

Martin Glinz Harald Gall

# Software Engineering

Kapitel 13

## Prozesse und Prozessmodelle



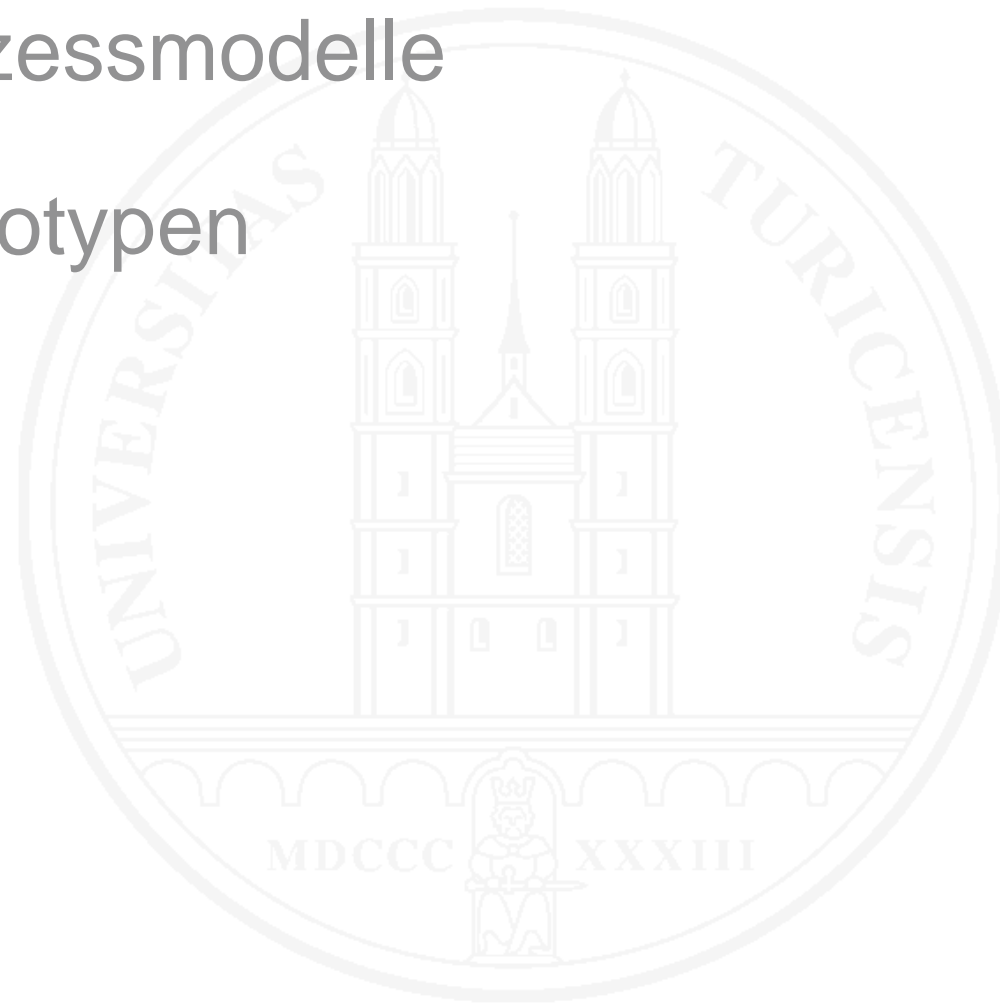
Universität Zürich  
Institut für Informatik

# 13.1 Grundlagen

---

## 13.2 Prozessmodelle

## 13.3 Prototypen



# Prozesse

---

**Prozess (process)** – **Ablauf** eines Vorhabens mit der Beschreibung der **Schritte**, der **beteiligten Personen**, der für diesen Ablauf **benötigten Informationen** und der dabei **entstehenden Informationen**

- **Ad-hoc-Prozesse**: spontan, individuell, ungerregelt
- **Systematische Prozesse**: Ablauf geplant und gelenkt
- **Herstellung und Pflege von Software** sind **Prozesse**

# Nicht-professioneller Bereich

---

Typischerweise **ad-hoc-Prozesse**

- Vage Sachziele
- Keine Termin- und Kostenvorgaben
- Keine Spezifikationen, bruchstückhafte Entwürfe
- Tests nur ansatzweise, Qualität ist kein Thema
- Entwicklung hat Probier-Charakter
- Produkte eher klein, kurzlebig und nicht dokumentiert
- In der Regel Einpersonennarbeit für den Eigengebrauch
- Pflegemaßnahmen spontan

