Kapitel 1

Softwaretechnik: Überblick

Prof. Dr. Rolf Hennicker

19.10.2010

Ziele

- Verstehen, womit sich die Disziplin der Softwaretechnik (engl. Software Engineering) beschäftigt
- Qualitätskriterien von Software kennen
- Die wichtigsten Vorgehensmodelle in der Software-Entwicklung kennen
- Die Grundprinzipien der objektorientierten Software-Entwicklung verstehen

1.1 Einführung

Software = Menge von Programmen mit begleitenden Dokumenten (einschl. Modellen).

Systemsoftware zum Betrieb und zur Wartung von Rechensystemen (Betriebssysteme, spezielle Dienstprogramme wie Editoren, WWW-Browser, Compiler, Shell, ...)

Anwendungssoftware zur Lösung bestimmter, kundenspezifischer Aufgaben in

- Wirtschaft (z.B. Banken, Versicherungen, Reisebüros)
- Verwaltung (z.B. KFZ-, Uni-, Lagerverwaltung)
- Technik (z.B. Steuerungssysteme, eingebettete Systeme, Monitoring, naturw. und Umwelt-Simulationen)

Beachte: Die Erschliessung neuer Anwendungsbereiche und der Umfang von Software nimmt ständig zu.

Problem: Komplexität der Software, häufige Änderungen von Umgebungen und Anforderungen.

Großes Softwaresystem $\geq$ 50.000 lines of code ($\geq$ 10 PJ)
Sehr großes Softwaresystem $\geq$ 1 Mio lines of code ($\geq$ 200 PJ)

Beispiele:
Handy: 200.000 loc, System R/3: 7 Mio loc, Windows95: 10 Mio loc

Konsequenz: Mangelnde Software-Qualität, Fehler!!

Beispiele:
Handy bis zu 600 Fehler (dh. 3 Fehler pro 1000 loc),
Windows95 bis zu 200.000 Fehler (dh. 20 Fehler pro 1000 loc)
Space Shuttle weniger als 1 Fehler pro 10.000 Zeilen.

Man spricht von
normaler SW bei 25 Fehlern pro 1000 loc
guter SW bei 2 Fehlern pro 1000 loc

Historisches:

- 1965 Softwarekrise:
  Umfang der SW für komplexe Anwendungen wird nicht mehr beherrscht,
  Methoden des "Programmierens im Kleinen"greifen nicht mehr
- 1968/69:
  Der Begriff "Software Engineering"wird geprägt
  Idee: Anwendung ingenieurmäßiger Methoden bei der SW-Entwicklung

Definition (nach Fairley 1985):
Software Engineering ist die technische und organisatorische Disziplin zur systematischen Herstellung und Wartung von Softwareprodukten, die zeitgerecht und innerhalb vorgegebener Kostenschranken hergestellt und modifiziert werden.

Ziele des SW Engineering

- Software hoher Qualität
  (aus Sicht des Benutzers und des Entwicklers)
- Beherrschung der Kosten und der Entwicklungszeit
  (aus wirtschaftlicher Sicht)

Bereiche des Software Engineering

- **Software-Entwicklung**
  - Definition von Anforderungen
  - Entwurf von Lösungen
  - Implementierung und Wartung
  - Erstellen von Modellen und Dokumenten
- **Qualitätssicherung**
  - Analytische Maßnnahmen wie Codeinspektion, Testen, Validieren, Verifizieren
  - Konstruktive Maßnahmen wie Einhaltung methodischer Richtlinien, Wiederverwendung qualitativ hochwertiger Komponenten
- **Projektmanagement**
  - Projektplanung (Projektbeschreibung, Zeitplan mit Meilensteinen, Kosten- und Personalplanung)
  - Projektkoordination (nach innen und nach aussen)
  - Projektkontrolle (Bewertungen, Risikoabschätzung, Berichte)

Zur Lösung dieser Aufgaben verwendet man

- **Vorgehensmodelle** (Aus welchen Phasen besteht der SW-Lifecycle?)
- **Techniken** (Notationen)
  zur konkreten Beschreibung und Lösung einzelner Aufgaben (z.B. UML, Java)
- **Methoden**
  zum systematischen Einsatz der Techniken
- **Werkzeuge** (CASE-Tools)
  zur Projektplanung, zum Editieren von Programmen und Grafiken, zur Codegenerierung, zur Compilierung und Interpretation, zur Spezifikations- und Programmanalyse, zum Testen, zum Konfigurationsmanagement

