# Einleitung

Bernhard Westfechtel
Angewandte Informatik I
Universität Bayreuth

# Gliederung

- Motivation
- Software Engineering
- Qualitätsanforderungen
- Software-Prozessmodelle
- Ziele und Überblick

# Motivation

# Die „Studentensicht" auf die Softwareentwicklung

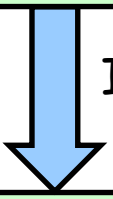
- Softwareentwicklung = Programmierung = Lösung eines kleines und wohldefinierten Problems
- Problemdefinition = Übungsaufgabe (oder Aufgabe im Praktikum)
- Programm wird von einem einzigen Programmierer entwickelt
- Problem kann in begrenzter Zeit gelöst werden (1-2 Stunden)
- Beherrscht werden müssen:
  - Programmiersprache
  - Programmierwerkzeuge (Editor, Compiler, Debugger)
  - Entwurf und Codierung von Algorithmen

# Softwareentwicklung in den 60er Jahren

- Programmieren ist eine Kunst (s. z.B. Donald Knuth´s Buchserie *The Art of Computer Programming*)
- Leistung der Hardware stark beschränkt ⇒
  Optimierung sehr wichtig (Speicherplatz, Laufzeit)
- Softwareentwicklung: Entwurf und Codierung von Algorithmen
- Programmierer wurden für ihre intellektuellen Fähigkeiten und die Anwendung von Programmiertricks bewundert
- Ein brillantes Programm ist ein Programm, das höchstens der Autor versteht (für eine begrenzte Zeit) ⇒
  Die „goldene Regel" der Programmoptimierung:
  *In (fast) jedem Programm kann man mindestens einen Befehl einsparen.*

# Beispiel (in C)

```
void strcopy (char *s, char *t) /* copy from t to s */
{
    while (*t != '\0') {
        *s = *t;
        s++;
        t++;
    }
}
```

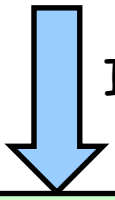


In der Bedingung kopieren

```
void strcopy (char *s, char *t) /* copy from t to s */
{
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
}
```

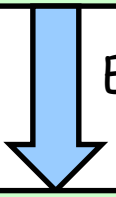
[Kernighan 1990]

# Beispiel (in C)



In der Bedingung inkrementieren

```
void strcopy (char *s, char *t) /* copy from t to s */
{
    while ((*s++ = *t++) != '\0')
        ;
}
```



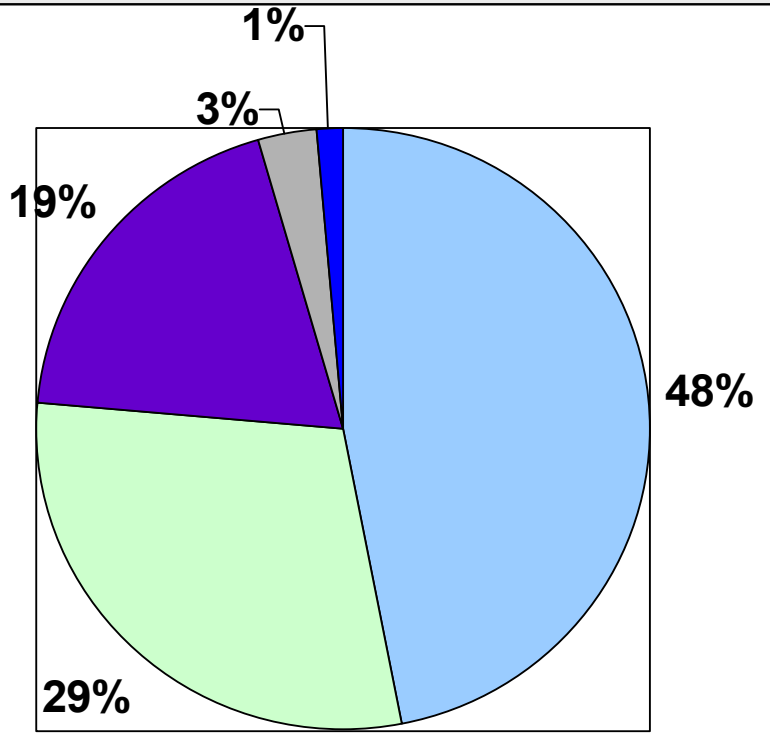
Explizite Abfrage eliminieren

```
void strcopy (char *s, char *t) /* copy from t to s */
{
    while (*s++ = *t++)
        ;
}
```

# Die Softwarekrise (entdeckt in den späten 60er Jahren)

- Software wird immer größer
- Zu lösende Probleme werden immer anspruchsvoller
- Methoden zur Softwareentwicklung sind nicht skalierbar
- Systematische Entwicklungsmethoden werden dringend gebraucht
- Softwareentwicklung sollte als Wissenschaft und nicht als Kunst betrachtet werden
- Symptome der Krise:
  - Viele Softwareprojekte werden abgebrochen
  - Viele Softwareprojekte sind „dead on delivery"
  - Erhebliche Zeit- und Kostenüberschreitungen

# Typische Statistik




(US Government Accounting Office, Statistik von 9 Projekten, 1979, Prozentzahl bezieht sich auf Projektvolumen Mio US $)

# Die Realität industrieller Softwareentwicklung

- Softwaresysteme sind sehr groß
  - SAP-R3 (Standardsoftware für betriebswirtschaftliche Anwendungen) besteht aus mehreren Megazeilen ($10^6$) Code
  - AXE-10 (Vermittlungssystem von Ericsson) umfasst etwa 10 Megazeilen
- Softwaresysteme leben sehr lange
  - BS 2000 Betriebssystem (Siemens) wird seit den 70er Jahren benutzt
- Softwaresysteme werden von sehr großen Teams entwickelt
  - Mehrere Tausend Entwickler arbeiten an SAP-R3
- Anforderungen sind ständigen Änderungen unterworfen
  - Änderungen werden nicht immer nachgezogen (z.B. Ariane, s.u.)
- Alle Softwaresysteme enthalten Fehler
- Trotzdem ist Korrektheit (und Zuverlässigkeit) absolut essenziell, z.B.
  - Telekommunikationssysteme (Ausfallzeit: wenige Minuten pro Jahr)
  - Raumfahrt-Steuerungssoftware (potenzieller Verlust von Milliarden €)