# Professioneller Bereich

---

- Ad-hoc-Prozesse nur für experimentelle Entwicklungen und Wegwerf-Prototypen
- Leider oft auch in sogenannter Endbenutzer-Entwicklung
- Entwicklung von Software als **Produkt** oder **Produkt-Teil**:  
**geplante und gelenkte Prozesse** notwendig
  - Abwicklung wird **geplant**
  - Klare **Termin-, Kosten- und Sachziele**
  - **Ablauf** des Prozesses wird **überprüft**
  - **Lenkungsmaßnahmen** bei Störungen im Ablauf (präventiv / reaktiv)
  - **Qualität** ist **wichtig**
  - Produkt wird **gepflegt** und in der Regel **weiterentwickelt**

# Systematische Entwicklung

---

- Prozess mit **definiertem Anfang** und **Ende**
- Ziel: Bereitstellung von Software, welche eine **gegebene Aufgabe erfüllt**
- Optionen
  - Entwicklung **neuer Software**
  - **Beschaffung, Konfigurierung** und **Anpassung** von Software

# Systematische Pflege (Wartung)

---

- Kontinuierlicher Prozess
- Erhaltung der Gebrauchstauglichkeit des Produkts als Ziel
- Regelt Reaktion auf aufgetretene Probleme
- Plant und lenkt
  - Weiterentwicklung der Software
  - laufende Anpassung an sich wandelndes Umfeld

# Software-Projekte

---

## Abwicklungsrahmen für

- systematische Entwicklung von Software
- größere, abgrenzbare Pflegevorhaben

**Projekt (project)** – zeitlich befristete Organisationsform zur Bearbeitung einer gegebenen Aufgabe. Dient zur Regelung der Verantwortlichkeiten und zur Zuteilung der für die Arbeit notwendigen Ressourcen.

Jede systematische Entwicklung von Software und jedes größere Software-Pflegevorhaben werden als **Projekt** organisiert.



# Software-Projekte vs. klassische Projekte

---

Software-Projekt  $\neq$  klassisches technisches Entwicklungsprojekt

Wesentlichster Unterschied: Software ist **immateriell**

- ⇒ keine natürlichen Ansatzpunkten für **Fortschrittskontrolle** und **Problemerkennung**
- ⇒ Sorgfältige **Planung**, **Prüfung** und **Lenkung** sehr wichtig
- ⇒ Verwendung (und Einhaltung) **geeigneter Prozesse**
- ⇒ adäquater Umgang mit **Projektrisiken**

# Software-Lebenslauf

---

- Jede isolierte Software-Komponente hat einen **Lebenslauf**
- Stadien: **Initiierung** – **Entwicklung** – **Nutzung**
- Tätigkeiten für die Erstellung einer Einzelkomponente:
  - **Spezifizieren** der Anforderungen
  - **Konzipieren** der Lösung
  - **Entwerfen** der Lösung im Detail
  - **Codieren** und Testen
  - **Integrieren** und Testen
  - **Installieren** und Testen

**Lebenslauf (life cycle)** – Zeitraum, in dem eine Software-Komponente initiiert, entwickelt oder genutzt wird

# Drei Klassen von Software

---

Lehman teilt Software in drei Klassen ein:

- **S-Typ:** Vollständig durch formale Spezifikation beschreibbar.  
Erfolgskriterium: Spezifikation nachweislich erfüllt
- **P-Typ:** Löst ein abgegrenztes Problem.  
Erfolgskriterium: Problem zufriedenstellend gelöst
- **E-Typ:** Realisiert eine in der realen Welt eingebettete Anwendung.  
Erfolgskriterium: Anwender zufrieden
  
- Software vom S-Typ ist **stabil**
- Software vom P- und E-Typ ist einer **Evolution** unterworfen

# Software-Evolution

---

**Evolution (evolution)** – Wandel der Software durch Wandel des Umfelds

- Jedes technische Produkt unterliegt einer Evolution
- In der **Informatik**
  - Evolution läuft besonders **schnell**
  - Teilweise **selbststeuernd**

# Konsequenzen für P- und E-Software

---

- Nie über mehrere Jahre hinweg gebrauchstauglich ohne Veränderungen
  - **Pflege ist kein Unfall**, sondern unvermeidlich
  - Überlegungen zu Software immer auf die **ganze Lebensdauer** beziehen
  