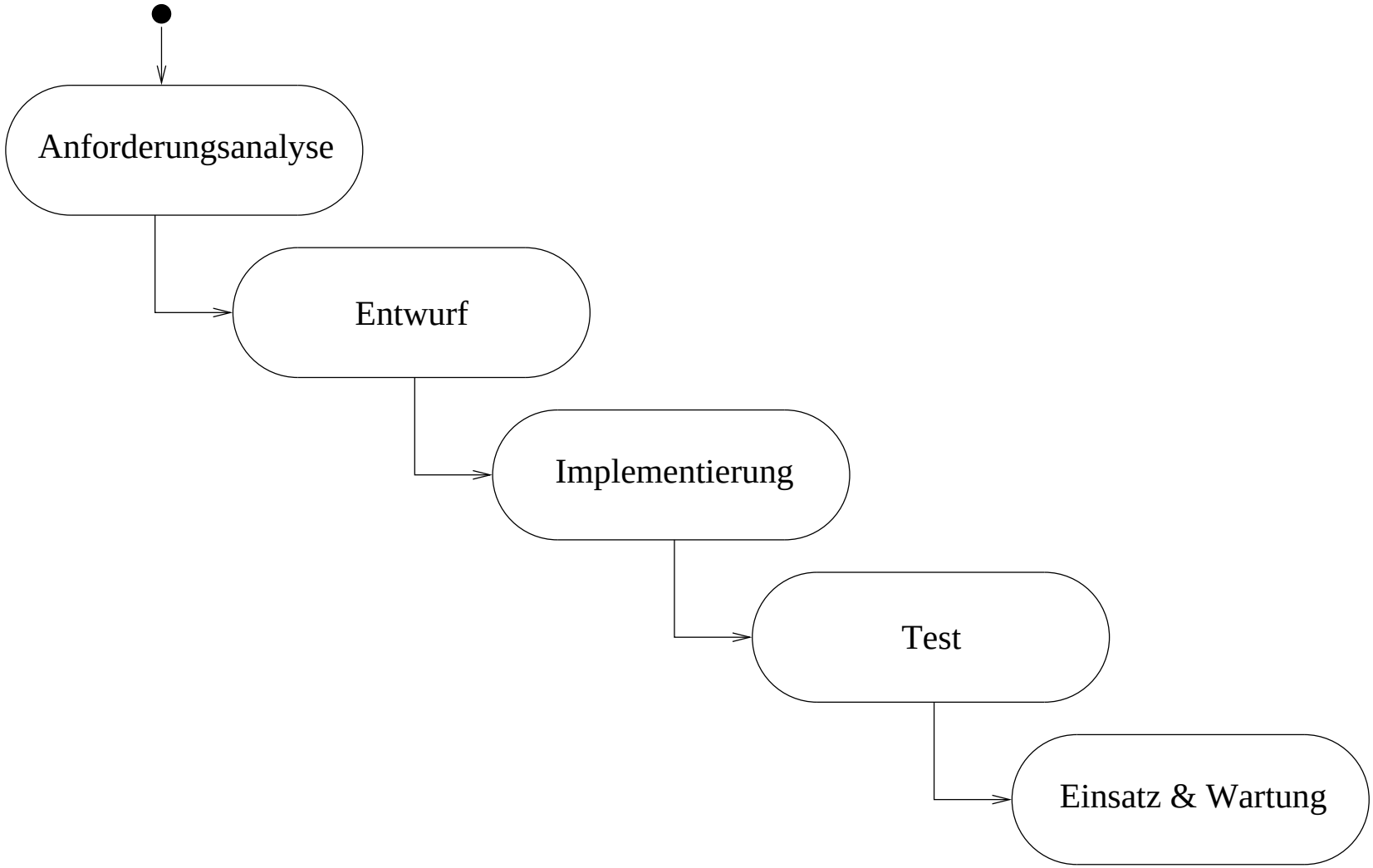
1.2 Qualitätskriterien von Software

1. **Korrektheit:**
   Ein Softwareprodukt erfüllt die in einer Spezifikation beschriebenen Anforderungen
   Beachte:
   - Der Nachweis der Korrektheit ist nur mit formalen Methoden möglich.
   - Falls die (formale) Spezifikation nicht die Anforderungen des Anwenders wiedergibt, leistet auch korrekte Software nicht das Gewünschte.
2. **Zuverlässigkeit:**
   Wahrscheinlichkeit, dass ein SW-Produkt eine gewünschte Funktion in einer bestimmten Zeit erfüllt. Hohe Zuverlässigkeit bedeutet, dass Fehler selten auftreten und nur geringe Auswirkungen haben.
3. **Robustheit:**
   Sinnvolle Reaktion bei Fehlern, die in der Umgebung auftreten (z.B. Bedienungsfehler, Fehler anderer Systeme)

4. **Benutzerfreundlichkeit:**
   Einfache Erlernbarkeit und Bedienbarkeit eines SW-Systems (Übersichtliche und adäquate Benutzerschnittstelle, Lernprogramme, Hilfen, verständliche Fehlermeldungen).
5. **Wartbarkeit:**
   - Möglichst einfache Lokalisierung und Behebung von Fehlern.
   - Möglichst einfache **Änderbarkeit** und **Erweiterbarkeit**.
6. **Performanz:**
   Angemessenes Laufzeitverhalten und Speicherplatzbedarf
7. **Dokumentation:**
   Sowohl für den Benutzer (Handbücher) als auch für den (Weiter-)Entwickler (Dokumentation der einzelnen Entwicklungsschritte, der Implementierung, von Testplänen, ...).
8. **Portabilität:**
   Übertragbarkeit der Software auf andere Rechner.

1.3 Vorgehensmodelle

Das Wasserfallmodell



Anforderungsanalyse (engl. requirements analysis):

- Analyse des Problembereichs
