



Designing Education  
Connecting People

## Das erwartet Sie:

- Daten und Informationen unterscheiden
- Prozess der Softwareentwicklung



# Software zur Verwaltung von Daten anpassen



Lernfeld 5

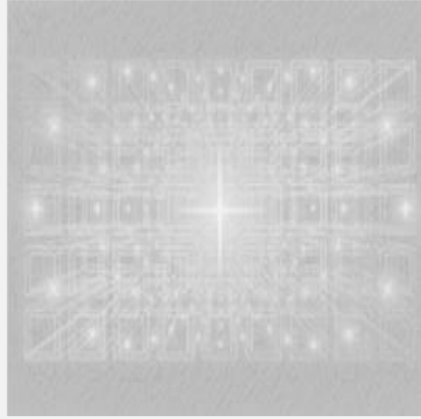
# Die Themen und Lernziele



Das Umfeld der  
Softwareentwicklung  
analysieren

**Lernziel**

Aufgaben und  
Kompetenzen in der SE  
kennenlernen



Grundlagen zur  
Verwaltung von  
Daten

Lernziel

Information versus Daten



Den Prozess der  
Software-  
entwicklung  
analysieren

**Lernziel**

Prozessphasen sowie  
Vorgehensmodelle  
kennenlernen



Den Prozess der  
Anforderungs-  
spezifikation  
beschreiben

**Lernziel**

Anforderungen an die  
zukünftige Software  
spezifizieren können



Einfache  
Anwendungen in  
Python schreiben

**Lernziel**

Programmiersprachen  
und –werkzeuge  
unterscheiden lernen

# Die Themen und Lernziele



Auf Dateien in  
Anwendungen  
zugreifen

---

**Lernziel**

Daten speichern und  
einlesen lernen



Verwaltung der  
Daten mithilfe von  
Datenbanken

---

**Lernziel**

Grundlagen von  
relationalen Datenbanken



Software testen  
und dokumentieren

---

**Lernziel**

Qualitätsbewusstsein  
entwickeln



Prozess der  
Softwareentwicklung  
evaluieren

---

**Lernziel**

Reflexion



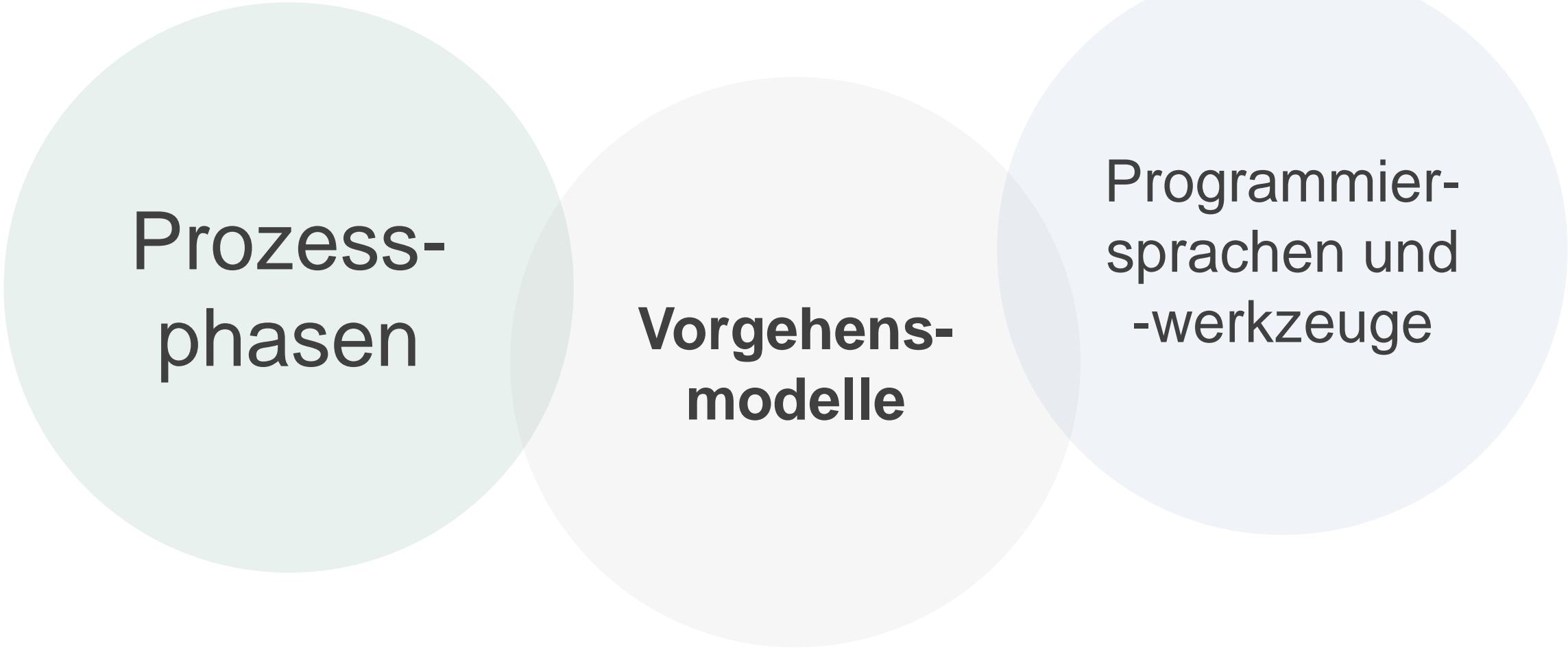
# Den Prozess der Software- entwicklung analysieren

## Lernziel

Prozessphasen sowie  
Vorgehensmodelle  
kennenlernen

# Der heutige Tag

Prozessanalyse der Softwareentwicklung



The diagram consists of three overlapping circles arranged horizontally. The leftmost circle is light green and contains the text 'Prozessphasen'. The middle circle is light gray and contains the text 'Vorgehensmodelle'. The rightmost circle is light blue and contains the text 'Programmiersprachen und -werkzeuge'. The circles overlap in the center, creating a Venn-like structure.

**Prozess-  
phasen**

**Vorgehens-  
modelle**

**Programmier-  
sprachen und  
-werkzeuge**



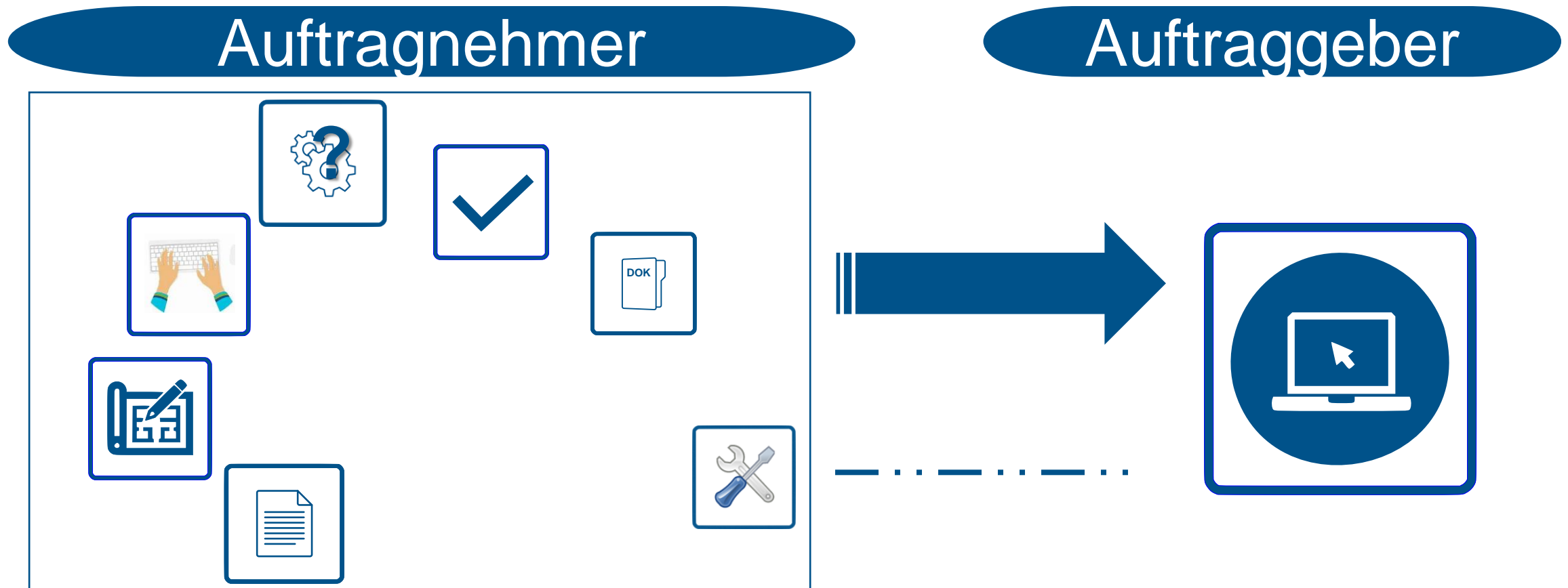
## 5.3.1 Die Prozessphasen beschreiben

Softwarelebenszyklus



## 5.3.1 Die Prozessphasen beschreiben

### Phasen des Softwareentwicklungsprozess



Nach IEEE-Standard 12207



Welche Aussage ist richtig?

- a) Grundlage der Softwareentwicklung ist der Softwarelebenszyklus.
- b) In der Analysephase werden die Anforderungen an die zu entwickelnde Software beschrieben.
- c) In der Planungsphase kommen Modellierungssprachen zu Einsatz.
- d) In der Umsetzungsphase wird die Software programmiert und notfalls an eine Datenbank angebunden.
- e) Die Testphase kann neben Modultests auch andere Tests enthalten.
- f) Die Kommentierung von Quellcode gehört zur Dokumentationsphase.
- g) Die Phasen der Softwareentwicklung werden immer auf die gleiche Art und Weise durchlaufen.



## 5.3.2 Software im Rahmen eines Projektes entwickeln

### Projektmanagement

engl. „to manage“ =

- ... koordinieren, betreuen, verwalten, erledigen,  
bewerkstelligen, schaffen, administrieren etc.

*Planen*

Festlegen, was gemacht wird

*Organisieren*

Geplantes zum Funktionieren bringen

*Überwachen/Steuern*

Verfolgen im Hinblick auf erfolgreiche Abwicklung

*Führen*

Zielorientiertes Anleiten anderer

„Das Projektmanagement ist die Anwendung von Methoden, Hilfsmitteln, Techniken und Kompetenzen in einem Projekt, um die eigentliche Projektarbeit effizient und zielführend zu gestalten.“ (ISO 21500)

## 5.3.2 Software im Rahmen eines Projektes entwickeln

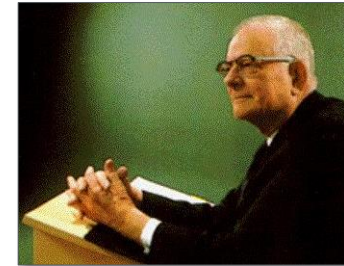
### Der „Deming-Zyklus“

- Standardisierung erfolgreicher Vorgehensweisen und Ergebnisse
- Reflexion des Prozesses
- Anstoß von Folgeaktivitäten

- Ist-Soll-Abgleich
- Anpassung bei Abweichungen
- Darstellung und Überprüfung der Ergebnisse



- Analyse der Ist-Situation, Beschreibung des Problems, Sammlung von Informationen
- Formulierung von Zielen
- Festlegung von Maßnahmen zur Lösung, Verbesserung oder Optimierung



William Edwards Deming

- Durchführung der Maßnahmen unter Einhaltung des Zeit- und Ressourcenplans
- Dokumentation der Maßnahmen

## 5.3.2 Software im Rahmen eines Projektes entwickeln

### Erfolgsfaktoren

- Genaue Definition von Projektzielen
- Entwicklung präziser Projektpläne und Kontrolle von deren Einhaltung
- Projektspezifische Gestaltung von Projektorganisation und Projektkultur
- Zusammenstellen eines geeigneten Projektteams
- Optimale Kommunikationsbedingungen

### Risikofaktoren

- Schlechte Kommunikation  
(z. B. Sprachbarrieren, unvereinbare Persönlichkeiten, schlechte Kommunikationskanäle)
- Schlechte Planung  
(z. B. knappe Ressourcen, zu wenig Zeit)
- Mangelnder Überblick über die Projektdetails
- Verwendung falscher Projektmanagementtools  
(z. B. Software nicht flexibel genug für das Projekt)
- Mangelnde Überwachung und Kontrolle
- Kein Risikomanagement
- Änderungen im Umfang (z. B. Fristen werden verschoben, Aufgaben geändert)



Welche Aussage ist richtig?

- a) Ein professionelles Projektmanagement ist wesentlich für die Durchführung von Projekten.
- b) Bei Projekten wird in der Regel eine Risikoabschätzung vorgenommen.
- c) Teambildung ist eine Aufgabe des Projektmanagements.
- d) Die Ressourcen- und Ablaufplanung spielt eine große Rolle im Projektmanagement.
- e) Die Kontrolle des Projektfortschritts ist in der heutigen Zeit nicht mehr so wichtig.
- f) Softwareprojekte können nach verschiedenen Vorgehensmodellen bearbeitet werden.

## 5.3.3 Vorgehensmodelle unterscheiden

### Klassische Vorgehensmodelle

- Wasserfallmodell
- Spiralmodell
- V-Modell
- Rational Unified (RUP)
- ...

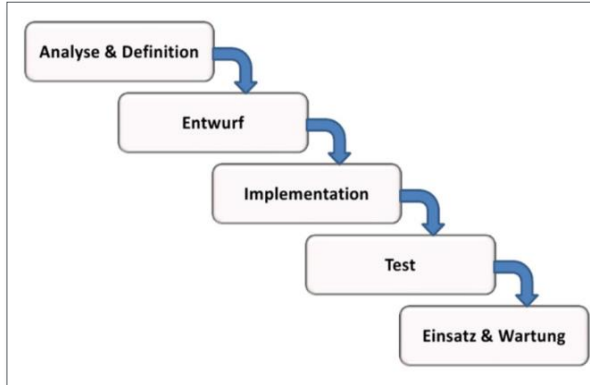
### Agile Vorgehensmodelle

- Scrum
- DevOps-Ansatz
- Extreme Programming (XP)
- Kanban
- ...