- ⇒ Längere Projekte: **Änderungen** der Anforderungen während der Entwicklung **wahrscheinlich**
  
- ⇒ **Bewegliche Ziele sind kein Unfall**, sondern naturbedingt

# Meilensteine

---

**Meilenstein (milestone)** – eine **Stelle** in einem Prozess, an dem ein geplantes **Ergebnis** vorliegt.

- **Das** Mittel zur
  - **Strukturierung** eines Prozesses / Projekts
  - **Kontrolle** des **Projektfortschritts**
  
- **Planung** durch Festlegung von Ergebnis und Ressourcen
- **erreicht**, wenn Ergebnis vorliegt
- **Kontrolle** durch Vergleich von SOLL-Aufwand mit IST-Aufwand
- Nur **wirksam**, wenn **messbar**

# Eignung von Zwischenresultaten für Meilensteine

---

## Ergebnis in einem Software-Prozess

Codierung zu 90% beendet

Die Komponente xyz hat internen Abnahmetest bestanden

Die Anforderungsspezifikation liegt vor

## Eignung für Meilenstein

**unbrauchbar**, da nicht messbar, sondern nur schätzbar

**geeignet**, da messbar (anhand des Abnahmetest-Protokolls)

**geeignet**, wenn Inhalt und Form des Dokuments durch ein Review überprüft werden

# Mini-Übung 13.1: Meilensteine

Ein zu entwickelndes System besteht aus einer Basiskomponente mit Datenbank und Benutzerschnittstelle sowie sechs voneinander weitgehend unabhängigen Anwendungsfunktionen.

Für die Basiskomponente wird ein Entwicklungsaufwand von 12 Personenwochen geschätzt; der Aufwand pro Anwendungsfunktion auf je drei Personenwochen.

Sie sind Projektleiterin. Ihr Management verlangt einen Meilenstein mit dem Kriterium «System zu 80% fertig».

- a) Wie reagieren Sie?
- b) Formulieren Sie ein messbares Kriterium für einen solchen Meilenstein.