- Festlegung der (funktionalen und nicht funktionalen) Anforderungen an das System
  Was *soll das System leisten?*
- Festlegung organisatorischer Richtlinien (Aufwandsabschätzung, Terminplanung,...) und von Rahmenbedingungen (z.B. vorhandene Soft- und Hardware).
- Machbarkeitsstudie
- Skizze der Systemarchitektur

Ergebnis der Anforderungsanalyse ist eine Anforderungsbeschreibung (Spezifikation, Modell). Diese dient

- als "Vertrag"zwischen Anwender und SW-Entwickler und
- als Grundlage für die weitere Systementwicklung.

Problem: Mögliche Missverständnisse zwischen Anwender und Entwickler.

Entwurf (engl. design):

- Beschreibt die Art und Weise, in der die gestellten Aufgaben gelöst werden sollen.
  Wie *lösen wir das Problem?*
- Festlegung der Systemarchitektur
- Entwurf der einzelnen Systemkomponenten (Wahl von Datenrepräsentationen und Algorithmen)

Ergebnis der Entwurfsphase ist eine (konstruktive) Entwurfsbeschreibung (Spezifikation, Modell).

Implementierung:
Codierung des Entwurfs in einer Programmiersprache, ggf unter Wiederverwendung vorhandener Komponenten.

Test:

- Test der einzelnen Komponenten ("unit testing")
- Schrittweises Zusammenfügen einzelner Komponenten mit jeweiligem Integrationstest
- Systemtest
- Abnahmetest (mit "echten"Daten des Anwenders)

Wartung:

- Fehlerbeseitigung nach Inbetriebnahme
- Änderung und Erweiterung des Systems

Schätzung der aktuellen Kosten in der Systementwicklung

Anforderungsanalyse	6%
Entwurf	5%
Implementierung	7%
Integration und Test	15%
Wartung	67% (davon 40% um existierende Programme zu verstehen!)