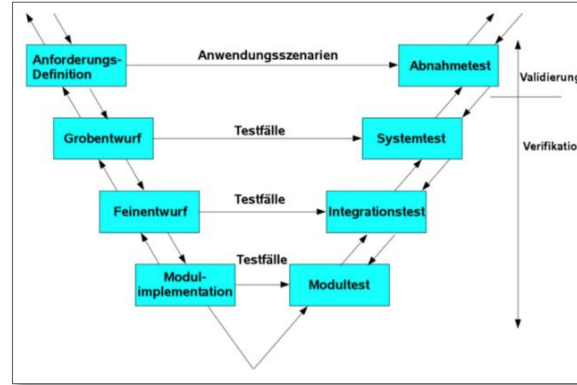
# 5.3.2 Software im Rahmen eines Projektes entwickeln

## Phasenmodelle und agile Frameworks

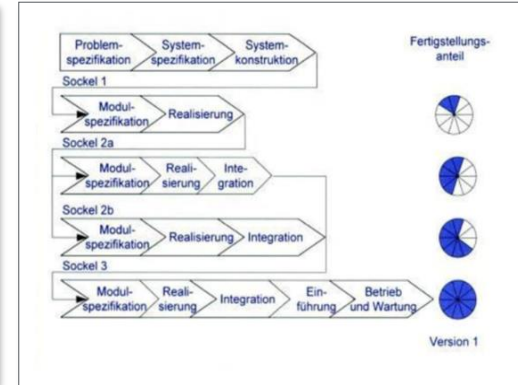
Sequentielle Modelle



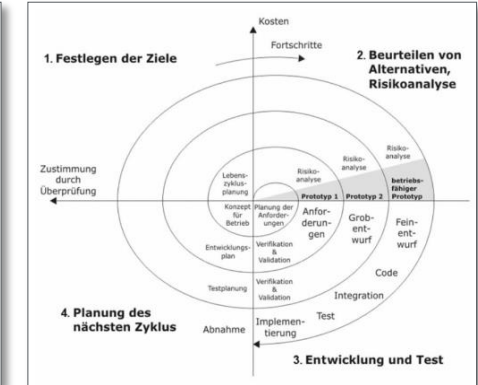
V - Modell



Iterative Modelle



Spiralmodelle

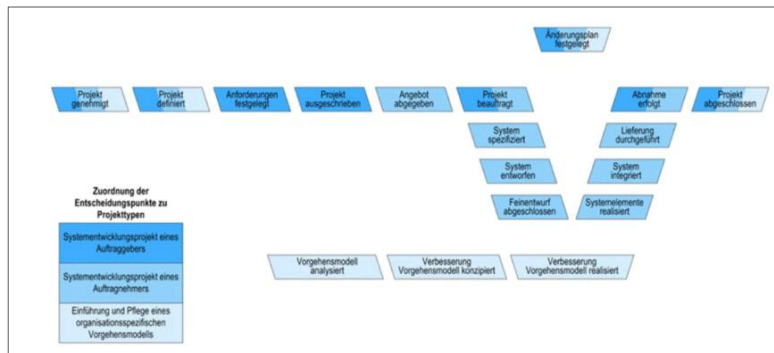


Klassische Vorgehensmodelle

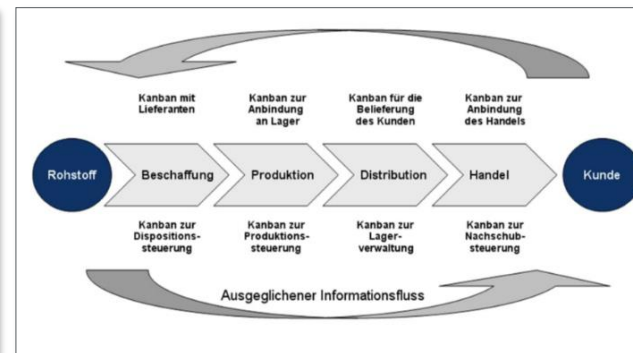
vs.

Agile Vorgehensmodelle

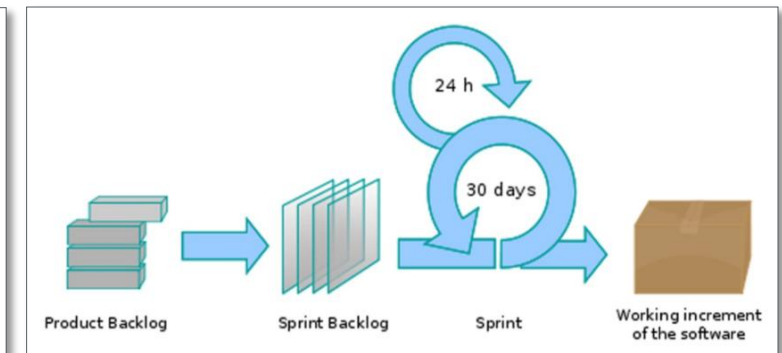
V-Modell XT



KANBAN



SCRUM





## 5.3.3 (1) Vorgehensmodelle unterscheiden

Das Wasserfallmodell ist ein Basismodell der Ablaufplanung in Projekten

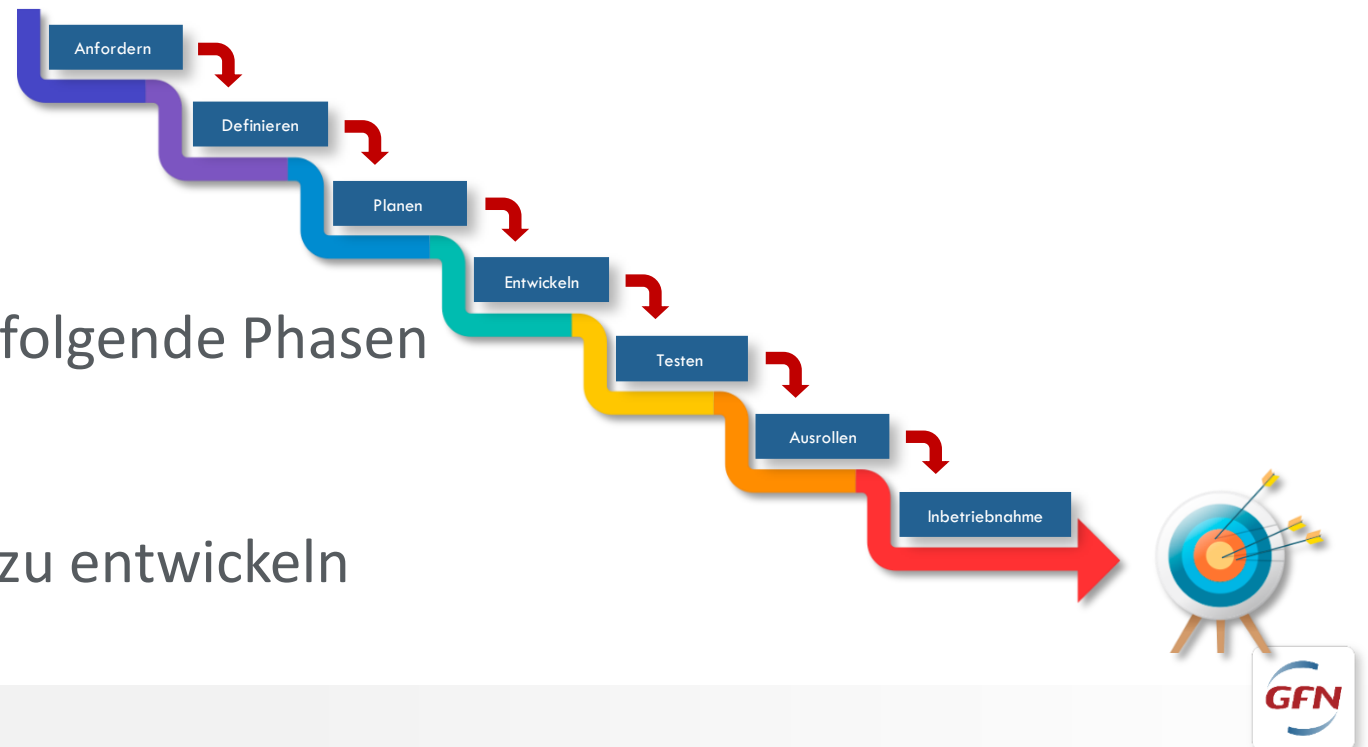
- Phasenergebnisse gehen immer als bindende Vorgaben für die nächsttiefere Phase ein
- Lineares / sequentielles Vorgehensmodell
- Beliebt in der Softwareentwicklung

### Vorteil:

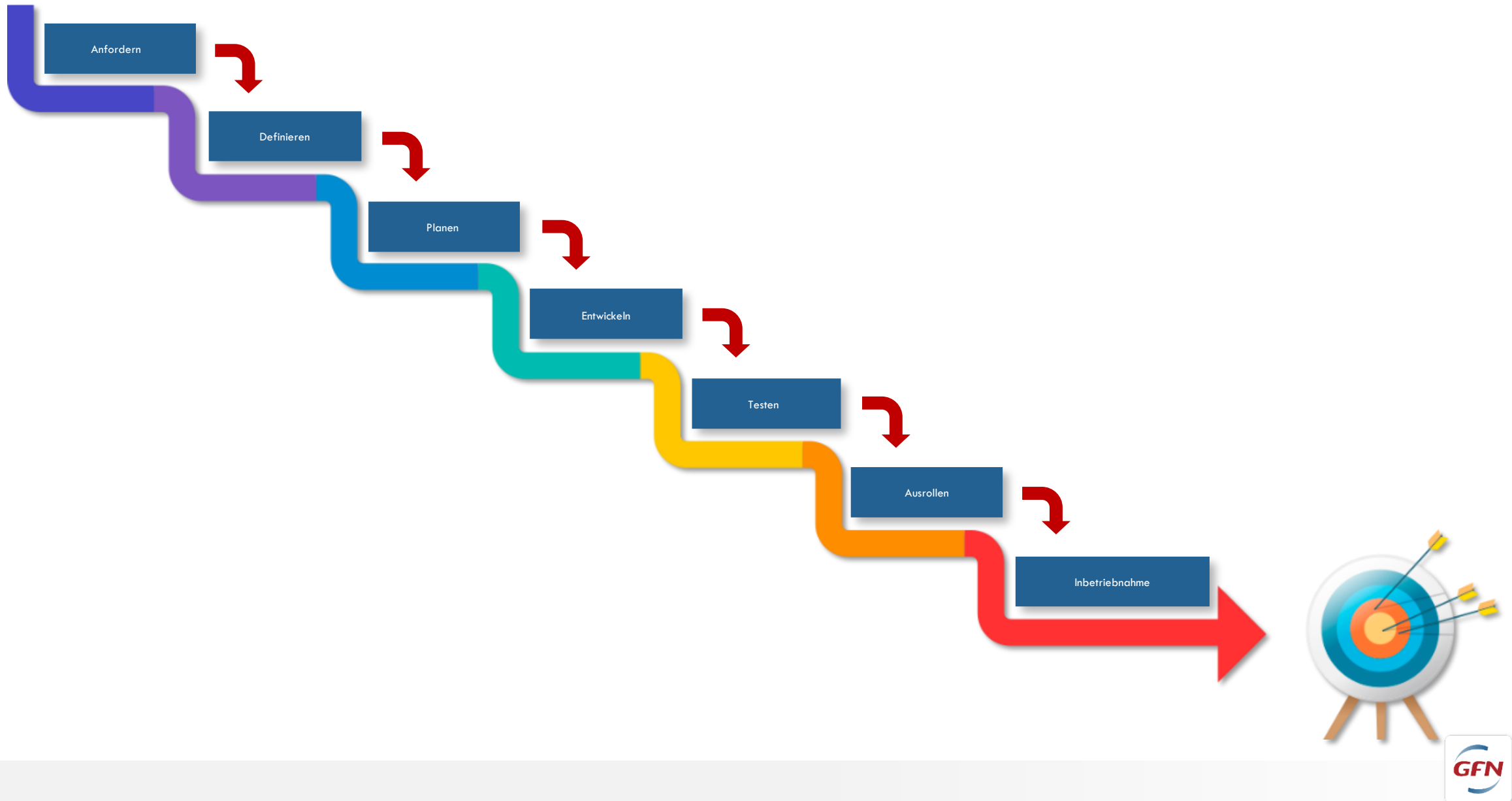
- Minimales „Planing Overhead“ für folgende Phasen

### Nachteil:

- Hohes Risiko, „am Kunden vorbei“ zu entwickeln



## 5.3.3 (1) Vorgehensmodelle unterscheiden



### 5.3.3 <sup>(1)</sup> Wasserfallmodell

- Aktivitäten sind in der vorgegebenen Reihenfolge und in der vollen Breite vollständig durchzuführen
- Am Ende jeder Aktivität steht ein fertiggestelltes Dokument, d. h., das Wasserfallmodell ist ein „dokumentgetriebenes“ Modell
- Der Entwicklungsablauf ist sequenziell, d. h., jede Aktivität muss beendet sein, bevor die nächste anfängt
- Es orientiert sich am sogenannten Top-Down-Verfahren
- Es ist einfach und verständlich
- Eine Benutzerbeteiligung ist in der Anfangsphase vorgesehen, anschließend erfolgen der Entwurf und die Implementierung ohne Beteiligung des Benutzers bzw. Auftraggebers. Weitere Änderungen stellen danach Neuaufträge dar

### 5.3.3 <sup>(1)</sup> Wasserfall Vorteile

- Klare Abgrenzung der Phasen
- Einfache Möglichkeiten der Planung und Kontrolle
- Bei stabilen Anforderungen und klarer Abschätzung von Kosten und Umfang  
sehr effizientes Modell

## 5.3.3 <sup>(1)</sup> Wasserfall Nachteile

- *Abgrenzungsproblem*: Klar voneinander abgegrenzte Phasen sind häufig unrealistisch – der Übergang zwischen ihnen ist fließend: Teile eines Systems können sich noch in der Planung befinden, während andere schon in der Ausführung oder im Gebrauch sind
- *Abfolgeproblem*: Einzelne Phasen laufen in der Theorie nacheinander ab, in der Praxis sind jedoch Rückschritte oft unvermeidlich
- Unflexibel gegenüber Änderungen und im Vorgehen (Phasen müssen sequenziell abgearbeitet werden)
- Frühes Festschreiben der Anforderungen ist oft problematisch, da es eventuell zu teuren Änderungen führt (mehrmals wiederholtes Durchlaufen des Prozesses bei Änderungen)
- Einführung des Systems sehr spät nach Beginn des Entwicklungszyklus, deshalb ein später Return on Investment (ROI)
- Fehler werden unter Umständen spät erkannt (*Big Bang*) und müssen mit erheblichem Aufwand entfernt werden

### 5.3.3 (2) Spiralmodell

Im Spiralmodell werden die folgenden vier Phasen bis zum Projektabschluss wiederholt:

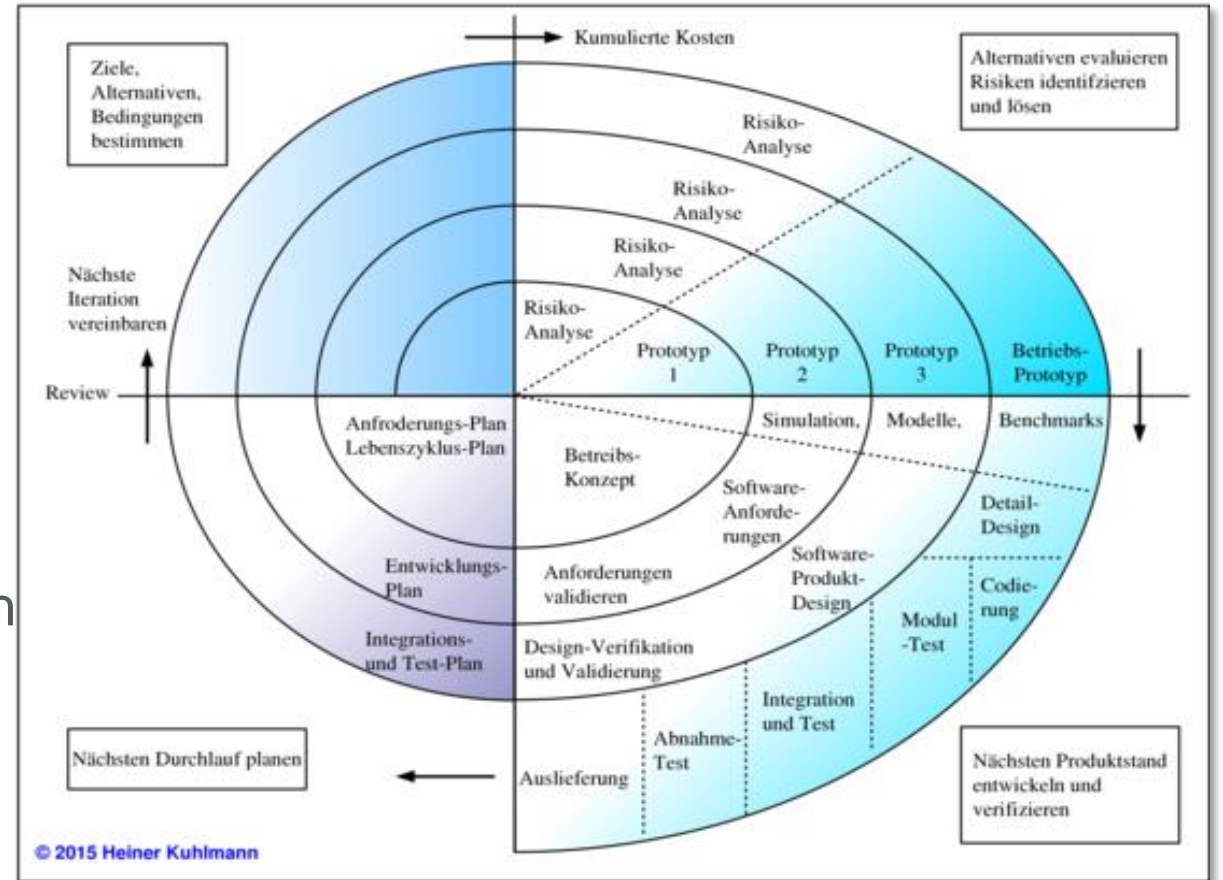
1. **Zielbestimmung** für diese Iteration
2. **Risikoanalyse:** Alternativen finden und bewerten
3. **Ausführung** der besten Alternative
4. **Review** und Planung der nächsten Iteration