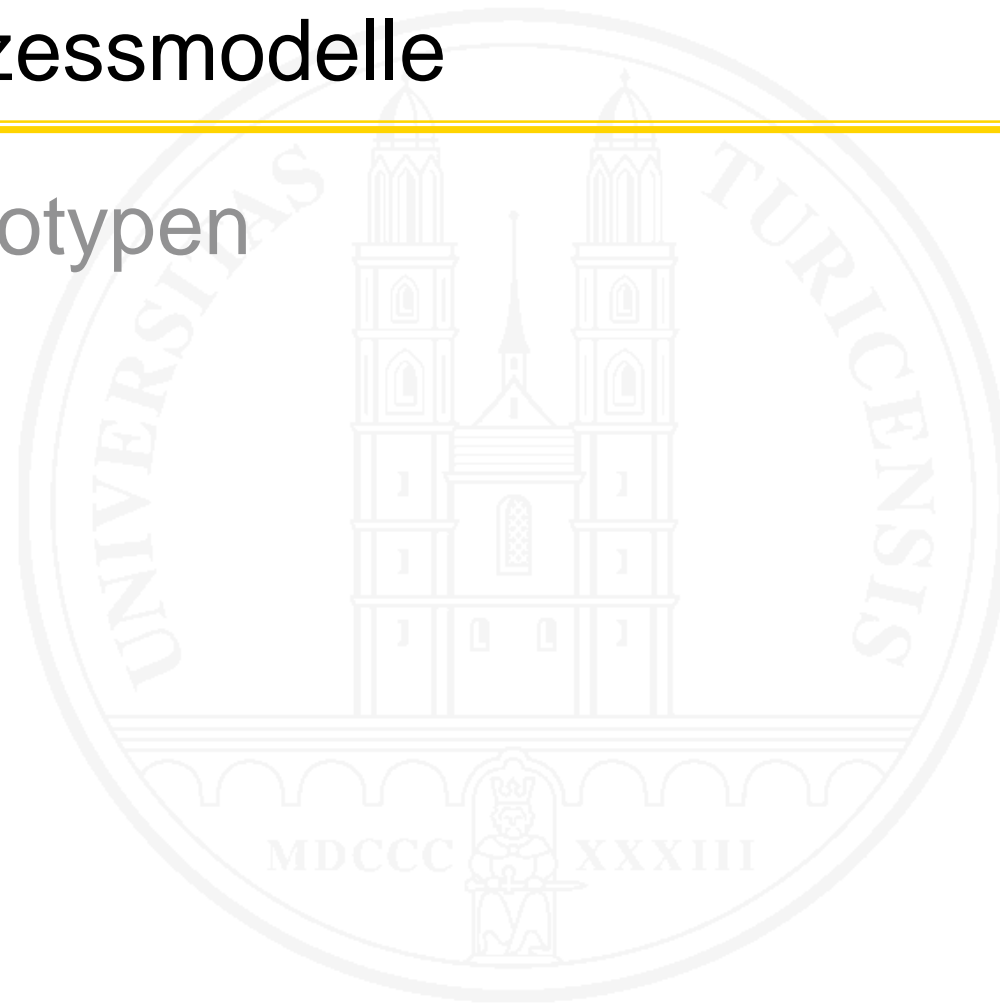


# 13.1 Grundlagen

# 13.2 Prozessmodelle

---

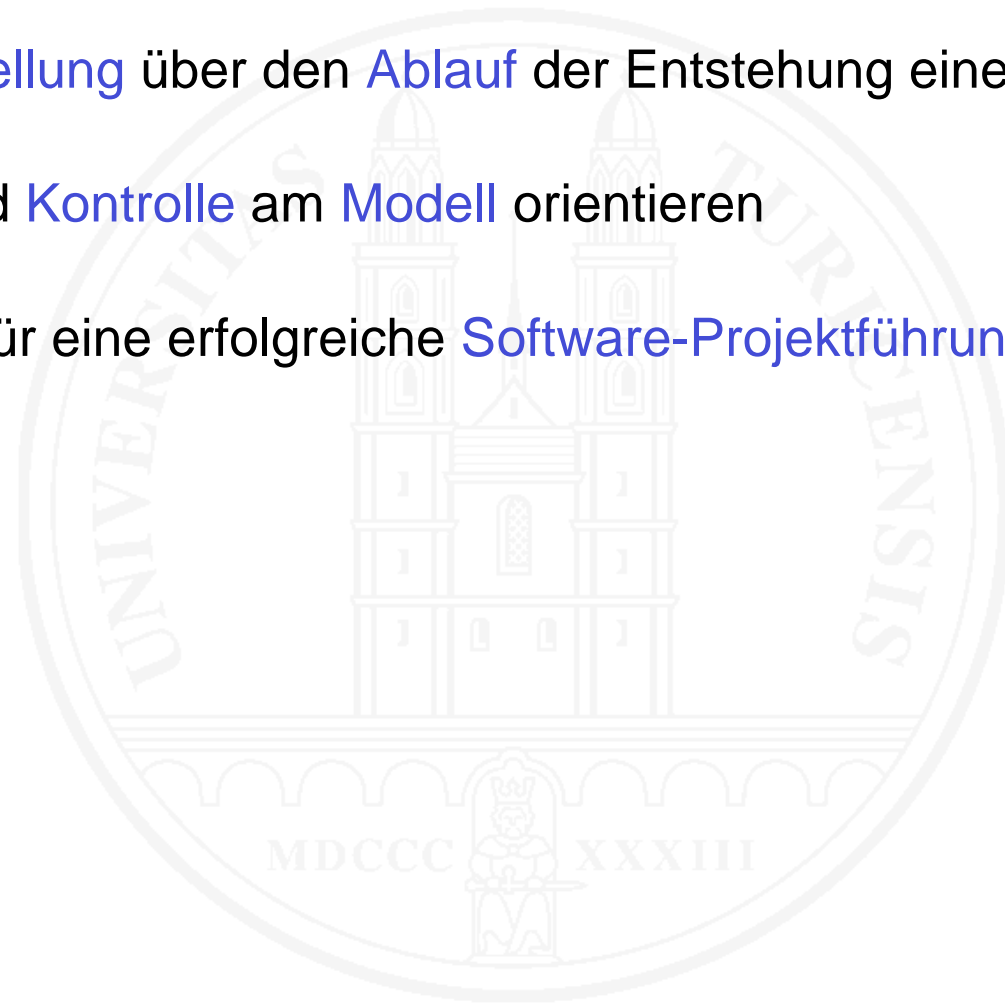
# 13.3 Prototypen



# Software-Prozessmodelle

---

- **Modellvorstellung** über den **Ablauf** der Entstehung einer Software
- **Planung** und **Kontrolle** am **Modell** orientieren
- **Grundlage** für eine erfolgreiche **Software-Projektführung**



