Methoden der Anwendung solcher Verfahren befasst

## Was ist eine Wissenschaft?

- Eine Wissenschaft ist bestimmt durch ihren Gegenstand und ihre Methode. Wissenschaft ist jede *intersubjektiv* überprüfbare Untersuchung von Tatbeständen und die auf ihr beruhende, *systematische Beschreibung* und *Erklärung*.

- Informatik als Wissenschaft

Gegenstand: Informationsverarbeitung i.a., speziell

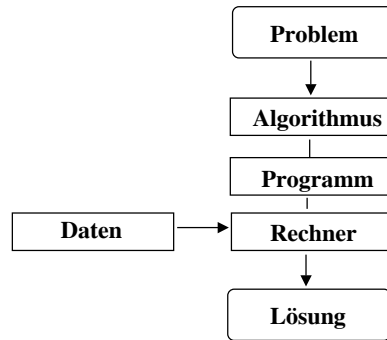
- Algorithmen
- Softwaretechnologie
- Anwendungen
- Hardware

Methode: angemessene Repräsentation und Formalisierung von *Information* und ihrer Verarbeitung

## Informationsverarbeitung und Problemlösen mit dem Computer

UNIVERSITÄT LEIPZIG

LPZ  E-BUSINESS  
applied telematics



DigInf 05/06

9

## Vorgeschichte der Informatik

UNIVERSITÄT LEIPZIG

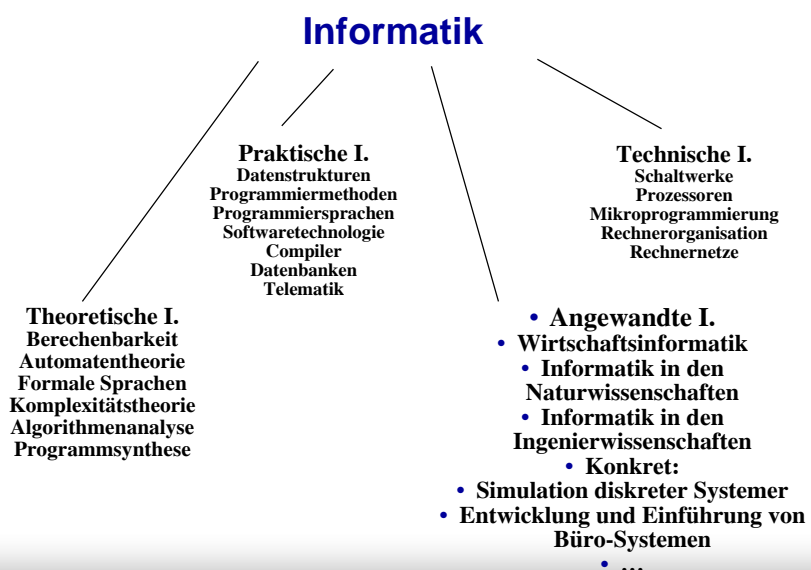
LPZ  E-BUSINESS  
applied telematics

- 700 • Abakus
- 300 • Euklidischer Algorithmus
- 500 • Erfindung Dezimalsystem in Indien
- 1202 • Leonardo v. Pisa (Fibonacci) verfaßt Einführung in dezimales Rechnen
- 1500 • Dezimalsystem mit arabischen Ziffern in Spanien
- 1524 • Adam Riese veröffentlicht Rechenbuch im Dezimalsystem
- 1615 • Nepersche Rechenstäbe zum Multiplizieren
- 1623 • Rechenmaschine von Schickard
- 1641 • Addiermaschine mit 8 Rechenstellen von Pascal
- 1674 • Rechenmaschine für 4 Grundrechenarten von Leibniz
- 1774 • Hahns erste zuverlässig arbeitende mechanische Rechenmaschine
- 1805 • Jacquards Webstuhl mit fester Programmsteuerung
- 1823 • Babbage entwickelt difference engine, Programmsteuerungsprinzip
- 1886 • Erfindung der Lochkarte durch Hollerith

DigInf 05/06

10

- 1934 • Zuse plant programmierbaren Relaisrechner auf Basis des Dualsystems
- 1941 • Zuse entwickelt elektromechanischen programmgesteuerten Rechner Z3
- 1944 • Aiken baut Mark I, sequentieller Rechner, Multiplikation: 6 s
- 1946 • Mauchly/Eckert entwickeln ENIAC, Röhrentechnologie
- 1949 • Edsac von Wilkes, erster Rechner mit gespeichertem Programm
- 1954 • Rechner der 1. Generation (IBM 650, USA)
- 1957 • Fortran (formula translator), 2. Generation, Transistoren
- 1960 • COBOL, ALGOL
- 1962 • BASIC
- 1971 • 3. Generation, integrierte Schaltungen, Mikroprozessoren, PASCAL
- 1973 • Programmiersprache C
- 1980 • Modula, Ada, PCs, Rechnernetze, Workstations
- 1986 • Laptops, Parallel-Rechner
- 1990 • optische Platten, Internet, World Wide Web