# Prominente Katastrophen (1)

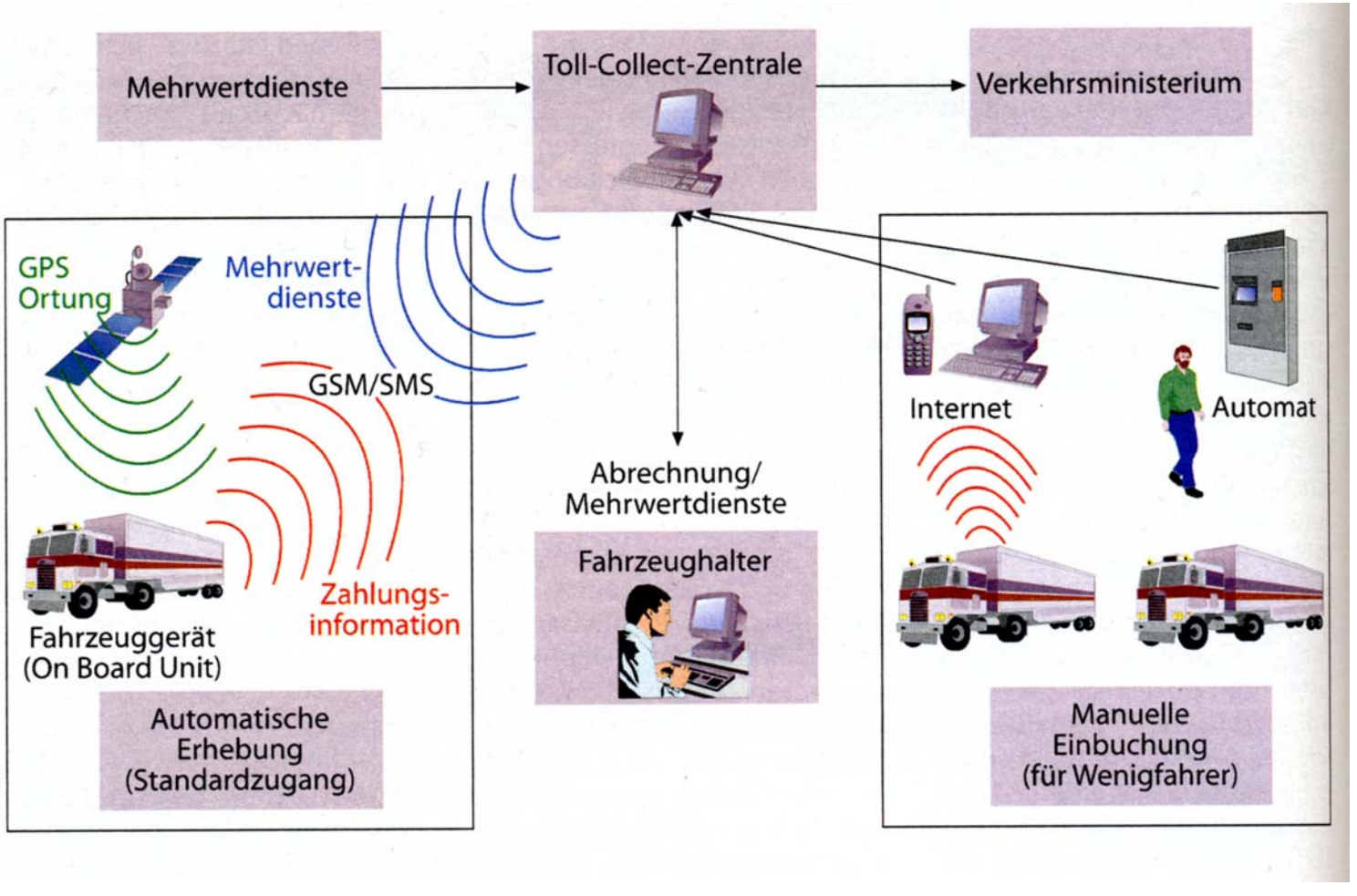
- Europäische Mars-Sonde: verloren wegen inkonsistenter Annahmen bzgl. Längenmaßen in verschiedenen Teilsystemen (Zoll vs. cm)
- Flughafen Denver: Inbetriebnahme mehr als ein Jahr verzögert durch Fehler in der Steuerung der Gepäckbeförderung
- Flugkontrolle Maastricht: Neues Kontrollsystem konnte mehrere Jahre wegen Software-Fehlern nicht in Betrieb genommen werden
- Ariane 5: Rakete wegen nicht abgefangener Ausnahme abgestürzt
  - Vorgänger Ariane 4 war langsamer (kleinere Beschleunigung)
  - Eine Minute nach dem Start war die Beschleunigung wesentlich höher als bei Ariane 4 $\Rightarrow$
    Überlauf einer Variablen zur Speicherung der Beschleunigung
  - Rakete kam vom Kurs ab
  - Rakete wurde von der Bodenkontrolle automatisch zerstört
  - Schaden: 500 000 000 US $

# Prominente Katastrophen (2)

- Toll Collect
  - Lkw-Maut mit prognostizierten 3,4 Milliarden € pro Jahr
  - Geplante Inbetriebnahme: 31. August 2003
  - Software-Probleme im Zusammenspiel Onboard Units – GPS
  - Tatsächliche Inbetriebnahme: 1. Januar 2005
    ⇒ Mautausfälle ca. 4,8 Milliarden €
- Fiscus
  - Föderales integriertes standardisiertes computergestütztes Steuersystem
  - Bund-Länder-Projekt ⇒ 17 Auftraggeber
  - 1,6 Millionen Codezeilen
  - 50 000 Seiten Dokumentation
  - 13 Jahre Entwicklungszeit
  - 170 Millionen € Kosten prognostiziert
  - 900 Millionen € tatsächliche Kosten
  - Ergebnislos abgebrochen

[Krempl 2004]

# Toll Collect



# SkyTrain am Flughafen Düsseldorf

Die vollautomatische Kabinenbahn SkyTrain, die den neuen Terminal mit dem Bahnhof des Düsseldorfer Flughafens verbindet, wurde nach sechs Pannen in sechs Tagen vorläufig gestoppt FOTO: DPA/TSCHAUNER

## Düsseldorfer SkyTrain wird für Siemens zur teuren Hängepartie

**Düsseldorf** – Die Zukunft des SkyTrain am Düsseldorfer Flughafen wird immer mehr zur Hängepartie. Die vollautomatische Kabinenbahn, die zwischen dem Terminalgebäude und dem Flughafenbahnhof schweben sollte, wurde nach sechs Pannen in sechs Tagen vorläufig gestoppt. Frühestens am kommenden Montag soll die Bahn aus dem Hause Siemens wieder seine 2,5 Kilometer lange Fahrt aufnehmen. Ein Sprecher des Flughafens schloss allerdings nicht aus, dass der SkyTrain noch länger stehen bleiben könnte.