## Vorteil:

- Änderungen an Software jederzeit möglich

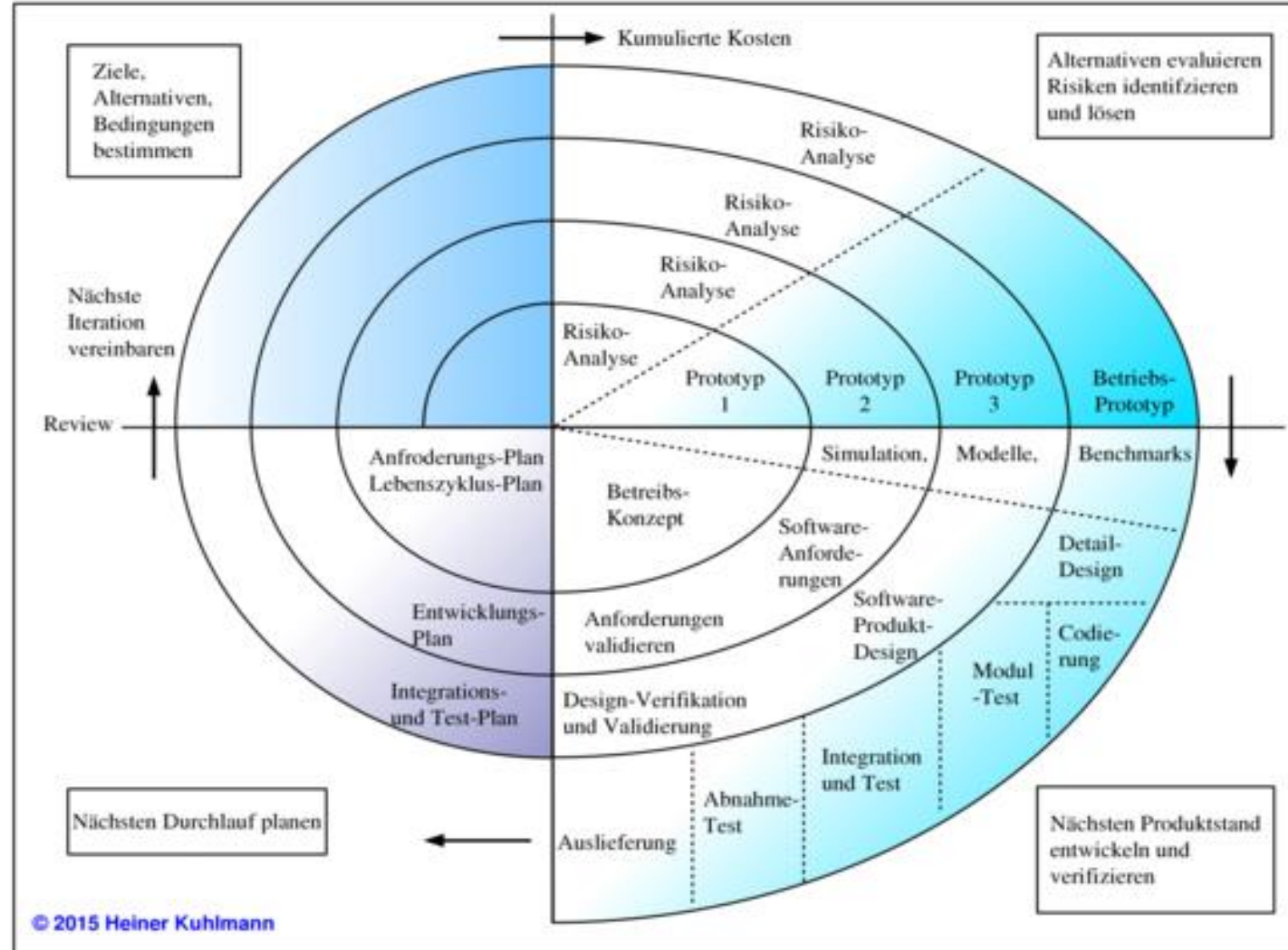
## Nachteil:

- Expertenwissen wird benötigt; eher zu komplex





## 5.3.3 (2) Spiralmodell



## 5.3.3 (3) V-Modell

Das V-Modell® wurde aus dem Wasserfallmodell abgeleitet und ist prinzipiell sowohl zur Bearbeitung von Softwareprojekten aber auch für die Bearbeitung anderer Projekte ausgelegt

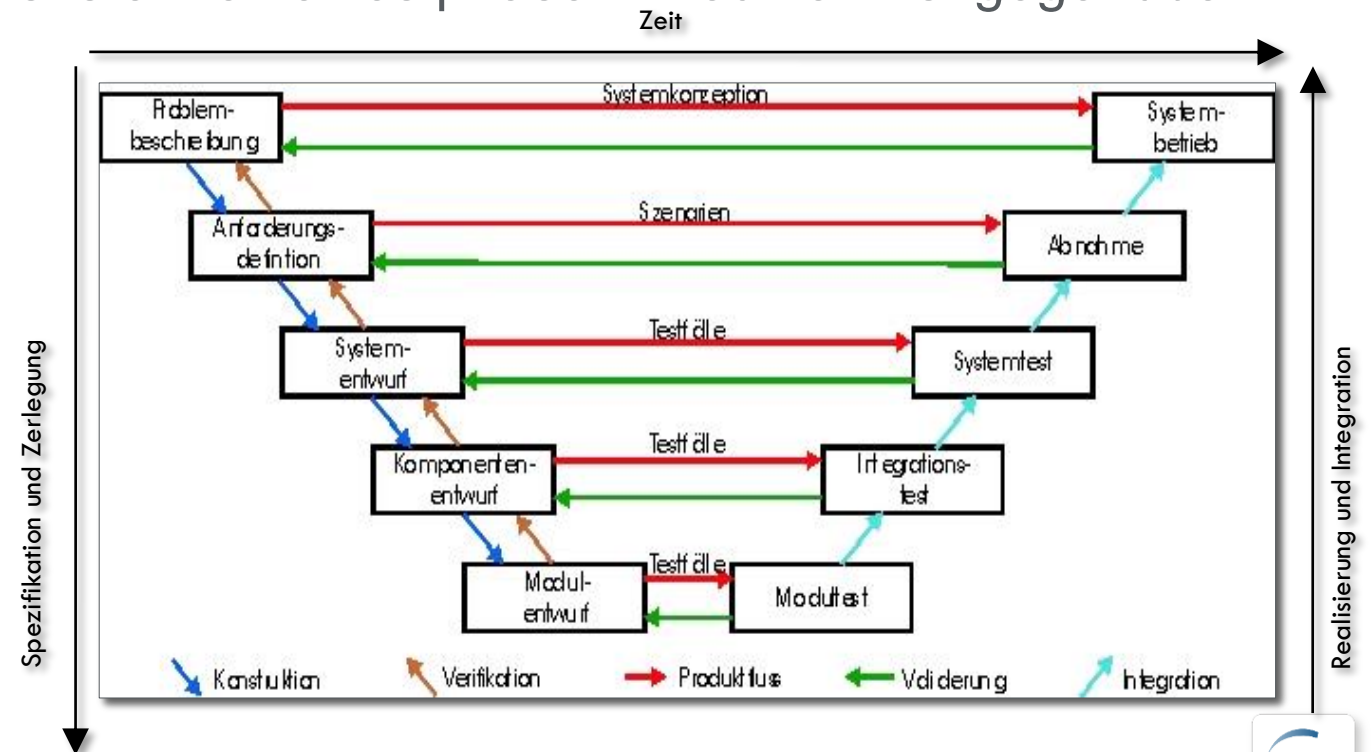
Jeder Spezifizierungsphase im linken Ast steht eine Testphase im rechten Ast gegenüber

### Vorteil:

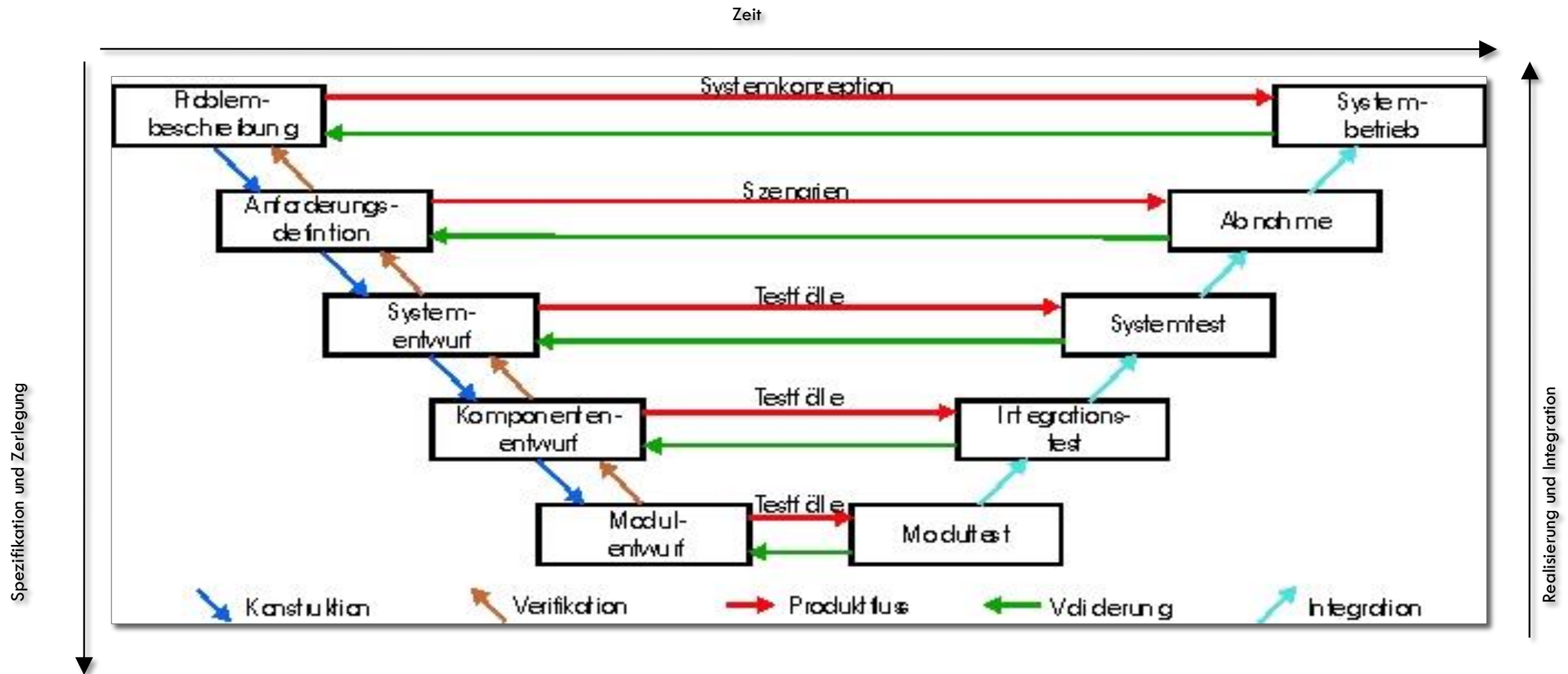
- Minimierung von Risiken unnötiger Entwicklung