Mathematisch-logische Orientierung

Ingenieurwissenschaftliche Orientierung

Evolutionäre Orientierung

Partizipative Orientierung

Mathematisch-logische Orientierung (Dijkstra)

Computer können nur Symbole manipulieren.  
Programme sind maschinell ausführbare Formeln. Ziel  
der Informatik ist es also, korrekte Formeln in Form  
von Programmen zu schreiben und mit diesen  
Anwendungen zu lösen.

Ingenieurwissenschaftliche Orientierung (Parnas)

Informatiker planen, entwerfen und konstruieren eine  
Menge von in der Regel aufeinander folgenden  
Artefakten. Dabei sollten sie wie andere Ingenieure  
vorgehen.

### Evolutionäre Orientierung (Brooks, Lehman)

Anforderungen an Software ändern sich rasant. Die wahren Anforderungen sind oft erst klar, nachdem die Anwender mit einer Version der Software konfrontiert worden sind. Kontinuierliche Überarbeitung ist deshalb das einzig Erfolg versprechende Motto.

### Partizipative Orientierung (Coy, Floyd)

Software muss sich in den Arbeitsablauf / den Anwendungskontext von Menschen einfügen. Die Interaktion mit allen Beteiligten ist deshalb das wichtigste Hilfsmittel, um zu nützlicher Software zu kommen.

### Computer

... technische Geräte, die umfangreiche Informationen mit hoher Zuverlässigkeit und großer Geschwindigkeit automatisch verarbeiten und aufbereiten können.

... parametrisierbar mit Arbeitsanweisungen, die das Verhalten festlegen

### Automaten

... können eine bestimmte Aufgabe erfüllen, indem sie in einer bestimmten Weise bedient werden und dann eine bestimmte Folge von Aktionen durchführen, z.B. Fahrkarten-, Kaffee-, Zigarettenautomaten

Wichtige Unterscheidung ist offensichtlich die Parametrisierung mit einer Arbeitsanweisung, auch als Algorithmus bezeichnet.

Der Begriff des Algorithmus geht zurück auf den arabischen Schriftsteller Abu Dshafar Muhammed Ibn Musa al-Khwarizmi (um 825 n. Chr. in Khiva, heute Usbekistan), der beschrieb, wie Eigentum auf mehrere erbende Ehefrauen und Kinder verschiedenen Standes vererbt wurde.

Algorithmus:

- präzise, endliche Verarbeitungsvorschrift (unter Verwendung von Elementaroperationen),
- Anzahl der Elementaroperationen ist beschränkt,
- so formuliert, dass die in der Vorschrift notierten Elementaroperationen von einer mechanisch oder elektronisch arbeitenden Maschine durchgeführt werden können,
- jede Elementaroperation wird auf der Maschine in endlicher Zeit durchgeführt,
- die Reihenfolge der durchzuführenden Elementaroperationen muss festgelegt sein (hierbei sind Wahlmöglichkeiten zugelassen).

Beispiele für Algorithmen:

- Regeln für das Addieren, Multiplizieren, Berechnen der Bundesliga-Tabelle nach einem Spieltag
- Euklidischer Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen
- Regeln für das Berechnen von Zinsen und Zinseszinsen
- Regeln für die doppelte Buchführung

Hart an der Grenze: Algorithmus oder nicht?

- Kochrezepte
  - Erhitze Öl in der Pfanne bis sich Bläschen bilden.
  - Schlage die Eier in die Pfanne.
  - Füge eine Prise Salz hinzu und rühre gut durch.
  - Lasse alles brutzeln bis die Eier die richtige Konsistenz haben.
- Bastelanleitungen
- Bedienung von Fahrkartenautomaten

## Euklidischer Algorithmus

Gegeben: Zwei positive ganze Zahlen  $n$  und  $m$

Gesucht: größter gemeinsamer Teiler von  $n$  und  $m$

1. Wenn  $m$  kleiner als  $n$ , vertausche Werte von  $m$  und  $n$ .
2. Dividiere  $m$  durch  $n$ , nenne den Rest  $r$ .
3. Wenn  $r$  gleich 0 ist, so gib  $n$  aus und terminiere.
4. Wenn  $r$  nicht gleich 0 ist, so weise  $m$  den Wert von  $n$  zu und  $n$  den von  $r$ .
5. Gehe zu 2.