Am 1. Juli, noch vor dem Start der Hauptreisezeit, ging die 115 Mio. Euro teure Schwebebahn an den Start.

Doch die Ernüchterung folgte schnell. Allein am Freitag hingen nach einer Störung zahlreiche Reisende 40 Minuten in zehn Metern Höhe, bis der fahrerlose Pannenzug abgeschleppt und sie befreit wurden. „Die Häufung von Problemen ist nicht akzeptabel", so der Flughafen-Sprecher.

Siemens machte Fehler in der Hardware des Steuerungssystems als Grund für den unfreiwilligen Zwischenstopp aus. Doch da hatte Flughafen-Chef Rainer Schwarz längst die Notbremse gezogen. Bis Freitag soll der Münchener Konzern nachweisen, dass die Probleme erkannt, behoben und neuerliche Betriebsausfälle weit gehend ausgeschlossen sind. Dann will Schwarz auch entscheiden, ob der SkyTrain ab Montag wieder fahren darf oder weiter nutzlos herumhängt.

Für Siemens ist diese Angelegenheit peinlich und teuer. Nachdem der Technologiekonzern bereits eine Vertragsstrafe zahlen musste, weil die Kabinenbahn aufgrund von Software-Problemen erst mit einjähriger Verspätung gestartet werden konnte, kommen nun die Ausfallkosten hinzu. Siemens werden sämtliche Aufwendungen, die durch den Ersatzverkehr mit sechs Bussen plus dem notwendigen Personal entstehen, in Rechnung gestellt. *Da.*

Geplante Inbetriebnahme August 2001

⬇

Eröffnung durch Ministerpräsident 1. Juli 2002

⬇

Vorübergehende Schließung 12. Juli 2002

DIE WELT 16.07.2002

# Schlussfolgerungen

- Softwareentwicklung sollte nicht als Kunst betrachtet, sondern als Wissenschaft behandelt werden
- Softwaresysteme sollten ingenieurmäßig entwickelt werden
- Software Engineering ist sehr viel mehr als nur Programmierung
  - Anforderungen müssen definiert und mit dem Kunden verhandelt werden
  - Softwareprojekte sind von erheblicher Größe und Dauer und müssen geplant und kontrolliert werden
  - Softwaresysteme müssen adäquat dokumentiert werden, so dass sie später von anderen geändert werden können
  - Außer dem „Programmieren im Kleinen" braucht man auch „Programmieren im Großen"
- Software Engineering bezieht sich nicht nur auf die Erstentwicklung
  - Die Wartung (nach der Auslieferung) nimmt einen signifikanten Anteil an und übersteigt im Aufwand oft die Entwicklung erheblich

# Software Engineering

# Die Geburt des Software Engineering

- Die Softwarekrise in den 60ern motivierte die Disziplin Software Engineering
- Der Begriff „Software Engineering" wurde von F.L. Bauer in einer Studiengruppe der NATO geprägt
- Kick-off: NATO Working Conference on Software Engineering, Garmisch-Partenkirchen, 7.-10. Oktober 1968
- Ziel: Software Engineering als Ingenieurdisziplin ⇒ Konzepte aus anderen Disziplinen kopieren (z.B. Maschinenbau oder Elektrotechnik)
- Software Engineering wird meist der Praktischen Informatik zugeordnet
- Abgrenzung gegenüber anderen Gebieten der Informatik schwierig
  - Überall wird Software Engineering gebraucht
  - Domänenspezifisches Software Enginering

# Definitionen

- Software
  - *Computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system* [ANSI 1983]
- Software Engineering
  - *The systematic approach to the development, operation, and maintenance of software* [ANSI 1983]
  - *An engineering discipline which is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone to use* [Sommerville 2001]
  - *Zielorientierte Bereitstellung und systematische Anwendung von Prinzipien, Methoden und Werkzeugen für die kooperative, ingenieurmäßige Entwicklung und Nutzung großer Softwaresysteme* [Balzert 2000]

# Definitionen

- Prinzip
  - *Grundsatz, den man seinem Handeln zugrunde legt*
  - Beispiele: Abstraktion, Hierarchisierung, Strukturierung
- Methode
  - *Planmäßig angewandte, begründete Vorgehensweise zur Erreichung von festgelegten Zielen (im Rahmen festgelegter Prinzipien)*
  - Beispiel: objekt-orientierte Methoden zur Definition von Daten mit Klassendiagrammen
- Verfahren
  - *Ausführbare Vorschrift oder Anweisung zum gezielten Einsatz von methodischen Vorgehensweisen*
  - Beispiel: Berechnung von kritischen Pfaden in Projektplänen
- Werkzeug
  - *Softwaresystem, das Schritte der Softwarenentwicklung gemäß einer bestimmte Methode oder eines bestimmten Verfahrens unterstützt oder automatisiert*
  - Beispiele: Projektmanagementsystem, CASE-Werkzeug (Computer-Aided Software Engineering)

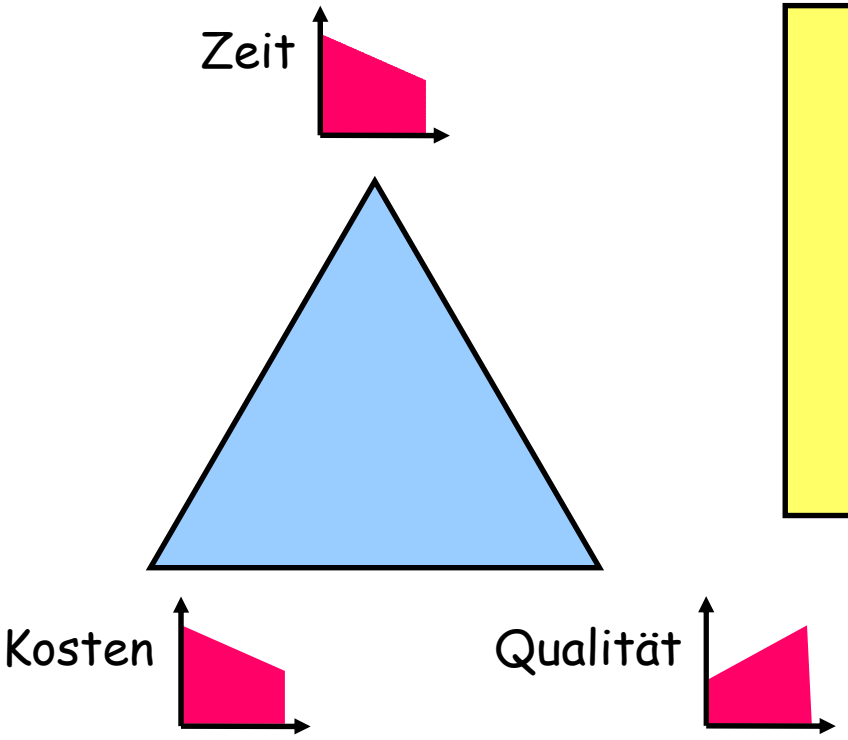
[Balzert 2000]