### Nachteil:

- Eher starre Struktur / unflexibel für Änderungen



## 5.3.3 (3) V-Modell



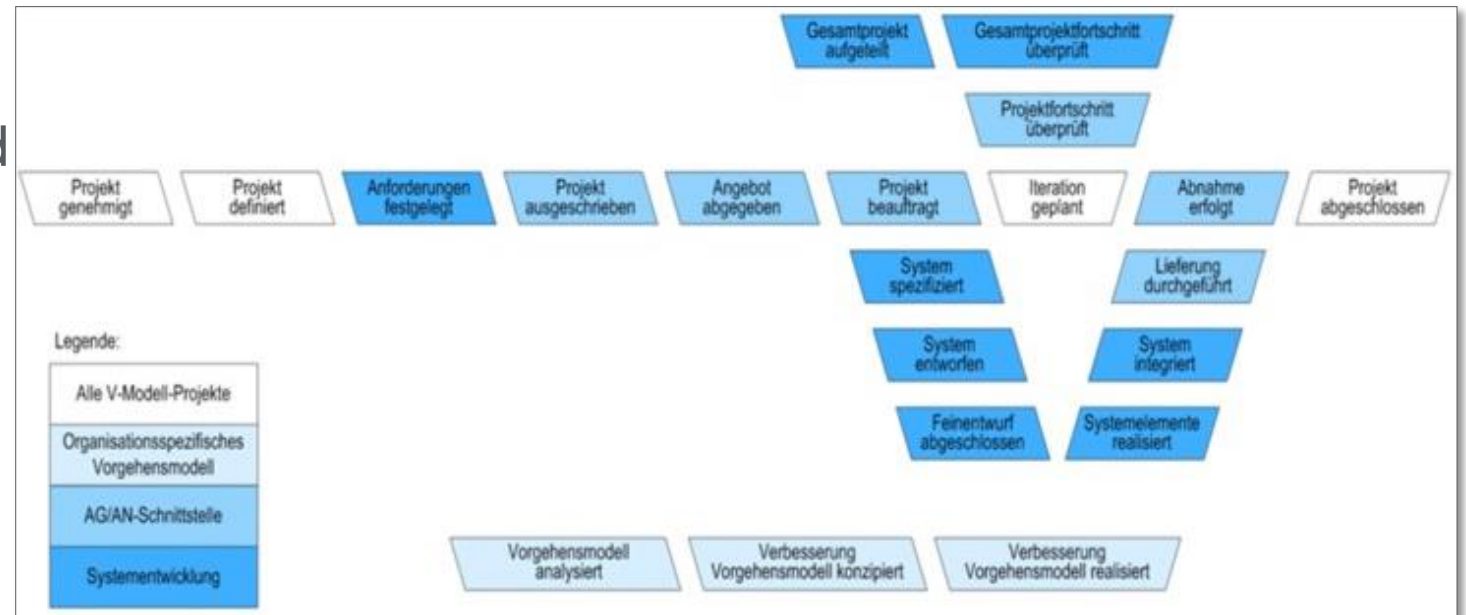
## 5.3.3 <sup>(3)</sup> V-Modell XT

Seit 2005 V-Modell® XT als Entwicklungsstandard für IT-Systeme des Bundes

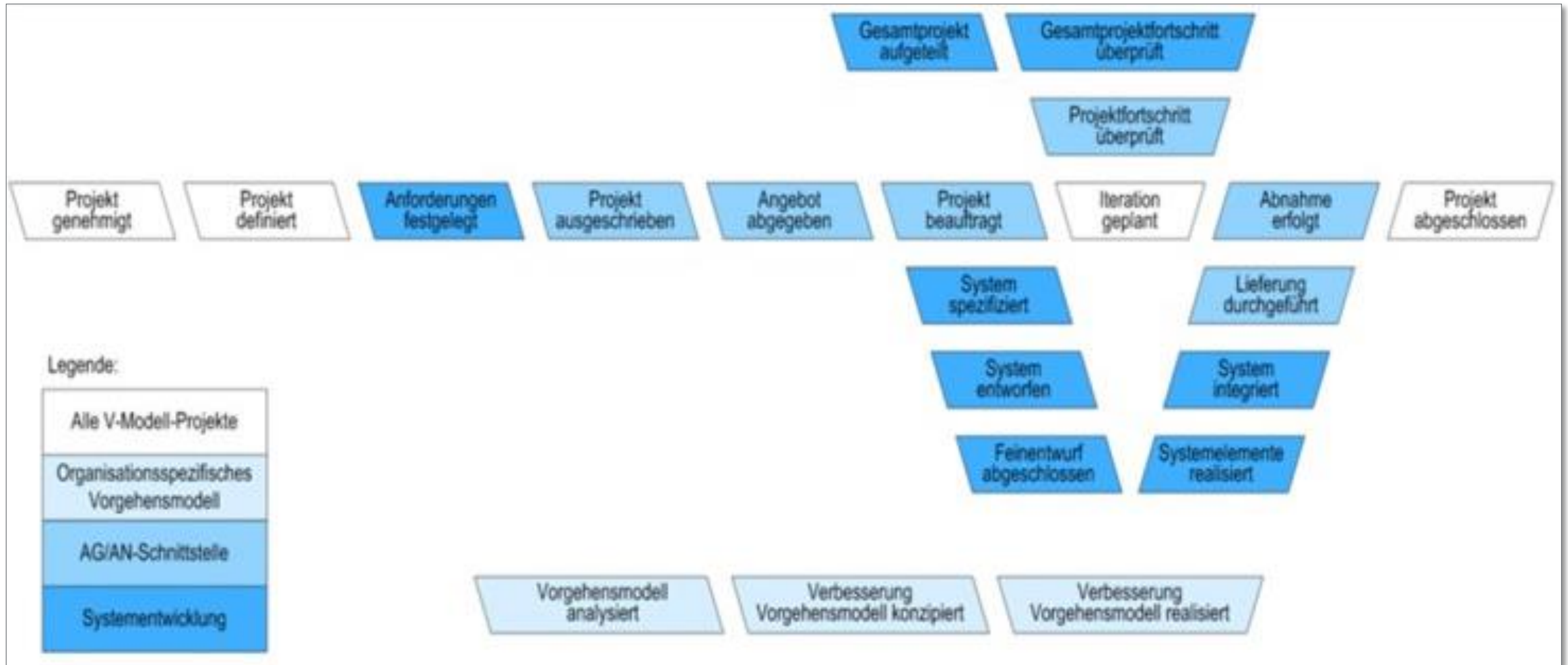
- Für die Planung und Durchführung von IT Projekten verbindlich vorgeschrieben
- Spezifikationen der jeweiligen Entwicklungsstufen als Grundlage für Tests (Teststufen)

Vorgehensmodell legt einheitlich fest:

- Was zu tun ist
- Wie Aufgaben durchzuführen sind
- Womit dies zu geschehen hat



## 5.3.3 (3) V-Modell XT





## 5.3.3 (4) Scrum

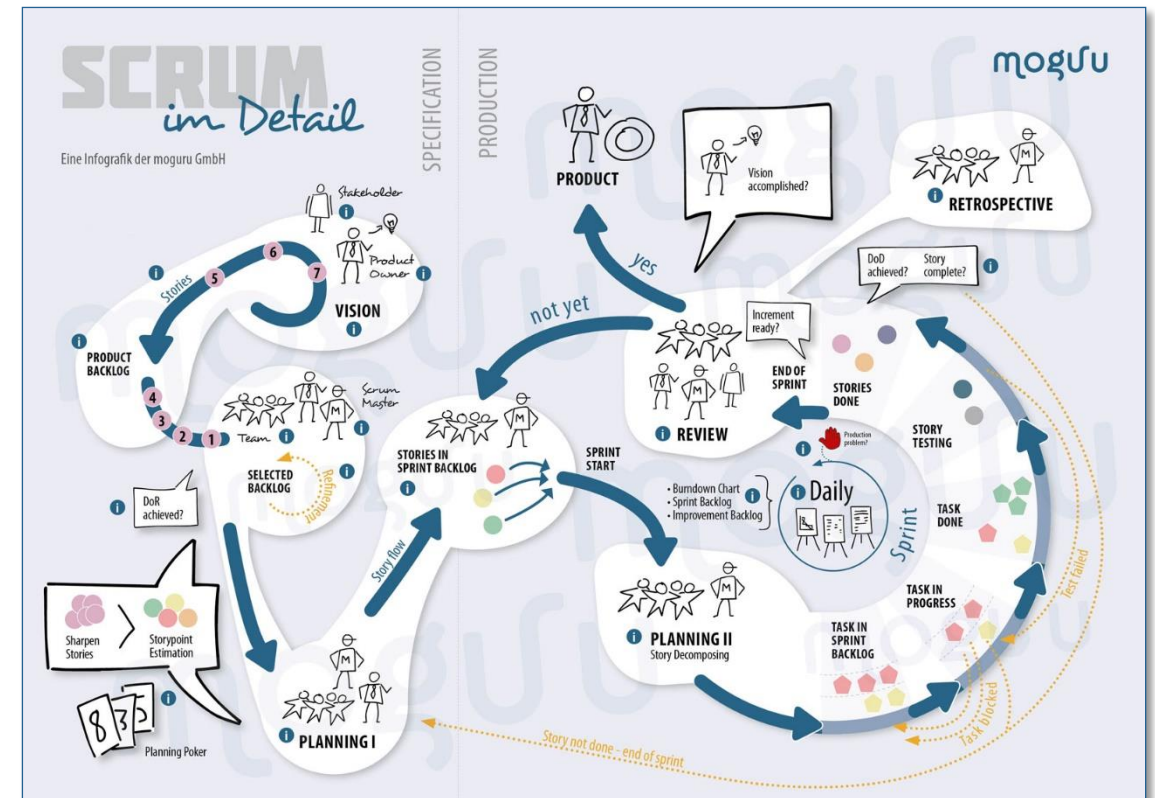
- **Teammitglieder und Zusammenarbeit** über Prozesse und Werkzeuge
- **Funktionierende Software** über umfangreiche Dokumentation
- **Zusammenarbeit mit dem Auftraggeber** über vertragliche Vereinbarungen
- **Eingehen auf Veränderungen** über Festhalten am Plan

### Vorteil:

- Zeitnahe Realisierung der Inkremente

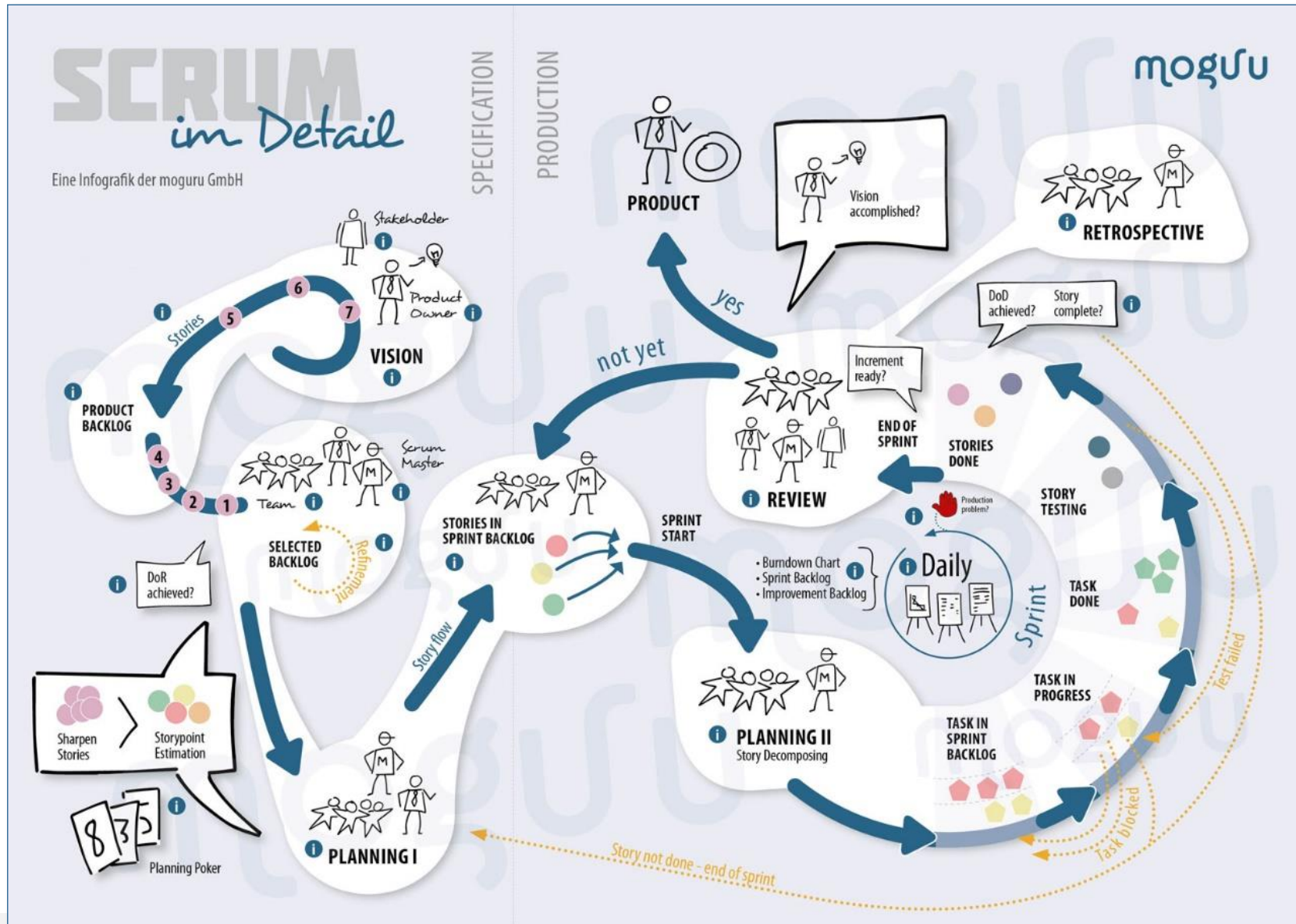
### Nachteil:

- Geringe Planbarkeit von Kosten und Zeit



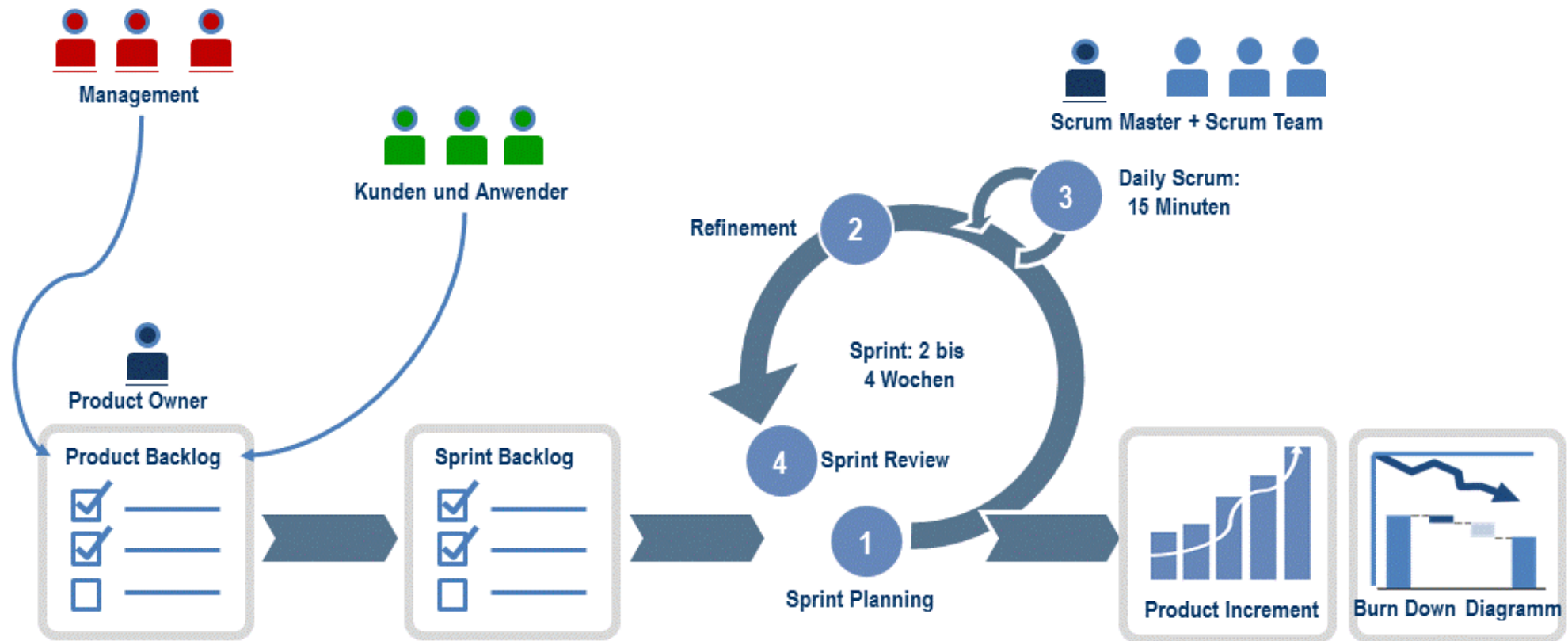


## 5.3.3 (4) Scrum



## 5.3.3 (4) Scrum

### Mit Scrum zur agilen Entwicklung: Wie funktioniert die Methode?



## 5.3.3 (4) Scrum

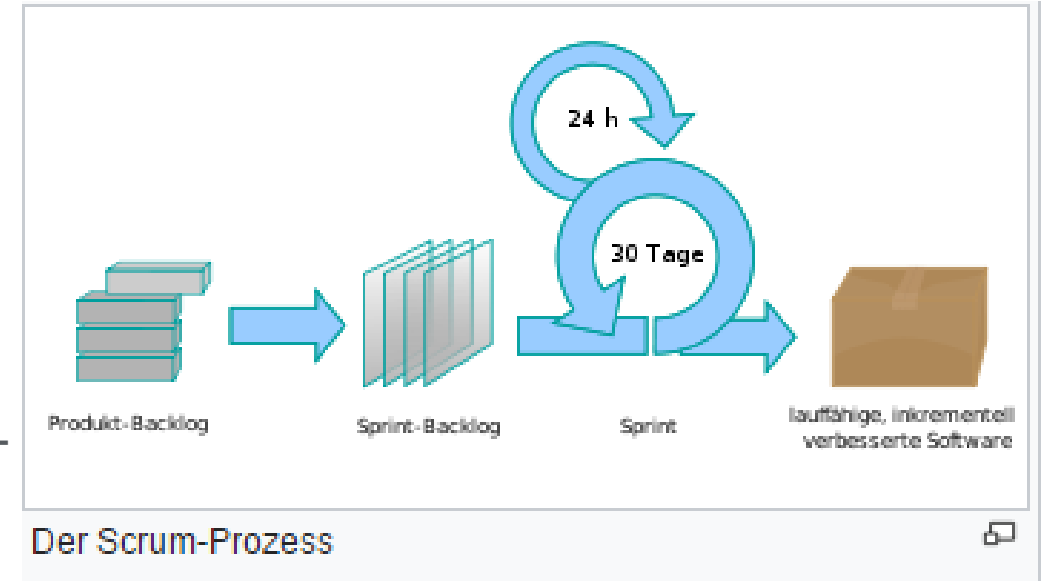
Der Ansatz von Scrum ist empirisch, inkrementell und iterativ