Bemerkungen zum Wasserfallmodell

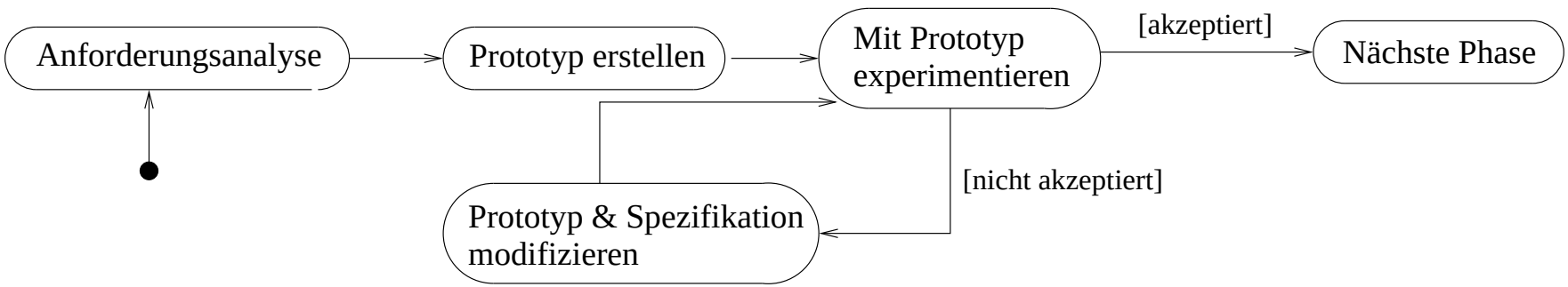
- Klare Trennung der verschiedene Phasen
- Schwierigkeiten in einer Phase verzögern das Gesamtprojekt
- Validierung des Produkts durch den Kunden ist erst nach Abschluss der gesamten Entwicklung möglich
- Streng sequentielle Vorgehensweise ist in der Praxis kaum möglich, da
  - Fehler aus früheren Phasen häufig erst später erkannt werden
  - Anforderungen sich ändern können
- Häufig werden einzelne Phasen weiter verfeinert, z.B. Grob- und Feinentwurf

Das Prototyp-orientierte Modell

Prototyp = Vorabversion (von Teilen) des intendierten Systems.

Charakteristika von Prototypen:

- schnelle und billige Herstellung
- häufig zur Demonstration von Benutzeroberflächen
- eingeschränkte Funktionalität, nicht robust, schlechte Performanz