# Was macht Software Engineering schwierig?

- Das Softwareprodukt ist „virtuell" (nicht greifbar)
- Man alle denkbaren konzeptionellen Objekte für die Modellierung von Software einsetzen (ggf. ohne Bezug zu physischen Objekten)
- Ein hoher Grad von Abstraktion geht einher mit einem niedrigen Grad an Standardisierung
- Es ist schwierig, den tatsächlichen Fortschritt objektiv zu messen („Die Programmierung ist zu 90 % abgeschlossen.")
- Softwareentwicklung ist inhärent nichtdeterministisch (immer für eine Überraschung gut)
- Der Softwareprozess ist noch nicht hinreichend verstanden
- Softwareprodukte sind einzigartig (Unikate)
- Entwickler sind nicht austauschbar

# Qualitätsanforderungen

# Ziel des Software Engineering




Durch einen wohldefinierten und kontrollierten Prozess dazu beitragen,

- Entwicklungszeiten zu verkürzen
- Entwicklungskosten zu verringern
- Produktqualität zu erhöhen

# Softwarequalität: Definitionen

- Softwarequalität
  - *Alle Eigenschaften eines Softwaresystems, die sich auf seine Fähigkeit beziehen, definierte oder implizit angenommene Anforderungen zu erfüllen*
- Qualitätsmerkmal
  - *Ein Bündel von verwandten Eigenschaften eines Softwaresystems, die benutzt werden, um Qualität zu beschreiben und zu bewerten*
- Qualitätsindikator
  - *Eine Eigenschaft eines Softwaresystems, die relevant für ein Qualitätsmerkmal ist*
- Qualitätsmetrik
  - *Quantitative Skala und zugehöriges Verfahren, um den Wert für einen Qualitätsindikator zu bestimmen*

[Balzert 1998]

# Probleme mit der Softwarequalität

- Was sind die Qualitätsmerkmale?
- Welche Merkmale sind relevant? Wie werden sie gewichtet?
- Welche Indikatoren können für ein Qualitätsmerkmal verwendet werden?
- Wie können Indikatoren gemessen werden?
- Wie zuverlässig sind die Qualitätsmetriken?
- Wie lassen sich Metriken kombinieren?
- Wie lässt sich Softwarequalität verbessern?
- Wie lassen sich die zueinander in Konflikt stehenden Anforderungen „Zeit und Kosten reduzieren" vs. „Qualität erhöhen" erfüllen?

# Hierarchie von Qualitätsmerkmalen (1)

- Funktionalität
  - *Bietet das System die verlangten Funktionen an?*
    - Korrektheit: richtige Abbildung Eingaben → Ausgaben
    - Angemessenheit: Eignung für spezifizierte Aufgaben
    - Interoperabilität: Fähigkeit, mit vorgegebenen Systemen zusammenzuarbeiten
    - Standard-Kompatibilität: Einhaltung von Standards
    - Sicherheit: Verhinderung unberechtigten Zugriffs
- Zuverlässigkeit
  - *Führt das System die verlangten Funktionen unter definierten Bedingungen über einen begrenzten Zeitraum aus?*
    - Reife: Geringe Versagenshäufigkeit durch Fehlerzustände
    - Fehlertoleranz: Fähigkeit, Hardware- oder Softwarefehler zu tolerieren
    - Wiederherstellbarkeit: Fähigkeit bei einem Versagen das Leistungsniveau wiederherzustellen

# Hierarchie von Qualitätsmerkmalen (2)

- Benutzbarkeit
  - *Bietet das System eine adäquate Benutzerschnittstelle an?*
    - Verständlichkeit: Aufwand für den Benutzer, das System zu verstehen
    - Erlernbarkeit: Aufwand für den Benutzer, die Anwendung des Systems zu erlernen
    - Bedienbarkeit: Aufwand für den Benutzer, das System zu bedienen
- Effizienz
  - *Verbraucht das System Ressourcen in akzeptablem Umfang?*
    - Zeitverhalten: Antwort- und Verarbeitungszeiten
    - Verbrauchsverhalten: Umfang der benötigten Betriebsmittel und Dauer der Nutzung

# Hierarchie von Qualitätsmerkmalen (3)

- Änderbarkeit
  - *Welcher Aufwand ist erforderlich, um antizipierte Änderungen durchzuführen?*
    - Analysierbarkeit: Aufwand, um Fehlerursachen zu ermitteln und änderungsbedürftige Teile zu bestimmen
    - Modifizierbarkeit: Aufwand zur Durchführung von Änderungen
    - Stabilität: Korrektheit und Zuverlässigkeit nach Änderungen
    - Prüfbarkeit: Aufwand zur Prüfung nach Änderungen
- Portabilität
  - *Welcher Aufwand ist erforderlich, um das System in eine andere Umgebung zu übertragen?*
    - Anpassbarkeit: an unterschiedliche Umgebungen
    - Installierbarkeit: Aufwand zur Installation in neuer Umgebung
    - Austauschbarkeit: Möglichkeit, das System anstelle eines anderen in dessen Umgebung zu verwenden