Die empirische Verbesserung fußt auf drei Säulen:

Transparenz: Fortschritt und Hindernisse eines Projektes werden regelmäßig und für alle sichtbar festgehalten

Überprüfung: In regelmäßigen Abständen werden Produkt-funktionalitäten geliefert und sowohl das Produkt als auch das Vorgehen beurteilt

Anpassung: Anforderungen an das Produkt, Pläne und Vorgehen werden nicht ein für alle Mal festgelegt, sondern kontinuierlich detailliert und angepasst. Scrum reduziert die Komplexität der Aufgabe nicht, strukturiert sie aber in kleinere und weniger komplexe Bestandteile, die Inkremente



## 5.3.3 (4) Scrum Vorteile

- Wenige Regeln, leicht verständlich und schnell einführbar
- Kurze Kommunikationswege
- Hohe Flexibilität/Agilität durch adaptives Planen
- Hohe Effektivität durch Selbstorganisation
- Hohe Transparenz durch regelmäßige Meetings und Backlogs
- Zeitnahe Realisation neuer Produkteigenschaften bzw. Inkremente
- Kontinuierlicher Verbesserungsprozess
- Kurzfristige Problem-Identifikation
- Geringer Administrations- und Dokumentationsaufwand

## 5.3.3 (4) Scrum Nachteile

- Kein Gesamtüberblick über die komplette Projektstrecke
- Hoher Kommunikations- und Abstimmungsaufwand
- Wenige konkrete Handlungsempfehlungen
- Zeitverluste bei zu „defensiven“ Sprintplanungen
- „Tunnelblick-Gefahr“ bei ausschließlicher Fokussierung auf Tasks
- Erschwerte Koordination mehrerer Entwicklungsteams bei Großprojekten
- Potenzielle Verunsicherung aufgrund fehlender Zuständigkeiten und Hierarchien
- Potenzielle Unvereinbarkeit mit bestehenden Unternehmensstrukturen

### 5.3.3 <sup>(3)</sup> DevOps-Ansatz

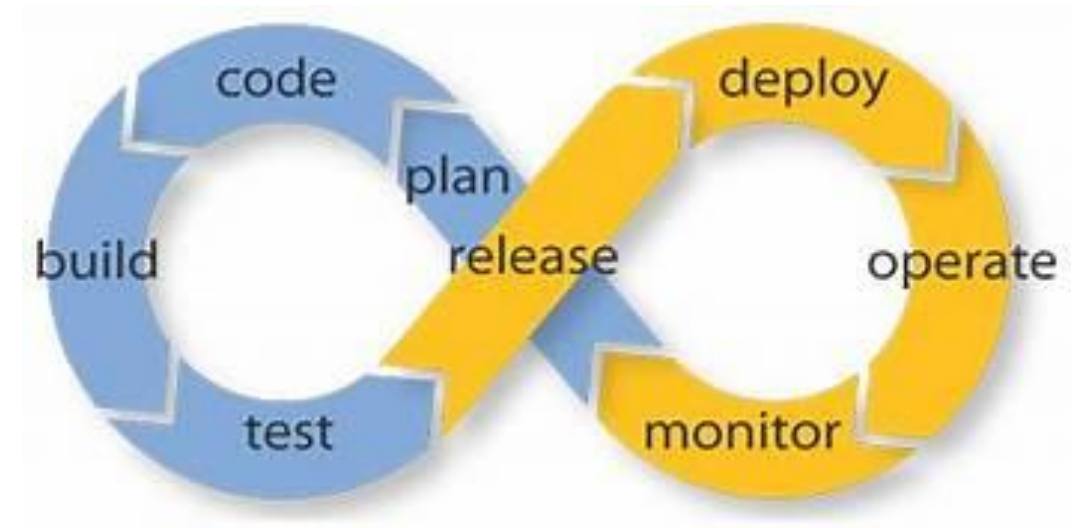
- **DevOps** setzt sich aus *Development* und *Operations* zusammen
- Wird von integrierten IT-Teams umgesetzt
- Verantwortung für Entwicklung und den Betrieb der Software
- Regelmäßiges Ausliefern von Software mit hoher Qualität

#### Vorteil:

- Erwartungen des Auftraggebers können präzise erfüllt werden
- Kürzere Problemlösungszeiten

#### Nachteil:

- Potentielle Unvereinbarkeit mit bestehenden Unternehmensstrukturen







Welche Aussage ist richtig?

- a) Mit agilen Vorgehensmodellen lässt sich nicht sehr flexibel auf veränderte Anforderungen reagieren.
- b) Im Wasserfallmodell werden die einzelnen Phasen linear durchlaufen.
- c) Ein großer Vorteil des Wasserfallmodells ist, dass der Auftraggeber in den Entwicklungsprozess stark eingebunden ist.
- d) Beim Spiralmodell werden einzelnen Phasen auch linear durchlaufen.
- e) Das Spiralmodell ist nicht für größere Projekte geeignet.
- f) Kleine Softwareprojekte werden am besten mit dem V-Modell umgesetzt.

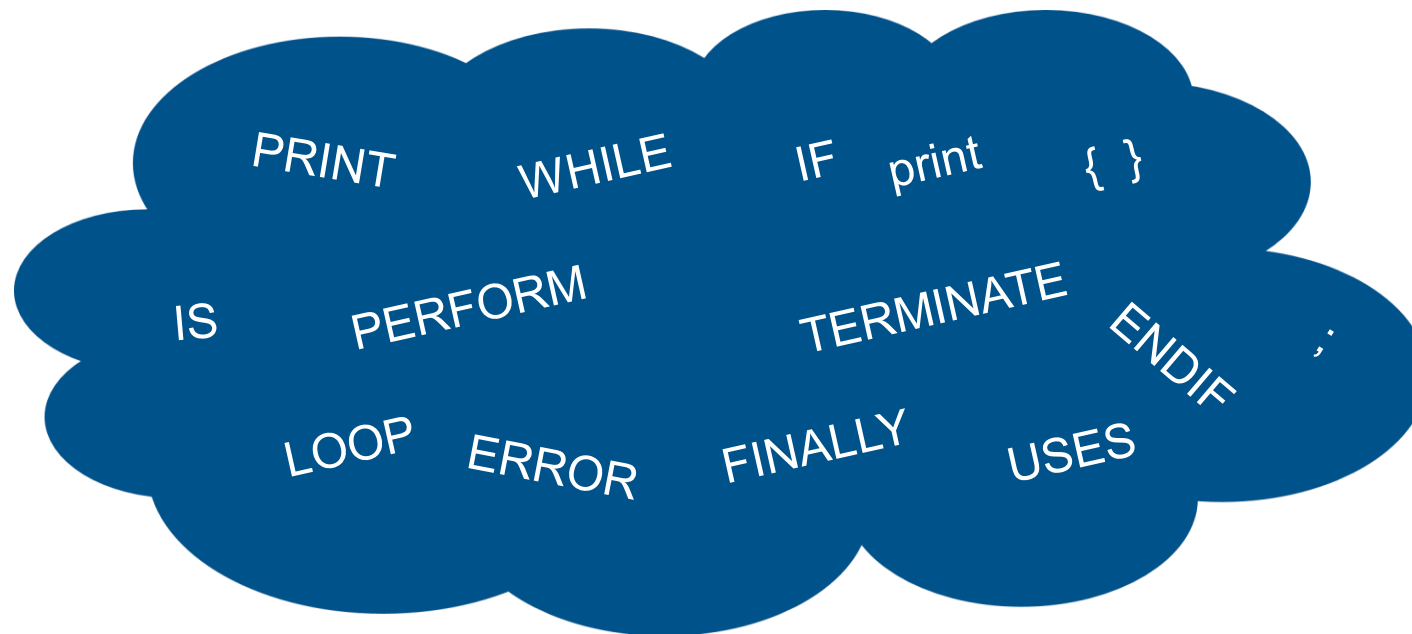


Welche Aussage ist richtig?

- g) Für größere Softwareprojekte im öffentlichen Dienst ist das V-Modell XT von großer Bedeutung.
- h) Mit Scrum kann man flexibel auf geänderte Anforderungen reagieren.
- i) Ein Sprint bei Scrum dauert mindestens vier Wochen.
- j) Der DevOps-Ansatz verbindet die Softwareentwicklung mit dem Betrieb der Software.
- k) Beim DevOps-Ansatz wird auf das Testen der Software meistens ganz verzichtet.

## 5.3.4 Programmiersprachen und –werkzeuge unterscheiden

Gemäß Duden ist eine Programmiersprache ein „System von Wörtern und Symbolen, die zur Formulierung von Programmen für die elektronische Datenverarbeitung verwendet werden“.



## 5.3.4 (1) Aufbau von Programmiersprachen

Bestandteile einer Programmiersprache

### Syntax

„Grammatik“ der  
Programmierung

### Schlüssel wörter

Bezeichner zur  
Verwendung  
Syntaxausdrücke,  
meist englisch,  
„reserviert“

### Zeichen- satz

Festgelegte Alphabet. u.  
numerische Zeichen  
sowie Sonderzeichen

## 5.3.4 (1) Aufbau von Programmiersprachen

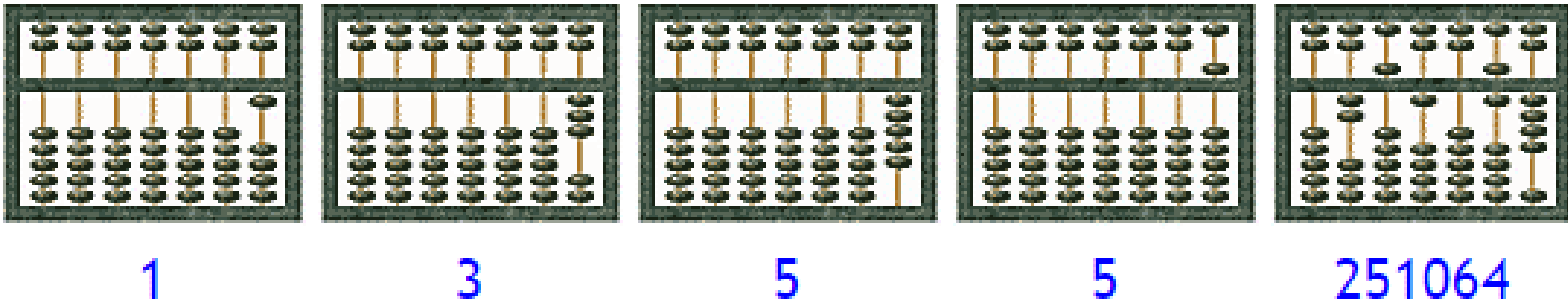
Grundfunktionen:

- Befehle zur Ein- und Ausgabe von Daten
- Befehle zur Deklaration von Variablen und Datenstrukturen
- Funktionen zur Verarbeitung von Zeichenketten
- Mathematische Grundfunktionen
- Kontrollstrukturen für die bedingte Ausführung oder die wiederholte Ausführung bestimmter Programmteile

## 5.3.4 (2) Historische Entwicklung

Vor dem 20. Jahrhundert: Rechnen mit dem Abakus

Ein Abakus besteht im Wesentlichen aus vertikal angeordneten Sprossen, auf denen verschiebbare Perlen befestigt sind. Beim chinesischen *suan pan* sitzen auf je einer Sprosse im oberen Bereich zwei Perlen; im unteren Bereich befinden sich auf jeder der Sprossen fünf Perlen. Ordnet man nun diese Perlen auf bestimmte Weise an, so lassen sich in eindeutiger Weise Zahlen darstellen. Die folgenden Beispiele zeigen, wie das geht:



## 5.3.4 (2) Historische Entwicklung

Zu jeder zulässigen Perlenanordnung gehört ein bestimmter Zahlenwert, wobei das dritte und das vierte Beispiel zeigen, dass die Umkehrung dieser Aussage nicht richtig ist. Wenn man alle Sprossenspalten *von rechts nach links* mit 0 bis 6 durchnummeriert, dann kann man mit einem solchen 7-sprossigen Abakus alle natürlichen Zahlen zwischen 0 und 1666665 darstellen und innerhalb dieses Zahlenraums addieren und subtrahieren, wenn folgende Regeln beachtet werden:

Addieren von $k \cdot 10^i$	Hochschieben von $k$ unteren Kugeln in der $i$ -ten Sprossenspalte
Subtrahieren von $k \cdot 10^i$	Herunterschieben von $k$ unteren Kugeln in der $i$ -ten Sprossenspalte
Addieren von $5 \cdot 10^i$	Herunterschieben einer oberen Kugel in der $i$ -ten Sprossenspalte
Subtrahieren von $5 \cdot 10^i$	Hochschieben einer oberen Kugel in der $i$ -ten Sprossenspalte

Eine Folge von Anweisungen, nach denen man unter Vorgabe von Eingangsdaten eine bestimmte Ausgabe erhält, nennt man **Algorithmus**.

# Vor dem 20. Jahrhundert

## 1614

- Ein Zeitaufwand von 30 Mannjahren ist erforderlich, um die von dem englischen Lord Napier herausgegebenen Logarithmentafeln zu erstellen
- Eine Rechenanlage nach heutigem Stand erledigt diese Arbeit in weniger als einer Minute



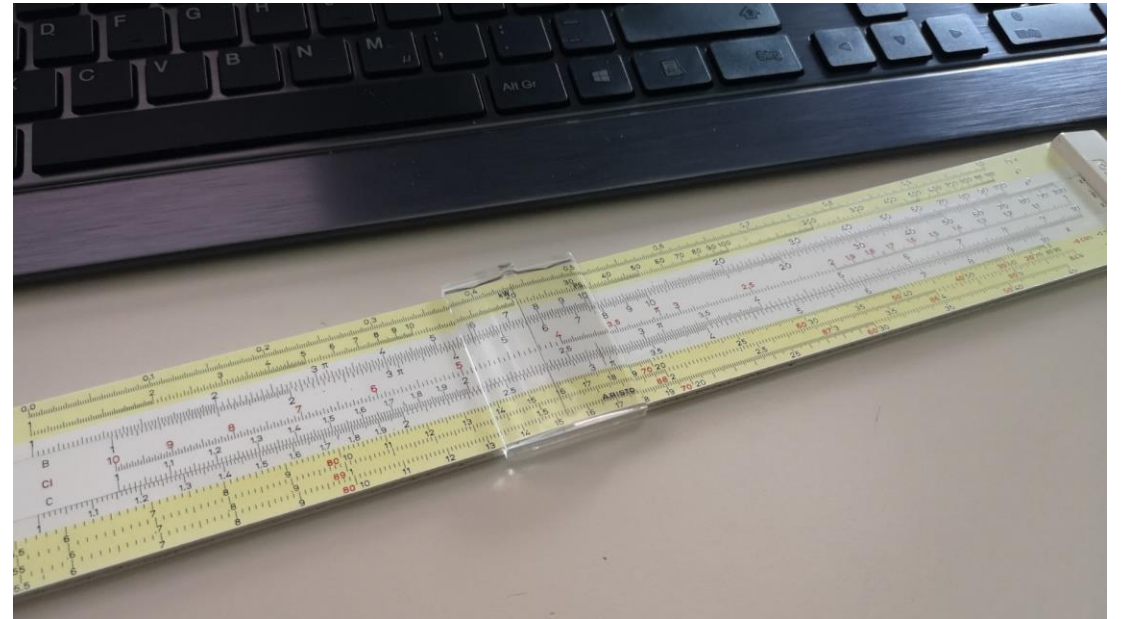
## 5.3.4 (2) Historische Entwicklung

1614

Blaise Pascal, 19 Jahre alt, franz. Mathematiker, baut eine mechanische Addiermaschine für 6-stellige Dezimalzahlen. Dies war die erste Rechenmaschine der Welt

1650

Der Rechenschieber wird erfunden



## 5.3.4 (2) Historische Entwicklung

### 1673

- G.W. von Leibnitz, Philosoph und Mathematiker entwickelt die erste Rechenmaschine für Multiplikation und Division
- Die Basis dieser Rechenmaschine wird bis heute zum Bau von rein mechanischen Maschinen verwendet
- Leibniz hat für die Datenverarbeitung insofern Bedeutung, als er schon 1679 in einer Handschrift auf die Bedeutung des Binär-(Dual-)Systems hinwies.