Beispielanwendung:  $n=15$ ,  $m=27$

## Algorithmus

Ein Algorithmus beschreibt also eine Funktion

$$f: E \rightarrow A$$

von der Menge der zulässigen Eingabedaten  $E$  in die Menge der Ausgabedaten  $A$ .

Hinweise:

- Nicht jede Funktion lässt sich durch einen Algorithmus berechnen!
- Präziser: nur wenige Funktionen lassen sich durch Algorithmen berechnen!

## Charakteristische Eigenschaften von Algorithmen

**Abstrahierung:** Ein Algorithmus löst eine Klasse von Problemen. Das konkret zu lösende Problem wird durch Parameter ausgewählt.

**Finithheit:** Die Beschreibung eines Algorithmus ist zu jedem Zeitpunkt endlich.

**Terminierung:** Algorithmen, die für jede Eingabe nach endlich vielen Schritten ein Resultat liefern und anhalten, heißen terminierend. Es gibt aber auch interessante, nicht-terminierende Algorithmen (Steuerung von Anlagen etc.).

## Charakteristische Eigenschaften von Algorithmen

**Determinismus:** Ein Algorithmus heißt deterministisch, wenn zu jedem Zeitpunkt seiner Ausführung höchstens eine Möglichkeit der Fortsetzung besteht.

**Determiniertheit:** Ein Algorithmus heißt determiniert, wenn er bei gleichen Eingaben und gleichem Zustand die gleiche Ausgabe liefert.

Hinweis: Ein terminierender, deterministischer Algorithmus ist immer determiniert.

## Nützlichkeit von Algorithmen

Unter den vielen, äquivalenten Algorithmen zur Lösung eines Problems müssen die passenden ausgesucht werden. "Passend" hängt dabei von einer Reihe nicht-funktionaler Kriterien ab:

- Korrektheit / Zuverlässigkeit / Robustheit
- Laufzeit und Speicherbedarf (Performanz)
- Benutzungsfreundlichkeit
- Wartbarkeit
- Wiederverwendbarkeit
- Portierbarkeit
- Interoperabilität

## Korrektheit / Zuverlässigkeit

- Korrektheit:
  - Übereinstimmung eines Programms mit seiner Spezifikation
    - ohne Spezifikation ist die Korrektheitsfrage nicht entscheidbar
    - bei informaler Spezifikation ist die Korrektheitsfrage nicht eindeutig entscheidbar
- Zuverlässigkeit:
  - dauerhafte Einsetzbarkeit, der Anwender kann sich erlauben, von der Software abzuhängen
    - Zuverlässigkeit ist ein relatives Kriterium, das vom Einsatzzweck der Software und vom Schadenspotential der Nichtverfügbarkeit der Software abhängt.
    - Zuverlässigkeit von Software wird bei neuer Software in aller Regel nicht erwartet (Bananen-Software). Wesentlicher Unterschied zu fast allen anderen industriellen Produkten.

- Zusammenhang zwischen Korrektheit und Zuverlässigkeit

	zuverlässig	nicht zuverlässig
korrekt	alle Anforderungen spezifiziert und erfüllt	unspezifizierte Anforderungen
nicht korrekt	nicht korrekte Fälle treten nicht auf (Überspezifikation)	Fehler verursachen Fehlverhalten

- Toleranz gegenüber nicht spezifizierter Bedienung / nicht spezifizierten Rahmenbedingungen
  - auch nicht robuste Software kann korrekt sein
  - nicht robuste Software kann leicht nutzlos werden
  - wesentliche Robustheitsanforderungen sollten deshalb spezifiziert werden

- Erfüllung der Anforderungen an Antwortzeitverhalten, Ressourcenbedarf
  - prinzipielle Überprüfungsmethoden:
    - Messen
    - Berechnen
    - Simulieren
  - häufiger Umgang mit der Eigenschaft Performanz:
    - Erstellen einer initialen Version
    - Verbesserung zum Zweck des Erreichens der notwendigen Performanz
    - Problem: deutliche Verbesserungen erfordern zuweilen grundlegendes Redesign

- Software ist benutzungsfreundlich, wenn die Anwender sie für einfach benutzbar halten
  - Gestaltung der Dialoge
  - einfache Konfigurierbarkeit
  - einleuchtender Aufbau (so weit sichtbar)
- alles andere als das Empfinden der Anwender zählt nicht!
- Häufiger Ansatz: Standardisierung der Benutzungsoberflächen