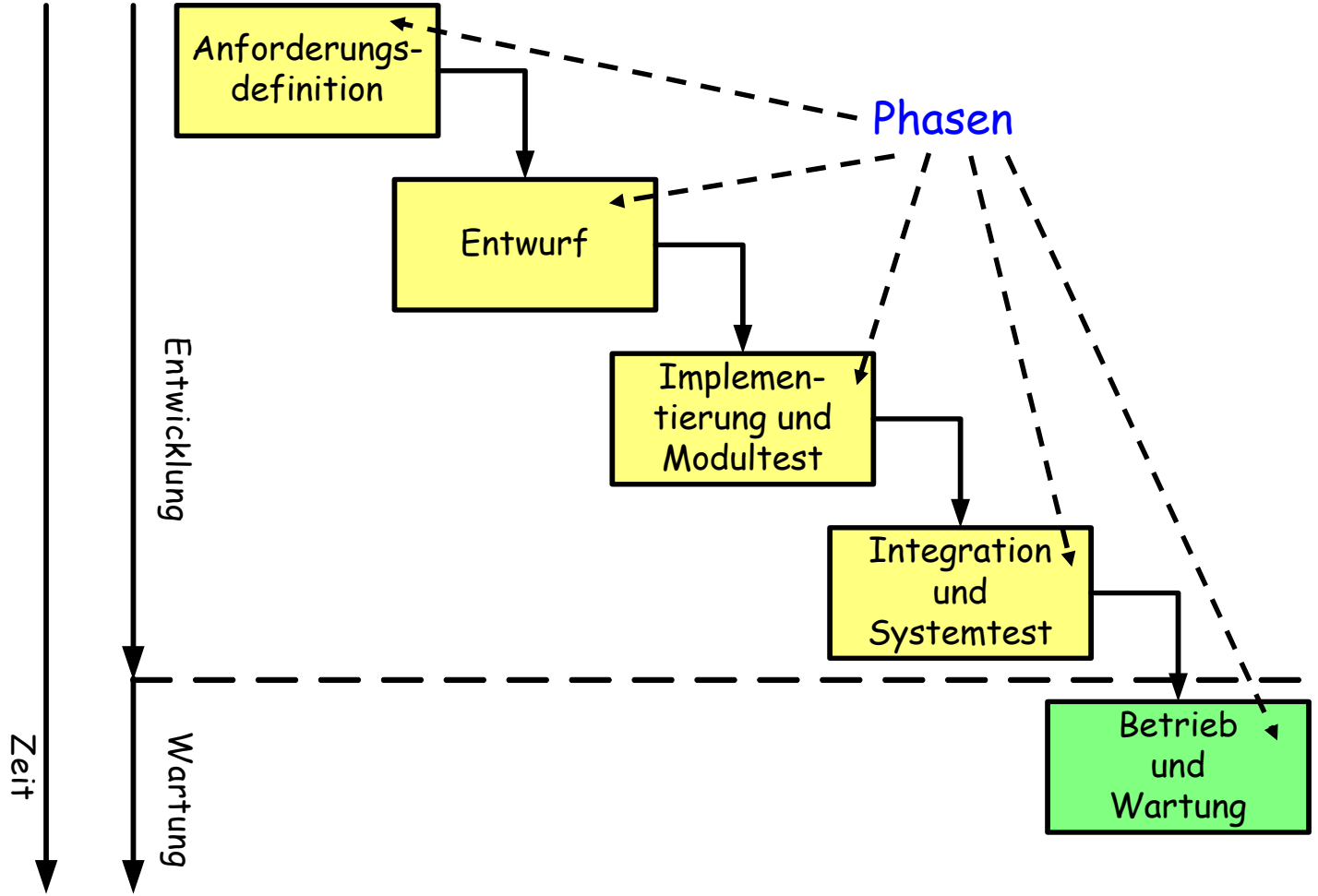
[Balzert 1998]

# Software-Prozessmodelle

# Definitionen

- Prozess
  - *Menge von partiell geordneten Prozessschritten zur Erreichung eines Ziels*
- Prozessmodell
  - *Beschreibung eines Prozesses in der realen Welt, die von irrelevanten Details abstrahiert*
- Lebenszyklusmodell
  - *Grobes Prozessmodell, das den Prozess in Phasen oder Arbeitsbereiche zerlegt*
- Phase
  - *Zeitintervall in der Ausführung eines Prozesses, in dem Aktivitäten einer bestimmten Art ausgeführt werden*
- Arbeitsbereich
  - *Menge von logisch zusammengehörigen Aktivitäten, unabhängig von der Zeit, zu der sie ausgeführt werden*

# Wasserfallmodell

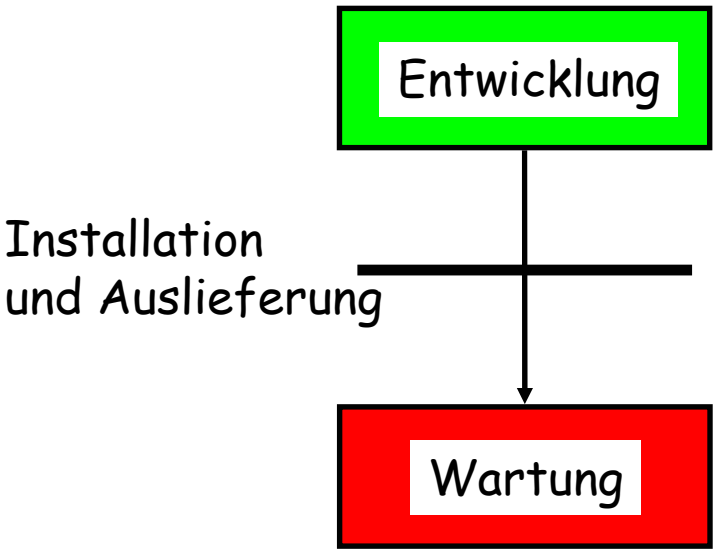


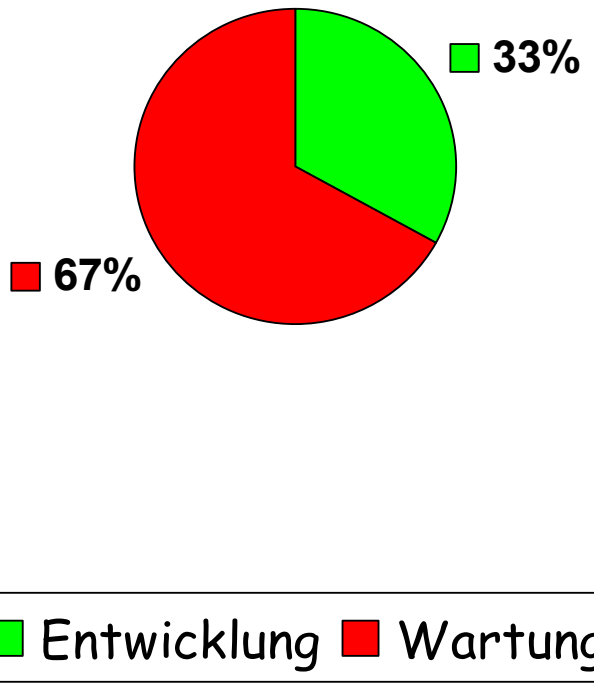
# Bewertung des Wasserfallmodells

- ☺ Wohlstrukturierter ingenieurmäßiger Prozess
- ☺ Fokus erheblich gegenüber der reinen Programmierung erheblich erweitert
- ☺ Fortschritt kann leicht gemessen werden
- ☺ Leicht zu erlernen