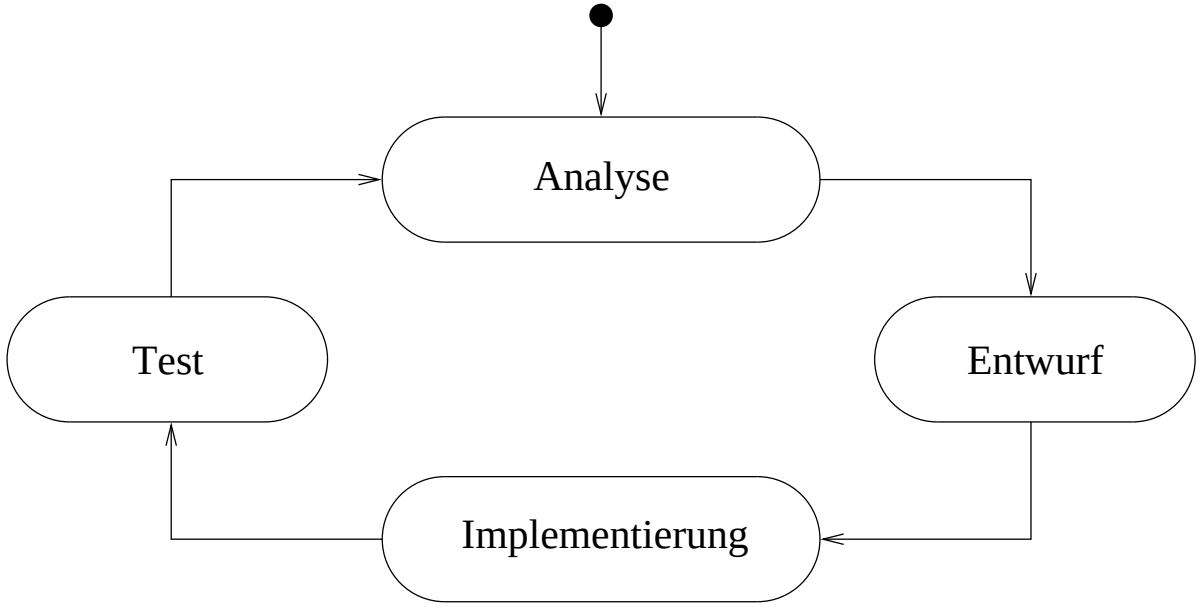


Vorteil: Frühzeitige Validierung durch den Kunden

Unterscheide: Wegwerf-Prototypen und Prototypen zur Weiterentwicklung

Iteratives Vorgehensmodell

Der Entwicklungsprozess besteht aus einer Folge von Zyklen (Iterationen).



Am Ende jedes Zyklus steht eine neue (ausführbare) Version des SW-Produktes, die die vorherige verbessert und erweitert.

Wartung = weiterer Zyklus zur Erstellung eines verbesserten Produkts nach Inbetriebnahme.

Vorteil: Häufiger Kontakt zum Anwender jeweils bei der Erstellung einer neuen Version ("evolutionäres Prototyping")

Nachteil: Risiko, dass die Architektur der bereits implementierten Teile nicht zu den neu hinzukommenden Anforderungen passt.

Spezielle Ausprägung: Unified Process (UP) nach Jacobson, Booch und Rumbaugh (1999).

Charakteristika des Unified Process:

- Anwendungsfall gesteuert ("Use Case driven")
- Architektur-zentriert
- Iterativ, wobei jede Iteration aus den Arbeitsschritten "Anforderung", "Analyse", "Entwurf", "Implementierung"und "Test"besteht.
- Verschiedene Iterationen werden in einzelnen Phasen zusammengefasst, wobei es die 4 Phasen "Beginn", "Ausarbeitung", "Konstruktion"und "Umsetzung"gibt.

Beginn (Inception): Eine "Vision" des Systems wird entwickelt

- Bestimmung der wichtigsten Anwendungsfälle
- Grobe Skizze der Systemarchitektur

Ausarbeitung (Elaboration):

- Genaue Beschreibung der meisten Anwendungsfälle
- Realisierung von essentiellen Anwendungsfällen
- Ausarbeitung der Systemarchitektur

Konstruktion (Construction):

- Implementierung aller Anwendungsfälle
- Stabilisierung der Systemarchitektur

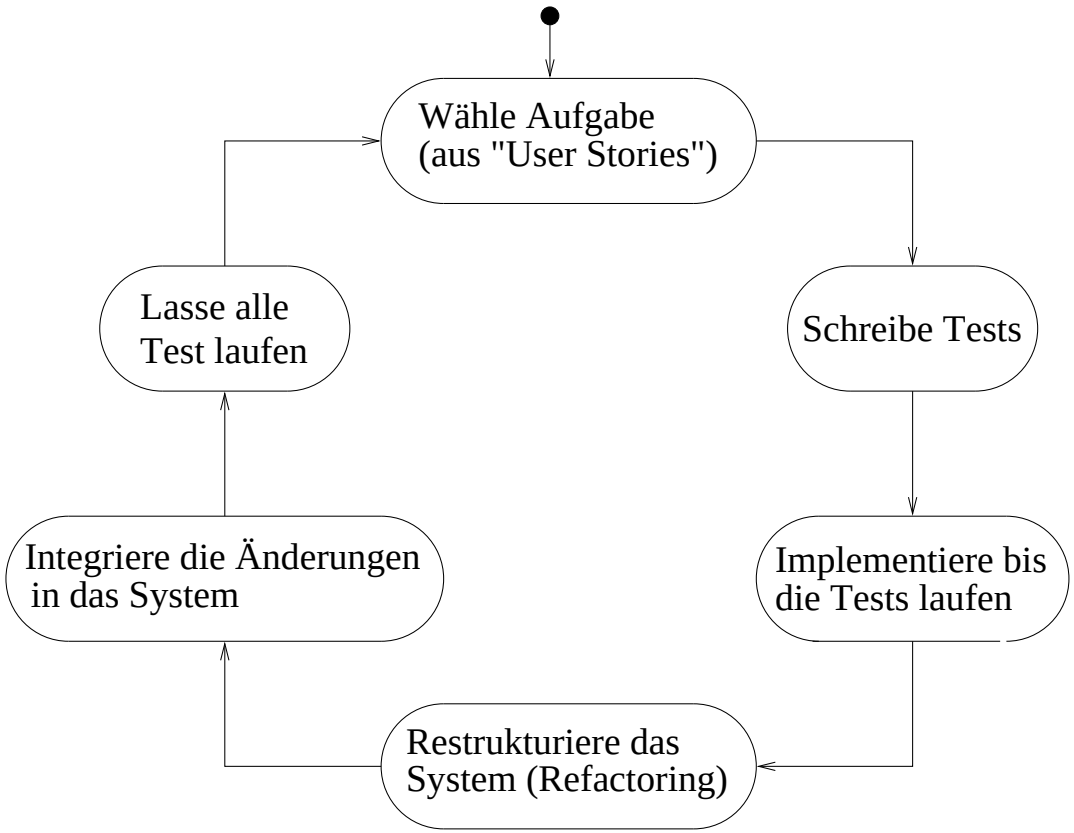
Umsetzung (Transition):