15.03.1679:

*„Diese Art Kalkül könnte auch mit einer Maschine ausgeführt werden.“*

## 5.3.4 (2) Historische Entwicklung

### 1801

- **Joseph-Marie Jacquard** entwickelt aus dem herkömmlichen, mehrschäftigen Webstuhl den Jacquard-Webstuhl
- Das Webmuster wird in breite Kartonbahnen eingestanz
- Durch die Stanzungen wird die „Kette“ geführt
- Die wichtigste Verbesserung von Jacquards Musterwebstuhl gegenüber all seinen Vorläufern bestand darin, dass er die Nockenwalze der österreichischen Webstühle durch das Endlosprinzip der Lochkartensteuerung ersetzte
- Dadurch konnten endlose Muster von beliebiger Komplexität mechanisch hergestellt werden
- Auf den Lochstreifen waren allerlei Informationen über das zu webende Muster enthalten

## 5.3.4 (2) Historische Entwicklung

- Die erste Programmiersprache wurde schon 1843 von einer Frau erfunden, von **Ada Lovelace**, einer britischen Mathematikerin
- Professor **Niklaus Wirth** (Eidgenössische Technische Hochschule Zürich) widmete Blaise Pascal die 1968 entworfene und 1970 mit einem ersten Compiler vorgestellte prozedurale Programmiersprache Pascal, die auf der Grundlage von ALGOL60 als eine spezielle Ausbildungssprache das Licht der Welt erblickte

1982 wurde der ISO-Standard für Pascal veröffentlicht.

Die Haupterweiterungen gegenüber ALGOL60 finden sich in der erweiterten Datenstrukturierung und der Möglichkeit, eigene Datentypen (*self-defined types*) zu deklarieren

## 5.3.4 (2) Historische Entwicklung

### 1890

Lochkarten benutzte auch Hermann Hollerith, der 1890 bei der amerikanischen Volkszählung half, den Auswertungsprozess zu automatisieren

### 1941

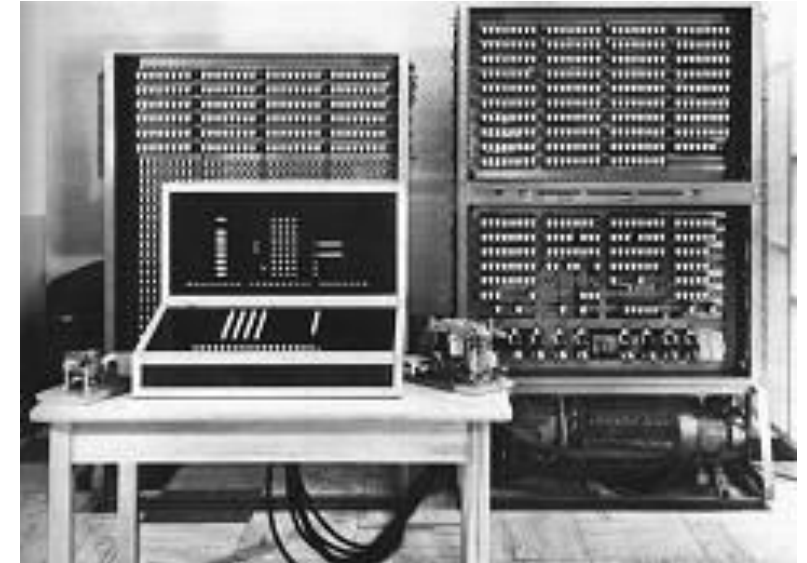
Konrad Zuses Rechner Z3 gilt als erster programmgesteuerter Relaisrechner. Die Programme wurden auf Lochstreifen abgelegt. Die Operationscodes der neun Befehle bestanden aus 8-Bit-Lochstreifen-Codierungen

Type	Instruction	Description	Opcode
I / O	Lu	read keyboard	01 110000
	Ld	display result	01 111000
Memory	Pr z	load address z	11 zzzzzz
	Ps z	store address z	10 zzzzzz
Arithmetic	Lm	multiplication	01 001000
	Li	division	01 010000
	Lw	square root	01 011000
	LS <sub>1</sub>	addition	01 100000
	LS <sub>2</sub>	subtraction	01 101000

## 5.3.4 (2) Historische Entwicklung

### Historische Entwicklung

- Wer hat's erfunden?
  - Konrad Zuse

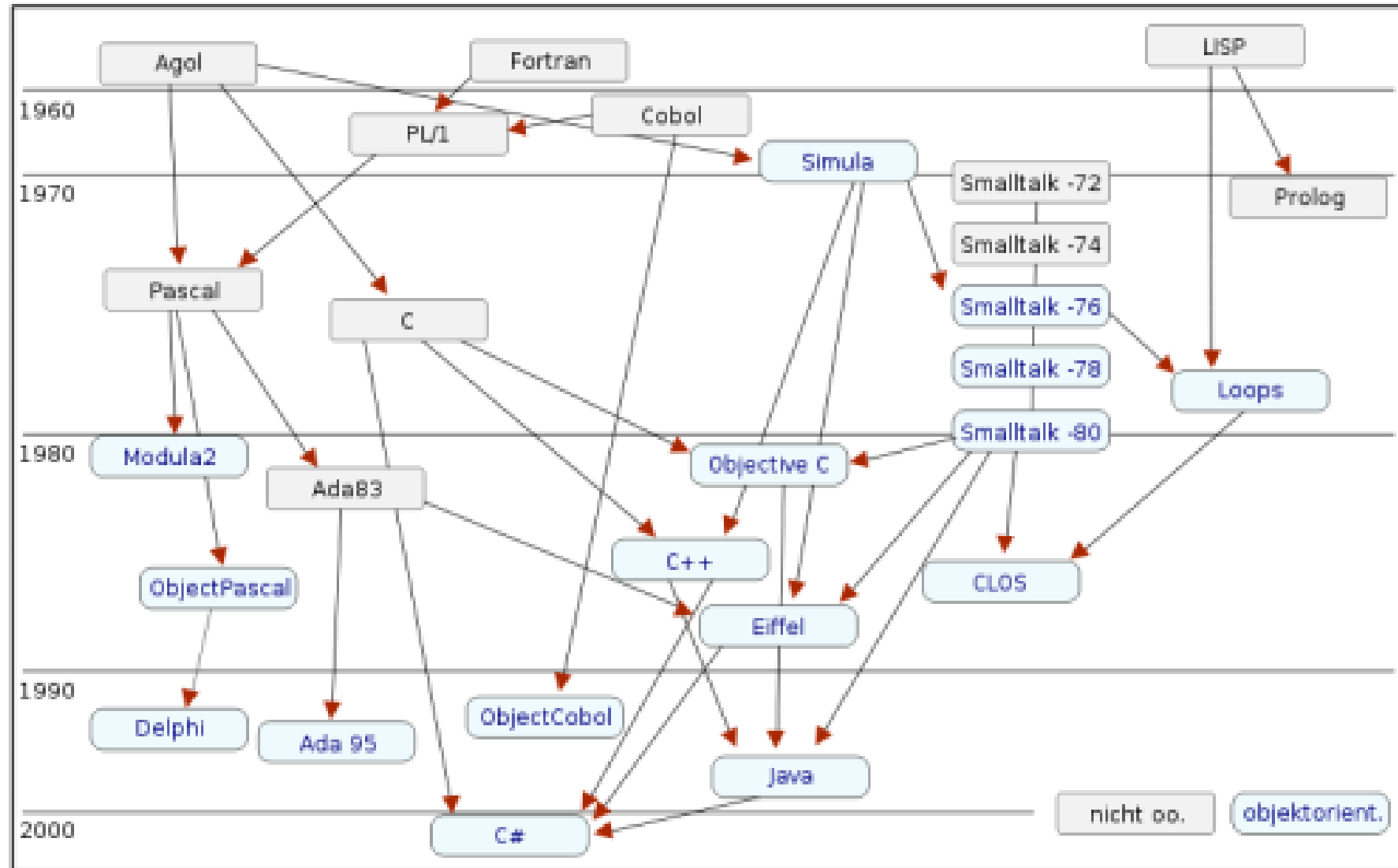


- Was kann der Computer?
  - kann alle ihm gestellten Aufgaben lösen

**Aber: Ohne unsere Hilfe ist er dumm!**



## 5.3.4 (2) Historische Entwicklung



## 5.3.4 <sup>(3)</sup> Einteilung von Programmiersprachen

### Klassifikation nach Art der Anwendung:

- Wissenschaftlich-technischer Bereich (Fortran, Java, C++)
- Kommerzieller Bereich, z. B. Banken und Verwaltung (COBOL, Java, C#)
- Systemsoftware, z. B. Betriebssysteme (C, C ++, Ada)
- Echtzeitsysteme (Ada95, Pearl)
- Künstliche Intelligenz und Expertensysteme (Haskell, Prolog, LISP)
- Telekommunikationssysteme (Chill)
- Web (Frontend: HTML, CSS, Javascript, Backend: Java, PHP, Python, Go, ...)



## 5.3.4 <sup>(3)</sup> Einteilung von Programmiersprachen

### Klassifikation nach Art der Anwendung:

- Datenbanksysteme (SQL)
- Statistische Anwendungen (SPSS, R)
- Simulationen (GPSS)
- Mathematik (Maple)
- Modellierungen (UML)
- Webseiten, z. B. Markup- und Seitenbeschreibungs-Sprachen (HTML, XML, Post script)

Hinweis: Weitere Anwendungen sind möglich.

Viele der letztgenannten Sprachen (SQL, UML etc.) sind keine Programmiersprachen

## 5.3.4 <sup>(3)</sup> Einteilung von Programmiersprachen

### Klassifikation nach dem Programmiersprachenparadigma:

- Prozedurale bzw. imperative Programmiersprachen (C)
- (Deskriptiv) Funktionale Programmiersprachen (LISP)
- (Deskriptiv) Logische Programmiersprachen (PROLOG)
- Objektorientierte Programmiersprachen (C++)

Hinweis: Es gibt Mischformen, wie z. B. Java

Java ist imperativ, objektorientiert und öffnet sich durch die Lambdas inzwischen auch der funktionalen Programmierung

## 5.3.4 <sup>(3)</sup> Einteilung von Programmiersprachen

### Klassifikation nach der „Sprachhöhe“:

- Niedere Programmiersprachen, also an den Hardwareeigenschaften des Computers orientiert (z. B. alle Assemblersprachen)
- Höhere Programmiersprachen, also an der Verstehbarkeit durch den Menschen orientiert (Fortran, ALGOL, LISP)

Hinweis: Die Grenze ist nicht eindeutig definiert

---

### Klassifikation nach Art der Notation:

- Textuelle Programmiersprachen (fast alle Programmiersprachen)
- Grafische bzw. visuelle Programmiersprachen (Scratch, Snap!)

Hinweis: Letztere werden gerne im Programmier-Anfängerunterricht verwendet

## 5.3.4 <sup>(3)</sup> Einteilung von Programmiersprachen

### Klassifikation nach dem Verbreitungsgrad:

Dies ist eine interessante Klassifikation, die unterschiedlich gefüllt werden kann. Es gibt beispielsweise den sog. TIOBE-Index des niederländischen Unternehmens TIOBE

---

### Klassifikation nach Art der Ausführung:

- Compilersprachen (C, C ++, Pascal, Fortran, Cobol, Delphi, Eiffel)
- Interpretersprachen (Basic, Perl, Python, Skriptsprachen)

Hinweis: Bei einer Compilersprache wird der Quellcode mithilfe eines Compilers in ein ausführbares Programm übersetzt. Softwarefirmen können ihren Kunden die „exe- Files “ zur Verfügung stellen, ohne den Quellcode preisgeben zu müssen

Bei Interpretersprachen gibt es keine Übersetzung des Quellcodes: Er wird während des Programmablaufs vom Interpreter in Maschinensprache umgesetzt. Die Performance im Vergleich zu Compilersprachen ist aber deutlich schlechter

## 5.3.4 <sup>(3)</sup> Einteilung von Programmiersprachen

### Klassifikation nach der Sprachengeneration:

- 1. *Generation*: Maschinensprachen  
(Maschinencode für IBM-Großrechner OS/390)
- 2. *Generation*: Assemblersprachen (8085-Assembler)
- 3. *Generation*: höhere prozedurale und objektorientierte Sprachen  
(Fortran, Basic, Pascal, C, C++, Java)
- 4. *Generation*: Tabellenkalkulation, Datenbanksprachen (Lotus, dBase, SQL)
- 5. *Generation*: Sprachen der künstlichen Intelligenz (LISP, PROLOG)
- Hinweis: Sprachen der 4. Generation gelten in der Regel nicht als Programmiersprachen

## 5.3.4 <sup>(3)</sup> Unterscheidung nach Programmierparadigma

### Imperatives, prozedurales Paradigma

Dieses orientiert sich am Algorithmusbegriff mit den drei Steuerstrukturen Folge, Verzweigung und Wiederholung. Das Programm strukturiert sich in Prozeduren (bzw. Methoden, bzw. Funktionen) und Variablen (mit entsprechenden Datentypen) und hat häufig die Reihenfolge Eingabe, Verarbeitung und Ausgabe (EVA-Prinzip).

Ein Beispiel in Pascal:

```
PROGRAM Kreisberechnung;  
USES Crt;  
CONST Pi = 3.141592654;  
VAR Umfang, Radius;  
BEGIN  
    Write ('Geben Sie den Kreisradius ein: ');  
    ReadLn (Radius);                (* Eingabe *)  
    Umfang := 2 * Radius * Pi;      (* Verarbeitung *)  
    WriteLn ('Umfang: ', Umfang);   (* Ausgabe *)  
    ReadKey;  
END.
```

## 5.3.4 <sup>(3)</sup> Unterscheidung nach Programmierparadigma

### Objektorientiertes Paradigma

Ausgehend vom Objektbegriff (z. B. kreis1 und kreis2) und der Kommunikation zwischen den Objekten wird in der abstrahierten Klasse eine semantische Einheit aus Attributen und Methoden hergestellt.

Ein Beispiel einer Java-Klasse:

```
public class Kreis {  
    private double radius;  
    public Kreis(double radius) {  
        setRadius(radius);  
    }  
    public double getRadius() {  
        return radius;  
    }  
    public void setRadius(double radius) {  
        if (radius > 0)  
            this.radius = radius;  
    }  
    public double getUmfang() {  
        return 2 * Math.PI * radius;  
    }  
}
```



## 5.3.4 <sup>(3)</sup> Unterscheidung nach Programmierparadigma

### Logisches Paradigma

Nicht der Algorithmus wird definiert, sondern das Problem wird mithilfe von Regeln bzw. Prädikatenlogik beschrieben.