- ☹ Strikte Phasensequenz unrealistisch (Überlappungen, Rückgriffe)
- ☹ Anforderungen ändern sich ständig
- ☹ Spätes Feedback vom Kunden
- ☹ Wartung ist keine echte Phase (Aktivitäten aus allen vorherigen Phasen)
- ☹ Verführt zur Unterschätzung der Wartung (üblicherweise 60-80 % der Gesamtkosten!)
- ☹ Essenzielle Aktivitäten fehlen (Projektmanagement, Dokumentation)
- ☹ Bestimmte Aktivitäten werden während des gesamten Lebenszyklus ausgeführt (Projektmanagement, Dokumentation)

# Aufwand für Entwicklung und Wartung



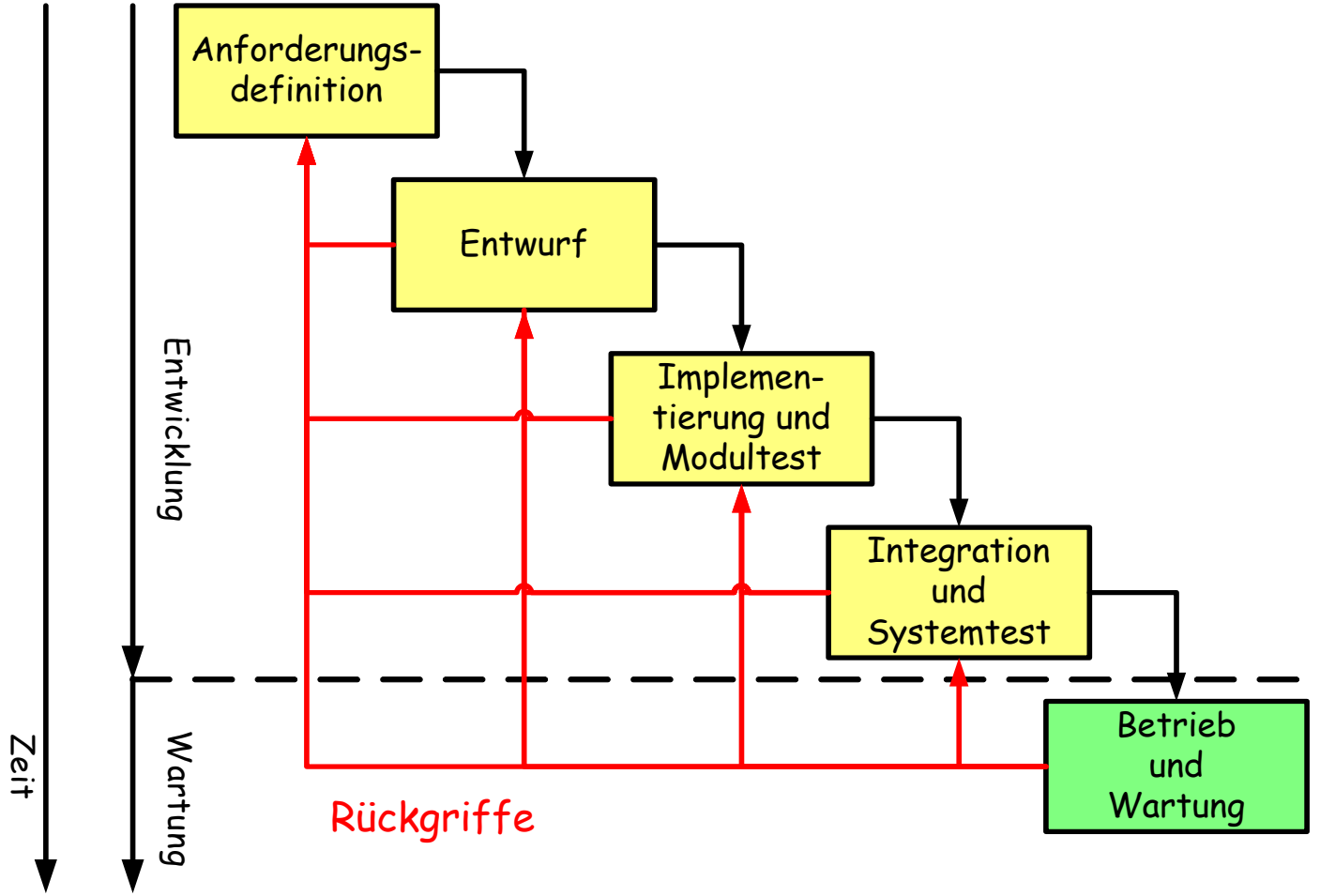
# Klassifikation von Wartungsaktivitäten

| Klasse | Beschreibung | Beispiel |
|---|---|---|
| Korrektive Wartung | Fehler beseitigen | Programmabstürze etc. |
| Adaptive Wartung | Veränderungen der Systemumgebung | Auf neues Betriebssystem portieren |
| Perfektionierende Wartung | Erweiterung oder Veränderung der Funktionalität | Fensterbasierte Benutzerschnittstelle |
| Präventive Wartung | Restrukturierung zur Erleichterung der Wartung | Plattformspezifischen Code isolieren |