## Benutzungsfreundlichkeit (Usability)

UNIVERSITÄT LEIPZIG



- Usability eines Produktes ist das Ausmaß, in dem es von einem **bestimmten Benutzer** verwendet werden kann, um **bestimmte Ziele** in einem **bestimmten Kontext** **effektiv, effizient und zufriedenstellend** zu erreichen. (ISO 9244-11)

DigInf 05/06

31

## Benutzungsfreundlichkeit (Usability)

UNIVERSITÄT LEIPZIG



- **Erlernbarkeit** (Learnability)
    - wie schnell, wie einfach produktive Arbeit mit neuem System & wie einfach Erinnerung an Art und Weise der Nutzung
  - **Effizienz** (Efficiency of use )
    - Anzahl der Aufgaben/Zeiteinheit, die ein Nutzer ausführen kann
    - Beinhaltet Effektivität (akkurat und komplett Ziele erreichen) und Effizienz (durch auf dem Weg zum Ziel verbrauchte Ressourcen) aus ISO Definition
  - **Reliabilität**, **Verlässlichkeit** (Reliability in use)
    - Fehlerrate (Nutzerfehler, nicht Systemfehler) bei der Benutzung & Dauer bis zur Wiederherstellung nach einem Fehler
  - **Zufriedenheit** (Satisfaction)
    - subjektive Meinung der Systemnutzer, Komfort und Akzeptanz
- Ableitbar davon sind Anforderungen, die in GUI und System umgesetzt werden

DigInf 05/06

32

- **Wartbarkeit:**
  - leichte Handhabbarkeit einer Software nach ihrer Auslieferung
- **Arten der Wartung:**
  - korrektive Wartung: Beseitigung von Fehlern
  - adaptive Wartung: nachträgliche Anpassung an neue Anforderungen, z.B. gesetzliche Änderungen, neue Organisation
  - perfektive Wartung: Verbesserung im Hinblick auf nicht-funktionale Anforderungen, z.B. Performanz, Ergonomie
- **Wartbarkeit hängt stark von Strukturierung ab**
- **Wartbarkeit nimmt zumeist mit der Lebensdauer der Software ab**
  - Lehmans Law of Increasing Software Entropy: die Struktur der Software nimmt mit zunehmender Lebensdauer ab.

- **Wiederverwendbarkeit:**
  - Wahrscheinlichkeit, mit der Software in einem anderen Kontext wiederverwendet werden kann (oft bezogen auf Komponenten)
- **Wiederverwendbarkeit entsteht nicht zufällig, sondern muss geplant sein (infrastrukturell unterstützt werden (z.B. durch Rolle „Wiederverwender“))**
- **Beispiele: parametrisierte Datenstrukturen, Klassenbibliotheken**

## Portierbarkeit / Plattformunabhängigkeit

UNIVERSITÄT LEIPZIG



- Portierbarkeit
  - die Portierbarkeit einer Software ergibt sich aus dem Aufwand, der nötig ist, um eine Software auf einer anderen Plattform (DBMS, BS, UIMS) lauffähig zu machen (im Verhältnis zu ihrem Entwicklungsaufwand)
- hoher Grad an Portierbarkeit durch Verkapselung von Plattformabhängigkeiten, vgl. Ansätze der modellgetriebenen Softwareentwicklung

DigInf 05/06

35

## Interoperabilität

UNIVERSITÄT LEIPZIG



- Interoperabilität:
  - eine Software ist interoperabel, wenn sie sich mit geringem Aufwand mit anderer Software integrieren lässt
- PC-Werkzeuge sind mittlerweile größtenteils interoperabel
- hochintegrierte Software-Systeme sind meist weniger interoperabel, weil sie „alles“ können
  - Beispiel: integrierte Entwicklungsumgebungen, Workflow-Management-Systeme und Datenmodellierung
- Interoperabilität wird immer mehr zu einem Muss, weil kaum noch Software in völlig neuen Gebieten eingesetzt wird
  - Beispiel: Interoperabilität mit Office-Werkzeugen, Großrechner-Software, Datenmodellierungswerkzeugen

DigInf 05/06

36

## Darstellung von Algorithmen

- Natürliche Sprache
  - für die Kommunikation von Ideen oft ausreichend, muss präzisierbar sein
- Stilisierte Prosa und Pseudo-Code
  - Mischung von Prosa und Konstrukten aus Programmiersprachen
  - solange Bedingung tu irgendwas, falls dann Bedingung tu dieses sonst jenes
- Graphische Darstellungen, Ablaufpläne, Struktogramme
  - machen Kontrollfluss sichtbar
- Programmiersprache
  - Präzision, Ausführbarkeit auf Maschine garantiert