Beispielsweise lässt sich in Prolog die Berechnung der Fakultät einer Zahl wie folgt beschreiben:

```
%Faktum
fakultaet(0,1).
%Regel
fakultaet(N,F):-
    N > 0,
    N1 is N-1,
    fakultaet(N1,F1),
    F is N * F1.
```

## 5.3.4 <sup>(3)</sup> Unterscheidung nach Programmierparadigma

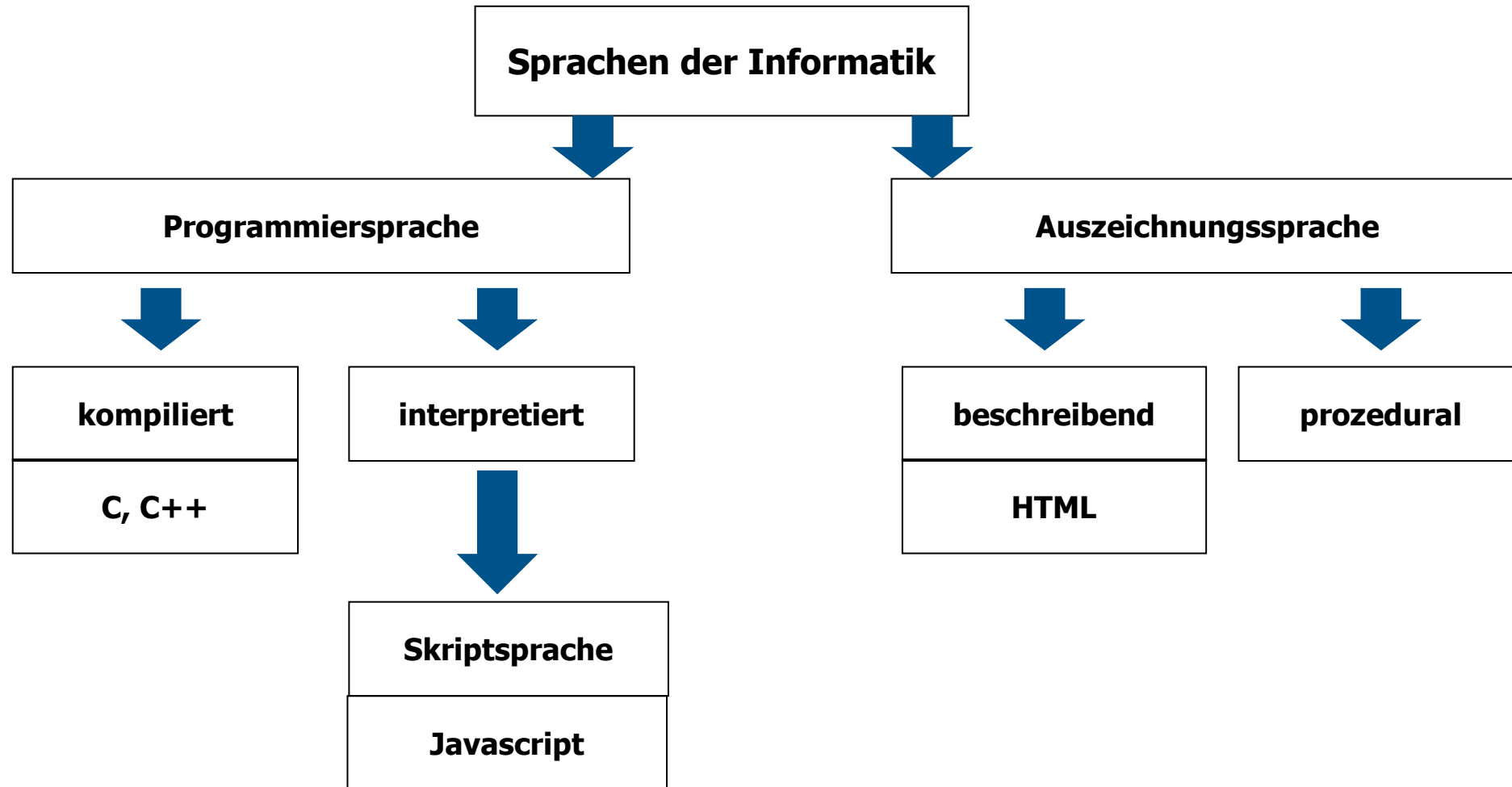
### Funktionales Paradigma

Weder Variablen noch Algorithmen werden eingesetzt. Das Programm besteht aus einer Abfolge von mathematischen Funktionen, die für bestimmte Eingangsparameter bestimmte Rückgabewerte berechnen

In Common Lisp lässt sich z. B. die Fakultät einer Zahl wie folgt iterativ berechnen:

```
(defun fact (n)
  (loop for i from 2 to n
        for fact = 1 then (* fact i)
        finally (return fact)))
```

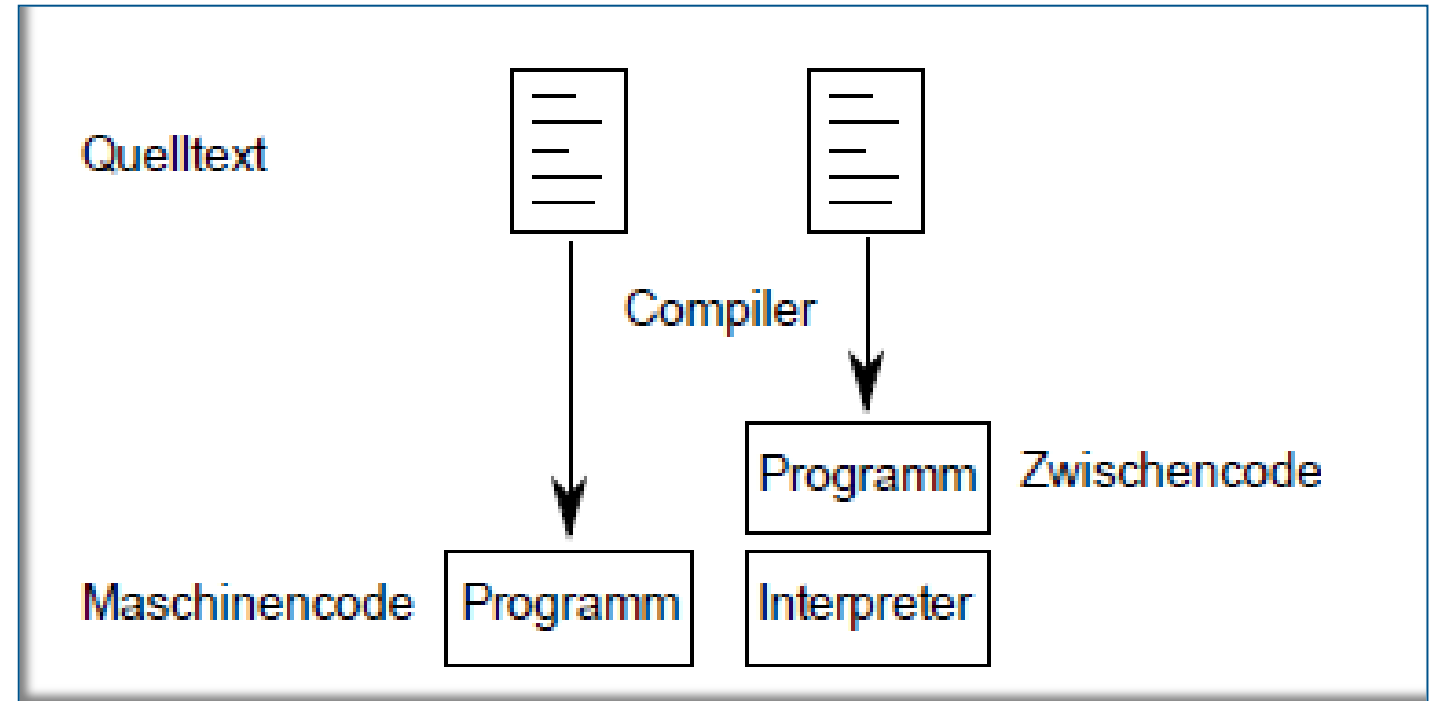
## 5.3.4 <sup>(3)</sup> Einteilung von Programmiersprachen



## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen

### Vom Quelltext zum fertigen Programm:

Je nach Programmiersprache wird der Quelltext vom Compiler direkt in Maschinencode übersetzt oder zunächst in einen Zwischencode übersetzt, welcher dann von einem Interpreter ausgeführt wird



## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen - ASSEMBLER

- Entstanden 1952
- Ein Programm, das für die Befehle, die ein Computer versteht, einen bestimmten Code hat
- Spezifisch für jeden Prozessor

Beispiel:

Hexadezimal    21 00 10 11 00 20 19 22 00 30

Für Prozessor Z80:

21 00 10 LD HL,1000H

11 00 20 LD DE,2000H

19 ADD HL,DE

22 00 30 LD (3000H),HL

25 END

- Lade den Wert 1000H in das Prozessorregister "HL"
- Lade den Wert 2000H in das Prozessorregister "DE"
- Addiere die beiden Register HL und DE, das Ergebnis ist danach in HL
- Speichere das Ergebnis in der Speicherstelle 3000H im Speicher ab.

## 5.3.4 (4) Aufbau von Programmiersprachen - ASSEMBLER

- Z80 war schnell
  - Namen für Speicherzellen vergebenbar
  - Adressen für Sprünge berechenbar
- Speicherkapazität und Rechenleistung maximiert

Aber:

- Testen der Programme war aufwändig
- Keine einheitliche Sprache, spezifisch für jeden Prozessor

Hallo Welt in Z80 Assembler:

```
bdos      org      100H
          equ      00005h

start:    ld        de,hallowelt
          ld        c,9
          call     bdos
          rst      0

hallowelt: defs     "Hallo Welt!",13,10,"$"
          end      start
```

## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen - Fortran

- Entstanden 1953 – 1954 unter Leitung von John Backus
- **Formula Translator**, wissenschaftlich/mathematische Programmiersprache
- Nur Großbuchstaben waren zulässig
- Variablennamen durften nur max. 6 Zeichen haben
- Variablen konnten ein Leerzeichen enthalten
- Eine Zeile hatte eine feste Struktur
- Programmtext dürfte z. B. nur in den Spalten 7 - 72 einer Zeile vorhanden sein, Sprung Labels nur in den Stellen 1 - 5



Grund → Lochkartenstanzung

Oben Text 80 spaltig, unten gestanzte Löcher

→ Größere Programme wurden „schwer“



## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen - Fortran

- Geeignet zum Berechnen von Dingen
- Ungeeignet Texte zu verarbeiten
- Keine strukturierte Programmierung, stattdessen GOTOS
- 1966 erster Standard FORTRAN V, FORTRAN 66
- 1977 Standardisierung mit strukturierten Elementen (kein GOTO)
- 1990 Abkehr vom starren Lochkarten-Konzept,
  - Groß- und Kleinschreibung,
  - Prozeduren,
  - Namen anstatt Zeilennummern und
  - CASE Abfragen
- Backus-Naur Form, zur Beschreibung der Grammatik von höheren Sprachen

```
PROGRAM  
  
PRINT *, "Hello World"  
END PROGRAM
```

## 5.3.4 <sup>(4)</sup> **Aufbau von Programmiersprachen - Cobol**

- Erster Entwurf 1960
- **C**ommon **B**usiness **o**riented **L**anguage
- Leicht lesbare Sprache
- Verarbeitung von Daten, Texten, Buchungen
- COBOL kann z. B. sehr einfach Zahlen formatieren, ausgeben, sortieren
- Sehr textlastig
- Mathematische Operationen haben Namen  
(MOVE ZERO TO XYZ; MULTIPLY XYZ BY 5)
- Keine strukturierte Programmierung (GOTO)
- 1968 erneuter Standard verabschiedet

## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen - Cobol

- COBOL 74 und 85 erweiterten die Sprache
- Noch heute im Bankenbereich anzutreffen

```
00001  IDENTICAL DIVISION.  
00002  PROGRAM-ID HELLO.  
00004  ENVIROMENT DIVISION.  
00005  CONFIGURATION SECTION.  
00006  SOURCE-COMPUTER IBM-PC.  
00008  OBJECT-COMPUTER IBM-PC.  
00010  INPUT-OUTPUT SECTION.  
00011  FILE-CONTROL.  
00012  SELECT AUSGABE ASSIGN TO SI.  
00013  DATA DIVISION.  
00014  FILE-SECTION.  
00015  FD AUSGABE  
00016  LABEL RECORD OMITTED.  
00017  01 DATA RECORD ZEILE PICTURE X(80).  
00018  WORKING STORAGE SECTION.  
00019  PROCEDURE DIVISION  
00020  BEGIN.  
00021  OPEN OUTPUT AUSGABE.  
00022  MOVE "HELLO WORLD!" TO ZEILE.  
00023  CLOSE AUSGABE
```

## 5.3.4 (4) Aufbau von Programmiersprachen - Basic

- J. Kemeny und T. Kurtz, 1964
- **B**eginners **A**ll-purpose Symbolic **I**nstruction **C**ode
- Einfach zu erlernen und kompakt, nicht strukturiert (GOTO)
- Setzte sich durch die Heimcomputer durch
- Heimcomputer hatten in den achtziger Jahren 16 - 64 KByte Speicher.  
Einen einfachen BASIC Interpreter bekam man in einem 8 KByte großen ROM unter, einen komfortablen in 16 - 32 KByte, das waren Systemanforderungen, die von keiner anderen Programmiersprache erreicht werden konnten
- Mit dem Aussterben der Heimcomputer 15 Jahre später versank auch BASIC in der Versenkung. Einzig allein Microsoft entwickelte die Sprache weiter, hat allerdings auch jetzt vor, die Entwicklung zugunsten von C# einschlafen zu lassen

```
10 PRINT "Hello World"  
20 END
```

## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen - Pascal

- Erste Version 1970, Nikolaus Wirth
- Daten von Code getrennt
- Blöcke waren durch Begin und end gekennzeichnet
- Leicht lesbar
- Basis war ALGOL (Algorithmic Language, FOR, WHILE, REPEAT-Schleifen)
- String-Routinen
- Strukturierung der Daten
- Außer Zeichen, Ganzzahlen oder Fließkommazahlen zusammengesetzte Daten (aus verschiedenen einfachen Datentypen), Teilbereiche (Buchstaben nur von A bis Z) oder Mengen (Herz, Karo, Pik, Kreuz) definieren und sich damit näher an der abstrakten Umwelt bewegen
- Bei Aufruf von Unterprogrammen, konnten diese eigene lokale Variablen verwenden, die wieder freigegeben wurden (Stack) (Stapelmaschine, später PDP-11)

## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen - Pascal

- Erfindung der virtuellen Maschine
- Übersetzung in Byte-Code → maschinenspezifische Übersetzung in Maschinencode
- Fehlt:
  - Strings haben eine feste Länge und sind auf 255 Zeichen beschränkt - Textverarbeitung damit zu machen, gestaltet sich schwierig
  - Es gibt keine Möglichkeit, maschinennah zu programmieren - dies haben die meisten Hersteller nachgerüstet, aber es ist dann eben nicht mehr standardisiert
- Im PC Bereich ist aus dem guten alten Pascal "Delphi" geworden, mit den besten Elementen aus Modula, C++ und Smalltalk

```
Program HelloWorld;  
  
begin  
    writeln('Hello world!');  
end.
```

## 5.3.4 (4) Aufbau von Programmiersprachen - C

- C - Kerningham und Denis Ritchie
- Entwicklung in den Sechzigern aus der Sprache BCPL über den Zwischenschritt "B" die Sprache "C". Als Ersatz für Assembler
- Sehr hardwarenah
- Mangelnde Syntax und Typprüfungen
- Kein modulares Design
- 1. Standard 1978, ANSI 1989, ANSI 1999
- UNIX war kostenlos → Durchsetzen von C
- Systemaufrufe waren auf die Datentypen und Funktionsweise von C ausgerichtet und andere Programmiersprachen mussten diese erst nachbilden
- Die Syntax wurde "weitervererbt" an C++, Java, C#, PHP, JavaScript ...