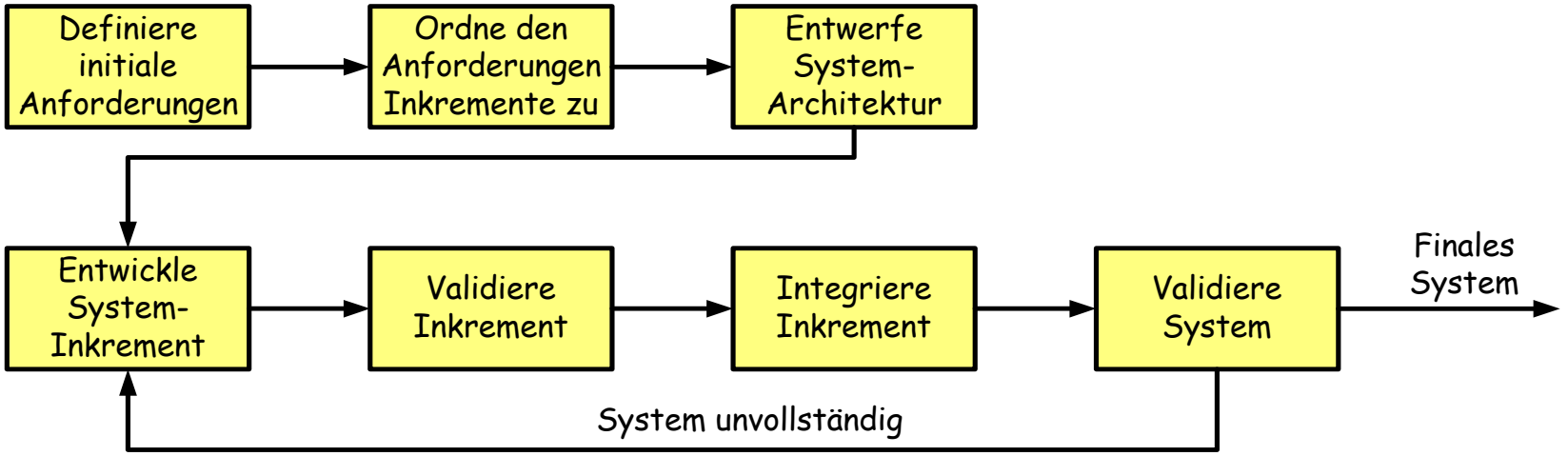
# Verbesserungen des Wasserfallmodells

- **Erweitertes Wasserfallmodell**
  - Wasserfallmodell mit Rückgriffen in frühere Phasen
- **Inkrementelles Entwicklungsmodell**
  - System wird nicht in „Big Bang"-Manier entwickelt; statt dessen wird es in Inkrementen entwickelt
    $\Rightarrow$ ein Zyklus für jedes Inkrement
- **Spiralmodell**
  - Inkrementelles Entwicklungsmodell mit spezifischem Fokus auf dem Management von Risiken
- **Arbeitsbereichsmodell**
  - Softwareprozess wird in Arbeitsbereiche (logisch zusammengehörige Aktivitäten) statt in Phasen (Zeitintervalle) unterteilt
  - Definiert nur die auszuführenden Aktivitäten und deren logische Beziehungen, aber nicht die zeitliche Reihenfolge
  - Kann mit jedem der obigen Prozessmodelle kombiniert werden