## Berechenbarkeit von Funktionen

- Eine Funktion  $f: M \rightarrow N$  heißt berechenbar genau dann, wenn es einen Algorithmus gibt, der für jeden Eingabewert  $m \in M$ ; für den  $f(m)$  definiert ist, nach endlich vielen Schritten anhält und als Ergebnis  $f(m)$  liefert. In allen Fällen, in denen  $f(m)$  nicht definiert ist, bricht der Algorithmus nicht ab.
- Beispiele für berechenbare Funktionen:
  - Bestimmen des größten gemeinsamen Teilers,  $\text{ggT} : \text{Nat} \times \text{Nat} \rightarrow \text{Nat}$
  - Feststellen, ob eine natürliche Zahl Primzahl ist,  $\text{prim} : \text{Nat} \rightarrow \{\text{True}, \text{False}\}$
  - Sortieren von  $r$  natürlichen Zahlen:  $\text{sort} : \text{Nat}^r \rightarrow \text{Nat}^r$

## Nicht-berechenbare Funktionen: Das Halteproblem

UNIVERSITÄT LEIPZIG



- Motivation: Fehler, die eine Programmterminierung verhindern und Endlosschleifen bedeuten.
- Gesucht: Programm, das vor dem Ausführen von anderen Programmen prüft, ob eine Endlosschleife existiert.
- Halteproblem: Bestimmen Sie ein Programm, das für ein beliebiges anderes Programm ermittelt, ob es eine Endlosschleife enthält.
- Potenzielle Lösung: stopp-tester: PROGRAMM X DATEN  
→ {hält, hält-nicht}

DigInf 05/06

39

## Nicht-berechenbare Funktionen: Das Halteproblem

UNIVERSITÄT LEIPZIG



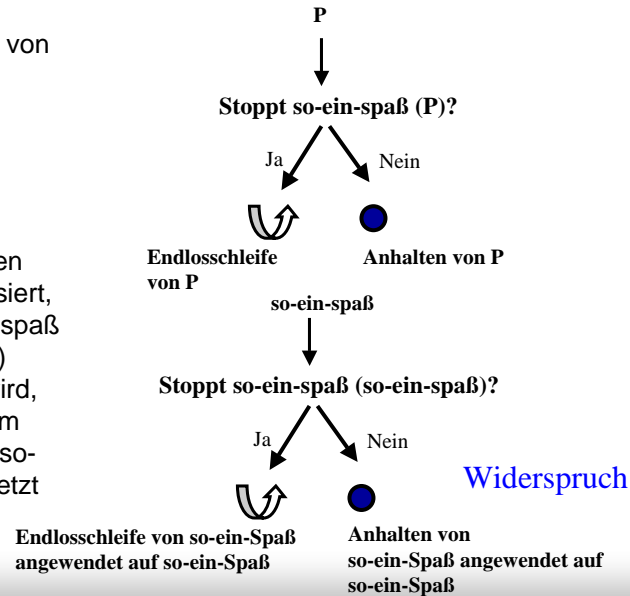
- Beweis, dass so ein stopp-tester nicht existieren kann:
  - Annahme: stopp-tester existiert
  - Da stopp-tester alle Programme mit allen Daten prüfen soll, können wir stopp-tester-neu folgendermaßen definieren:
    - stopp-tester-neu (P) prüft, ob, ein Programm P endet, falls es mit den Daten P aufgerufen wird, also stopp-tester-neu (P) = stopp-tester (P,P)
  - Wenn stopp-tester existiert und damit dann auch stopp-tester-neu, dann definieren wir so-ein-spaß mit der Eingabe P wie folgt:
    - so-ein-spaß (P) = ( Falls stopp-tester-neu (P) = hält-nicht ) dann stoppe sonst schleife endlos

DigInf 05/06

40

## Nicht-berechenbare Funktionen: Das Halteproblem

- Arbeitsweise von so-ein-spaß (grafisch):
- Nun überlegen wir, was passiert, wenn so-ein-spaß (so-ein-spaß) ausgeführt wird, also in obigem Bild P durch so-ein-spaß ersetzt wird.



## Nicht-berechenbare Funktionen: Das Halteproblem

- Auflösung des Widerspruchs nur dadurch möglich, dass so-ein-spaß nicht existieren kann.
- Einzige Voraussetzung war die Existenz stopp-tester-neu. Also, kann stopp-tester-neu nicht existieren.
- Einzige Voraussetzung war die Existenz von stopp-tester, Also, kann stopp-tester nicht existieren.
- Widerspruch zur Ursprungsannahme.