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen - Smalltalk

- 1969 schuf die durch ihre Fotokopierer reich gewordene Firma Xerox den Xerox PARC bei Palo Alto(Forschungsinstitut)
- Grafische Benutzeroberfläche, die Apple für den MAC lizenzierte und Microsoft kopierte
- 1969 - 1970 aber auch Smalltalk. Smalltalk ist eine Programmiersprache, die völlig objektorientiert ist
- Alles ist ein Objekt
- Es gibt keine einfachen Datentypen
- Der Typ einer Variable kann geändert werden
- Kontrollstrukturen sind Objekte
- Alles agiert mit Hilfe von Botschaften

Der Name entstand dadurch, dass Programme in Smalltalk sehr wenig Code benötigen, man also "nicht viel reden" muss



## 5.3.4 (4) Aufbau von Programmiersprachen - Smalltalk



- 1. Standard 1980, mit Smalltalk 80
- 1983 letzten Feinschliff
- Smalltalk erforderte eine grafische Oberfläche mit Fenstern für die Klassendefinitionen, den Debugger und das Transcript-Fenster, in dem man Smalltalk Programme startete oder einfach Ausdrücke auswertete
- In Zeiten, in denen Programmierung bedeutete: Editieren -> Programm speichern -> Compiler starten -> Linker starten -> Programmausführen, jeweils mit unterschiedlichen Programmen von der Kommandozeile aus, war dies enorm progressiv

```
Transcript show: 'Hello World!'; cr.
```

## 5.3.4 (4) Aufbau von Programmiersprachen - C++



- 1982/3 machte sich Bjarne Stroustrup daran, auf Grundlage von C eine objektorientierte Erweiterung "C mit Klassen" zu entwickeln
- 1987 → C++

Aber:

Es blieb die unlesbare Syntax und die mangelhafte Typprüfung

- Die Syntax ist so komplex, es gibt so viele Möglichkeiten, mit Sprachmitteln ein und dasselbe zu erreichen

```
#include <iostream.h>

class Hello {
public:
    Hello ()
    {
        cout << "Hello world!" << endl;
    }
};

int main()
{
    Hello();
    return 0;
}
```

## 5.3.4 <sup>(4)</sup> Aufbau von Programmiersprachen - Java



- 1991 bei SUN, Entwicklung von Java (Name des Lieblingskaffees der Entwickler)
- Baut auf C auf
- Ist rein objektorientiert, Prozeduren-Elemente von C entfielen
- Basiert auf einer virtuellen Maschine

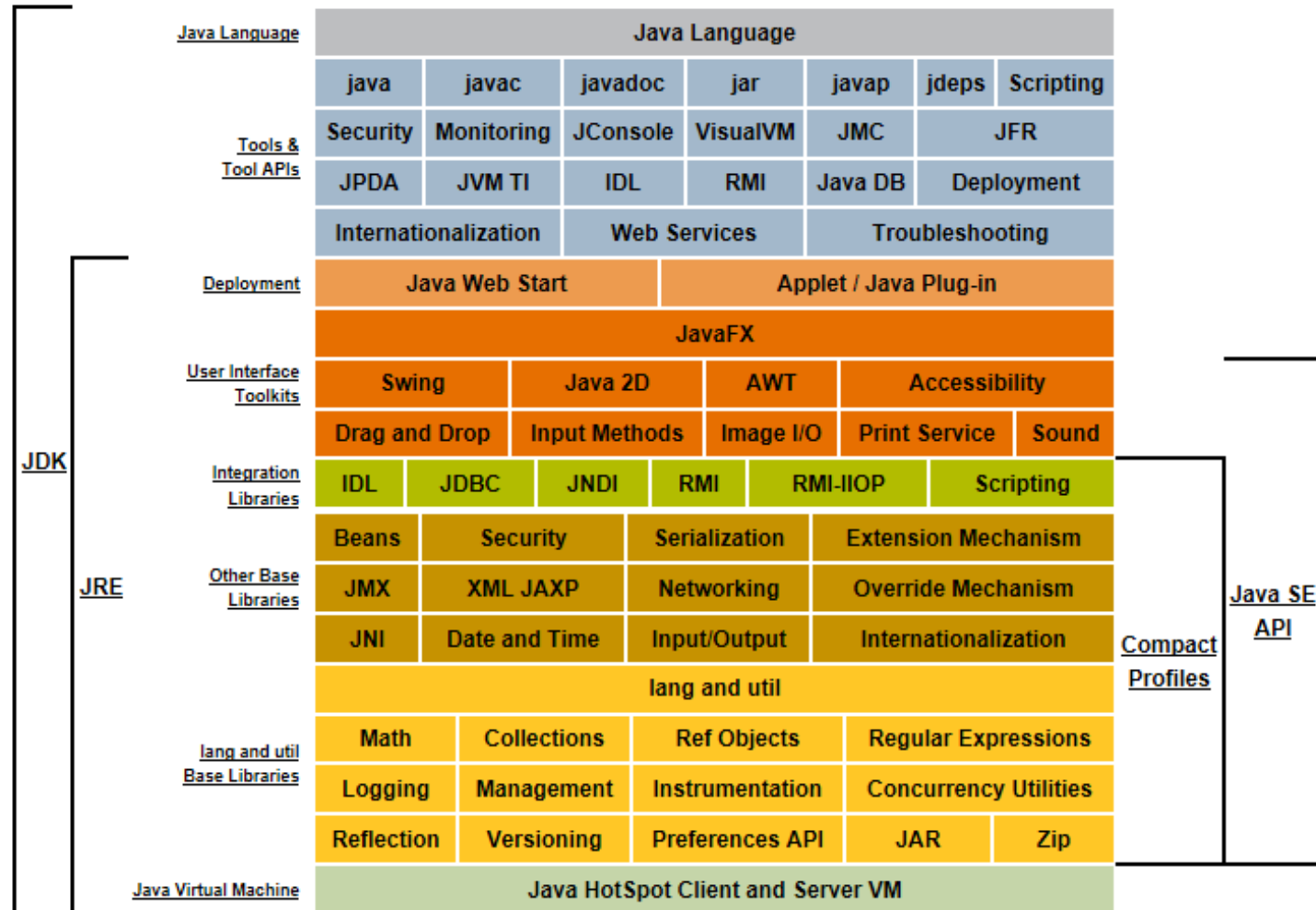
Vorteile:

- Plattformunabhängig
- greift nicht auf die Ressourcen des Computers zu
- Kann auf verteilten Systemen laufen
- Beinhaltet Routinen für grafische Oberflächen, die auf unterschiedlichen Systemen laufen (Mac OS, Windows, X11 UNIX)
- 1995 erste Version

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

## 5.3.4 (4) Aufbau von Programmiersprachen - Java

Description of Java Conceptual Diagram



## 5.3.4 <sup>(4)</sup> Auswahl einer Programmiersprache - Python

- **Python** (['pʰaɪθn], ['pʰaɪθɒn], auf Deutsch auch ['pʰy:tn])
- Die Sprache wurde Anfang der 1990er Jahre von Guido van Rossum am Centrum Wiskunde & Informatica in Amsterdam als Nachfolger für die Programmier-Lehrsprache ABC entwickelt
- Der Name geht nicht, wie das Logo vermuten ließe, auf die gleichnamige Schlangengattung der Pythons zurück, sondern bezog sich ursprünglich auf die britische Komikertruppe **Monty Python**.  
In der Dokumentation finden sich daher auch einige Anspielungen auf Sketche aus dem **Flying Circus**



## 5.3.4 <sup>(4)</sup> Auswahl einer Programmiersprache - Python

- Python ist eine universelle, üblicherweise interpretierte, höhere Programmiersprache
- Sie hat den Anspruch, einen gut lesbaren, knappen Programmierstil zu fördern. So werden beispielsweise Blöcke nicht durch geschweifte Klammern, sondern durch Einrückungen strukturiert
- Aktuelle Version: 3.9.0 (5. Oktober 2020)

```
def fact(x):  
    return 1 if x <= 1 else x * fact(x - 1)
```

## 5.3.4 <sup>(5)</sup> Frameworks

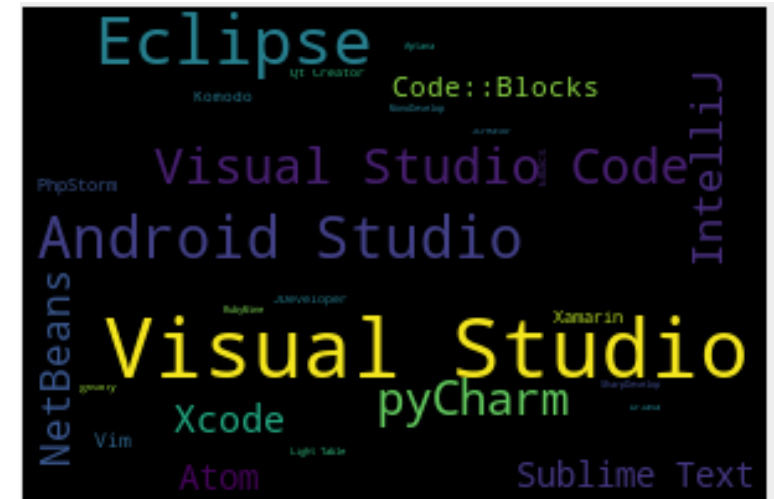
Unterstützen den Entwickler, durch fertige Funktionalitäten ...

- Grundgerüst für eine spätere Softwarelösung
- Stellt eine Struktur zur Verfügung, die von sämtlichen Anwendungen gleichermaßen genutzt werden kann
- Einsatzbereiche
  - Application Frameworks
  - Domain Frameworks
  - Test Frameworks
  - Web Frameworks



## 5.3.4 (6) Programmierwerkzeuge

- Unterstützen die Entwicklung von Programmen
- Entwicklungsumgebung (IDE : **I**ntegrated **D**evelopment **E**nvironment)
- Bei umfangreichen Programmen unverzichtbar
- Einheitliche Oberfläche
- In einer IDE sind enthalten:
  - Texteditor mit Syntaxhervorhebung
  - Compiler oder Interpreter, Debugger
  - Projekt- und Versionsverwaltung
  - Durch Plugin-Schnittstellen lassen sich die meisten Entwicklungsumgebungen auch um weitere Funktionalitäten erweitern

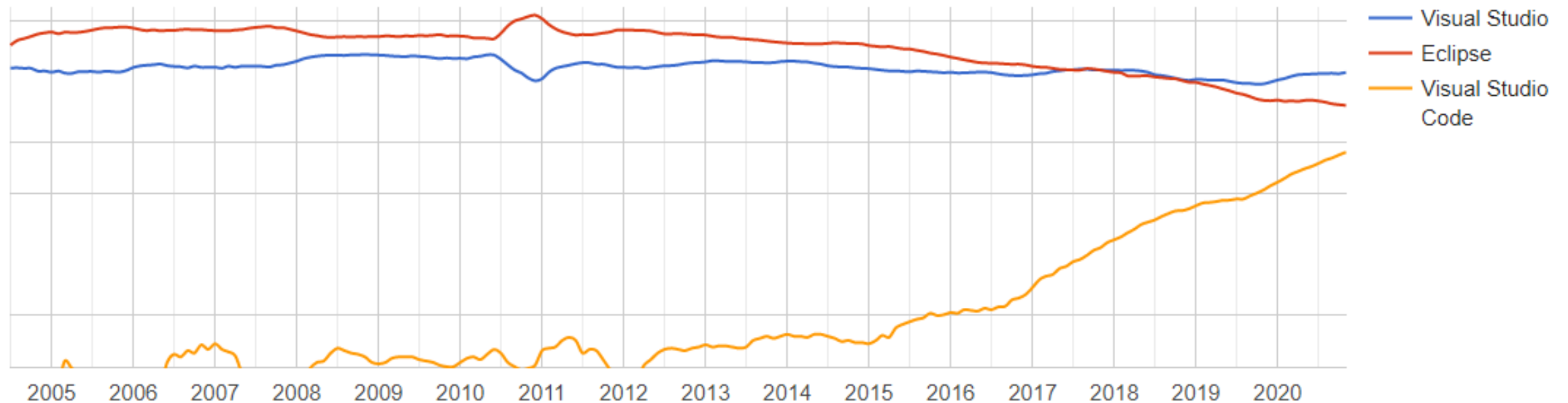




## 5.3.4 (6) Programmierwerkzeuge



Top IDE index



IDE Index, vollständiges Ranking unter [pypl.github.io/IDE](https://pypl.github.io/IDE)

## 5.3.4 <sup>(6)</sup> Programmierwerkzeuge

Wichtige Programmierwerkzeuge	
Texteditor	Zur Eingabe und Änderung des Quelltexts von Programmen Syntaxhervorhebung (Syntax-Highlighting) für eine bessere Lesbarkeit des Quelltextes
GUI-Editor	Erleichtert das Design von grafischen Benutzeroberflächen Anordnen von Steuerelementen per Drag-and-Drop- Unterstützung der Regeln für das Programmieren von Events
Compiler Interpreter Assembler	Die Hauptaufgabe dieser Werkzeuge ist die Übersetzung des Quellcodes in ausführbare Maschinensprache. Es gibt für viele Programmiersprachen sowohl Compiler als auch Interpreter oder eine Kombination aus beiden
Debugger	Der Debugger (von engl. „Entlausen“, Bug = Schädling, Fehlfunktion) unterstützt den Entwickler bei der Fehlersuche. Mithilfe des Debuggers kann das Programm Schritt für Schritt ausgeführt und die einzelnen Programmzustände untersucht werden
Versions- verwaltung	Speicherung und Verwaltung von Dateien und Dokumenten, welche bei der Entwicklung von Programmen erstellt werden

## 5.3.4 <sup>(6)</sup> Programmierwerkzeuge

- **Compiler**

Ein Compiler übersetzt den Quelltext von einem Programm in einen Maschinencode, ohne jedoch die Befehle auszuführen.

Die Programmiersprache, aus der die übersetzten Programme stammen, wird als Quellsprache bezeichnet. Dementsprechend nennt man das zu übersetzende Programm Quelltext

- **Interpreter**

Ein Interpreter übersetzt während der Laufzeit eines Programms den vorhandenen Quelltext Anweisung für Anweisung in eine maschinenverständliche Form , d. h., er bearbeitet ein Programm zeilenweise, wobei jede Zeile einer Syntaxprüfung unterzogen wird



Welche Aussage ist richtig?

- a) Es gibt nur sehr wenige Programmiersprachen.
- b) Programmiersprachen besitzen eine Syntax und Schlüsselwörter.
- c) Es gibt imperative und deklarative Programmiersprachen.
- d) Die Auswahl einer Programmiersprache für eine Programmieraufgabe ist in aller Regel unabhängig vom jeweiligen Aufgabengebiet.
- e) Um die Popularität der Programmiersprachen zu messen, gibt es Indizes wie RedMonk oder den TIOBE-Index.
- f) Ein Framework stellt dem Programmierer verschiedene Bibliotheken und Module für bestimmte Programmieraufgaben zur Verfügung und erleichtert somit die Programmierarbeit.



Welche Aussage ist richtig?

- g) C ist eine neue Programmiersprache.
- h) Python ist leicht zu erlernen und eignet sich sehr gut für den Einstieg in die Programmierung.
- i) Eine IDE ist ein spezieller Texteditor.
- j) Ein Debugger findet logische Fehler in einem Programm.
- k) Interpreter übersetzen den Quelltext bei jedem Programmstart neu.
- l) Kompilierte Programme sind in der Regel langsamer in der Programmausführung als interpretierte Programme.

# Zusammenfassung – Den Prozess der Softwareentwicklung analysieren



IT-Berufe  
Grundstufe 1-5

Westermann  
**Kapitel 5.3**