- Erstellen einer β-Version
- Testen und Verbessern
- Übergabe an den Kunden

XP: eXtreme Programming

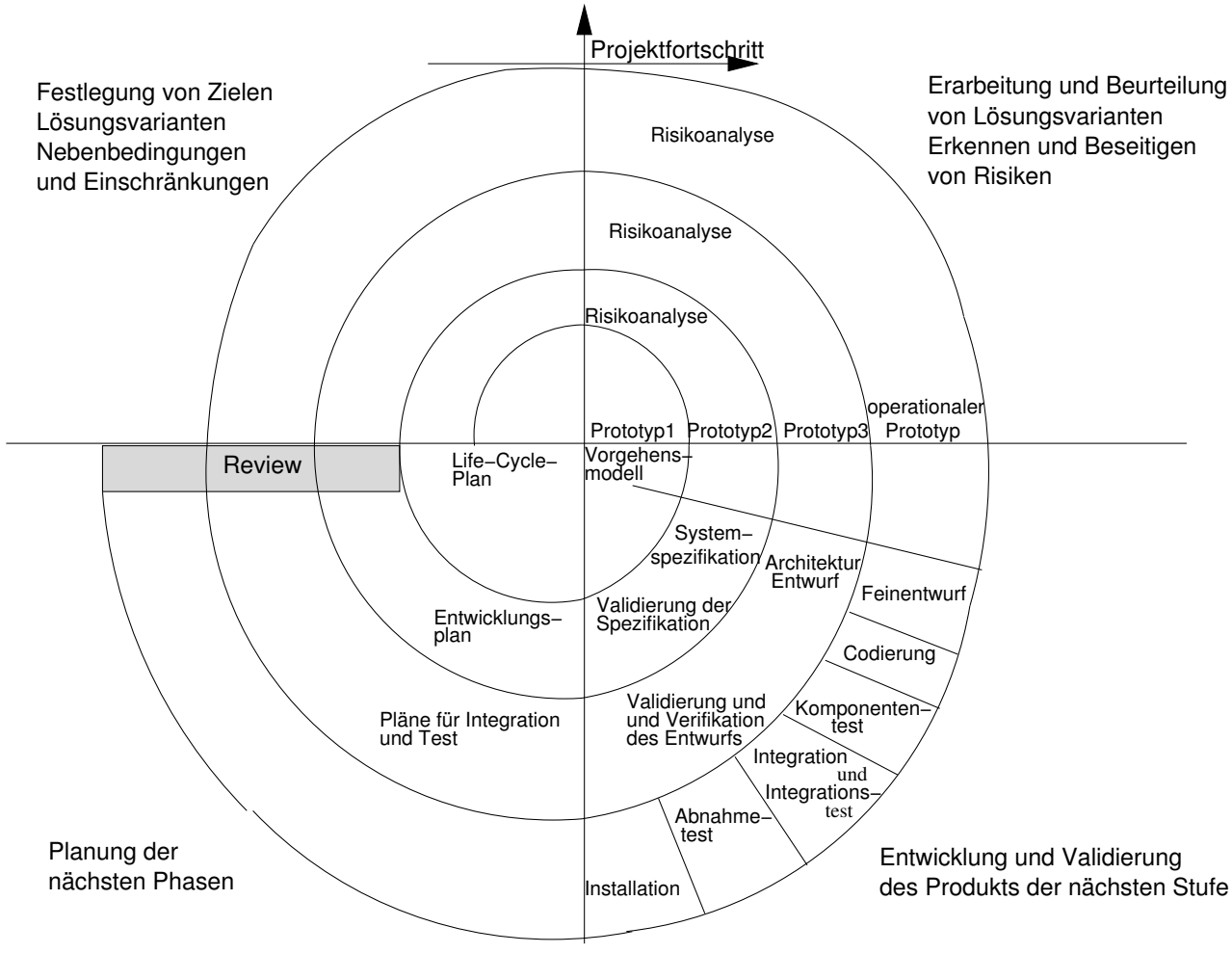
Charakteristika

- Test-getrieben
- viel Kommunikation (paarweises Programmieren, schnelle Rückmeldung durch den Anwender)
- Einfachheit (implementiere das Einfachste, das funktioniert)
- inkrementelle Erweiterung (iterativ)



Das Spiralmodell (nach Boehm, 1988)

Grundidee: Risikobeherrschung



1.4 Grundprinzipien der objektorientierten SW-Entwicklung

- Ein System besteht aus einer Menge von Objekten, die sich (gegenseitig) Nachrichten schicken. Der Empfang einer Nachricht löst eine Operation des (empfangenden) Objekts aus.
- Ein Objekt ist ein individuelles Exemplar mit einer eindeutigen Identität.
- Eine Klasse beschreibt eine Menge von Objekten mit gemeinsamen Merkmalen.
- Merkmale von Objekten sind:
  - Attribute (z.B. Name, Alter einer Person, ...)
  - Operationen, die jedes Objekt der Klasse ausführen kann (z.B. radfahren, schreiben, ...)
  - (mögliche) Beziehungen zu anderen Objekten (z.B. Auto gehört einer Person)
- Die aktuellen Attributwerte (und Beziehungen) zu einem Zeitpunkt bestimmen den Objektzustand.
- Die aktuellen Zustände aller zu einem Zeitpunkt existierenden Objekte (und deren Beziehungen zu anderen Objekten) bestimmen den Systemzustand.
- Klassen können spezialisiert werden (z.B. Auto ist ein spezielles Fahrzeug) (*Vererbungsprinzip!*)

Vorteile der objektorientierten SW-Entwicklung

- Objekte der "realen Welt" können auf ein objektorientiertes Modell und ein objektorientiertes System abgebildet werden.
- Objektorientierte Techniken können auf verschiedenen Ebenen der Systementwicklung eingesetzt werden.

  $$\text{OO-Analyse} \rightarrow \text{OO-Entwurf} \rightarrow \text{OO-Programm}$$

- Einfache Erweiterbarkeit und Wiederverwendbarkeit (z.B. durch Hinzunahme von Subklassen durch Vererbung).

Zusammenfassung

- Ziele des SW Engineerings sind
  - die Erstellung von Software hoher Qualität
  - die Beherrschung der Kosten und der Entwicklungszeit
- Bereiche des SW Engineerings sind SW-Entwicklung, Qualitätssicherung, Projektmanagement.
- In der SW-Entwicklung verwenden wir Vorgehensmodelle, Methoden, Techniken und Werkzeuge.
- Qualitätskriterien von SW sind Korrektheit, Zuverlässigkeit, Robustheit, Benutzerfreundlichkeit, Wartbarkeit, Effizienz, Dokumentation, Portabilität.
- Bekannte Vorgehensmodelle sind Wasserfallmodell, Prototyp-orientiertes Modell, Iteratives Modell (z.B. UP), XP (eXtreme Programming), Spiralmodell.
- Ein objektorientiertes System besteht aus einer Menge von Objekten, die Nachrichten austauschen und auf den Empfang von Nachrichten reagieren (z.B. durch Änderung des Objektzutands).