# Erweitertes Wasserfallmodell (mit Rückgriffen)

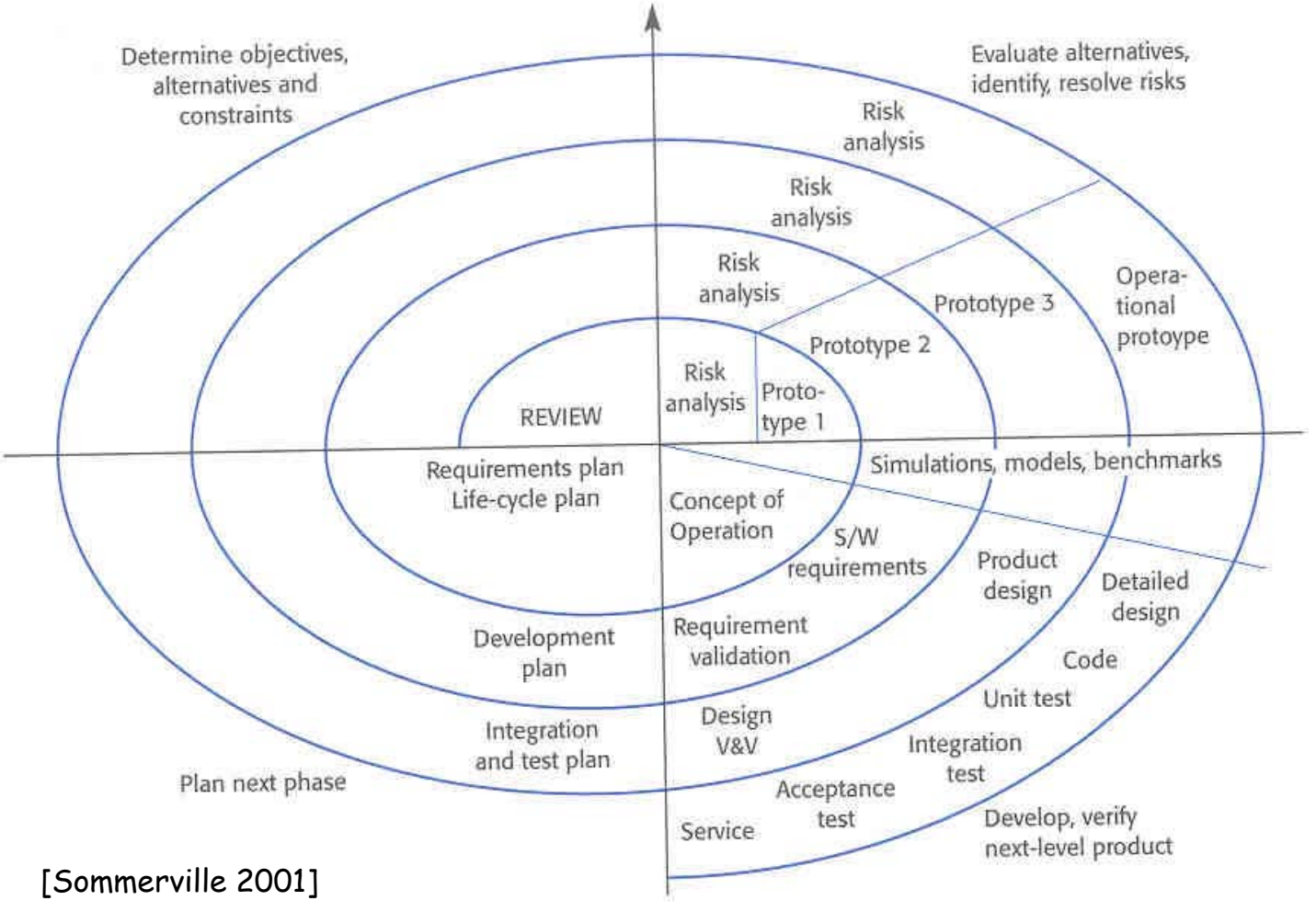


# Inkrementelles Entwicklungsmodell



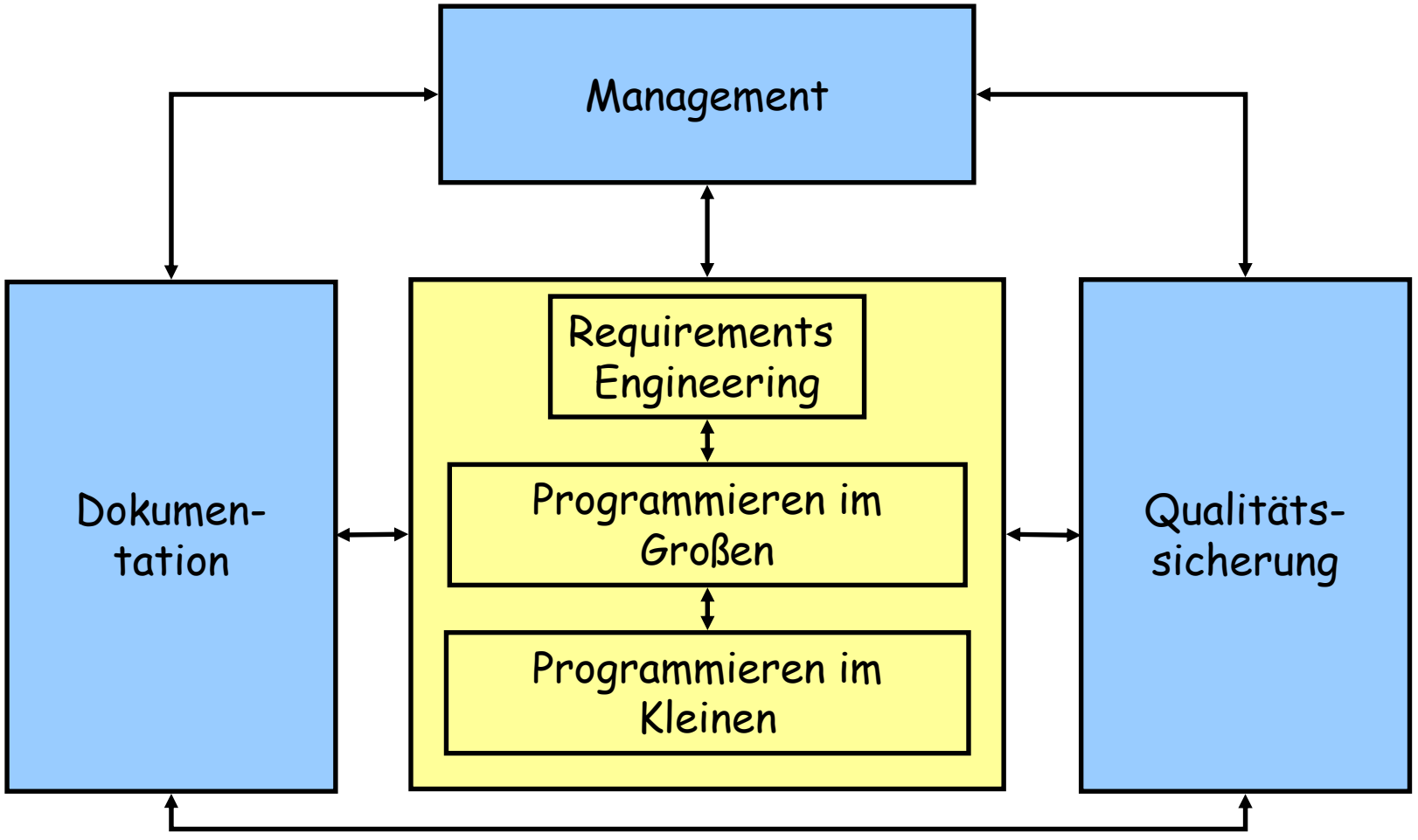

[Sommerville 2001]

# Spiralmodell




[Sommerville 2001]

# Arbeitsbereichsmodell



# Arbeitsbereiche

- **Requirements Engineering**
  - Definition der Anforderungen an ein Softwaresystem
- **Programmieren im Großen**
  - Entwurf der Architektur eines Softwaresystems (Zerlegung in Teilsysteme und Module)
- **Programmieren im Kleinen**
  - Implementieren der Module (Erstellen des Programmcodes)
- **Qualitätssicherung**
  - Verifikation, Tests, Reviews, etc.
- **Dokumentation**
  - Informelle Beschreibung eines Softwaresystems für den Benutzer oder für Systementwickler
- **Management**
  - Koordination aller Aktivitäten im Softwareprozess

# Andere Prozessmodelle

- **Evolutionäre Entwicklung**
  - Softwaresystem wird in einer Reihe von Versionen entwickelt
  - Keine expliziten Phasen oder Arbeitsbereiche
  - Explorativer Ansatz
  - Gut geeignet für unklar definierte oder unbekannte Anforderungen
- **Formale Entwicklung**
  - Anforderungsdefinition wird in eine formale Spezifikation überführt
  - Die Spezifikation wird schrittweise in eine Implementierung transformiert
  - Transformationen können automatisch oder interaktiv ausgeführt werden
  - Jede Transformation muss die Semantik der Spezifikation erhalten
  - Gut geeignet für Probleme, die sich formal beschreiben lassen
- **Entwicklung mit Wiederverwendung**
  - Versuche, Softwaredokumente wiederzuverwenden, statt sie neu zu erstellen
  - Wiederverwendung nicht nur für Programme, auch für Architekturen, Anforderungsdefinitionen etc.
  - Kann mit jedem anderen Prozessmodell kombiniert werden
  - Nutzen der Wiederverwendung muss höher sein als die Kosten

# Ziele und Überblick

# Ziele dieser Veranstaltung

- Problembewusstsein für Software Engineering schaffen
- Fokus über die reine Programmierung hinaus ausdehnen
- Einführung in Software Engineering geben
- Mehrere Paradigmen abdecken (inklusive, aber nicht beschränkt auf objektorientierte Softwareentwicklung)
- (Fast) alle Arbeitsbereiche des Software Engineering abdecken
- Solide Grundkenntnisse im Rahmen des Bachelor-Studiums vermitteln
- Grundlage für weiterführende Veranstaltungen im Master-Studium legen

# Inhalt

- Einführung
- Requirements Engineering
- Programmieren im Großen
- Formale Spezifikation
- Objektorientierte Modellierung
- Projektmanagement
- Konfigurationsverwaltung
- Qualitätssicherung
- Prozessmodelle

# Literatur


- [ANSI 1983]
  ANSI/IEEE Std. 729-1983: *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Inc. (1983)
- [Balzert 2000]
  H. Balzert: *Lehrbuch der Softwaretechnik - Software-Entwicklung*, Spektrum Akademischer Verlag, Heidelberg (2000)
- [Balzert 1998]
  H. Balzert: *Lehrbuch der Softwaretechnik - Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*, Spektrum Akademischer Verlag, Heidelberg (1998)
- [Ghezzi 2002]
  C. Ghezzi, M. Jazayeri, D. Mandrioli: *Fundamentals of Software Engineering*, Prentice Hall (2002)
- [Kernighan 1990]
  B. Kernighan, D. Ritchie: *Programmieren in C*, Hanser Verlag (1990)
- [Krempl 2004]
  S. Krempl: *Das Casino-Prinzip*, c't 11/2004, 218-223 (2004)
- [Sommerville 2001]
  I. Sommerville: *Software Engineering*, Addison-Wesley, Harlow, England (2001)