# Ausgewählte, charakteristische Prozessmodelle

---

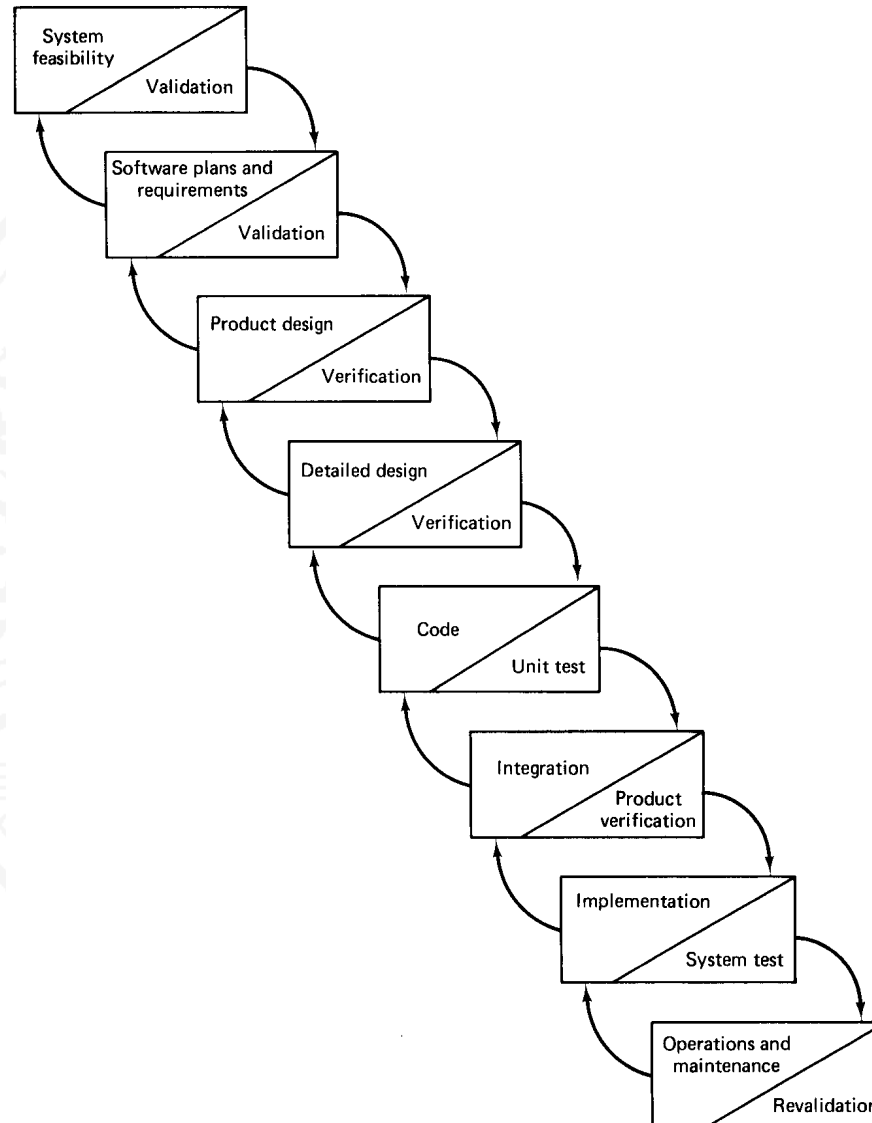
- **Wasserfall-Modell:** Gliederung in eine Folge von **Aktivitäten**
- **Ergebnisorientiertes Phasenmodell:** Gliederung in eine Folge von **Zeitabschnitten**; Phasenabschluss durch Meilensteine
- **Wachstumsmodell:** Gliederung in eine Folge von **Lieferschritten**
- **Spiralmodell:** Gliederung in eine zyklische, am **Risiko orientierte** Folge von Entwicklungsschritten

Außerdem

- **Agile Software-Entwicklung:** „**schlanke**“ Prozesse
- **Der Pflegeprozess**

# Wasserfall-Modell – 1: Das Prinzip

Wasserfall-Modell  
nach Boehm

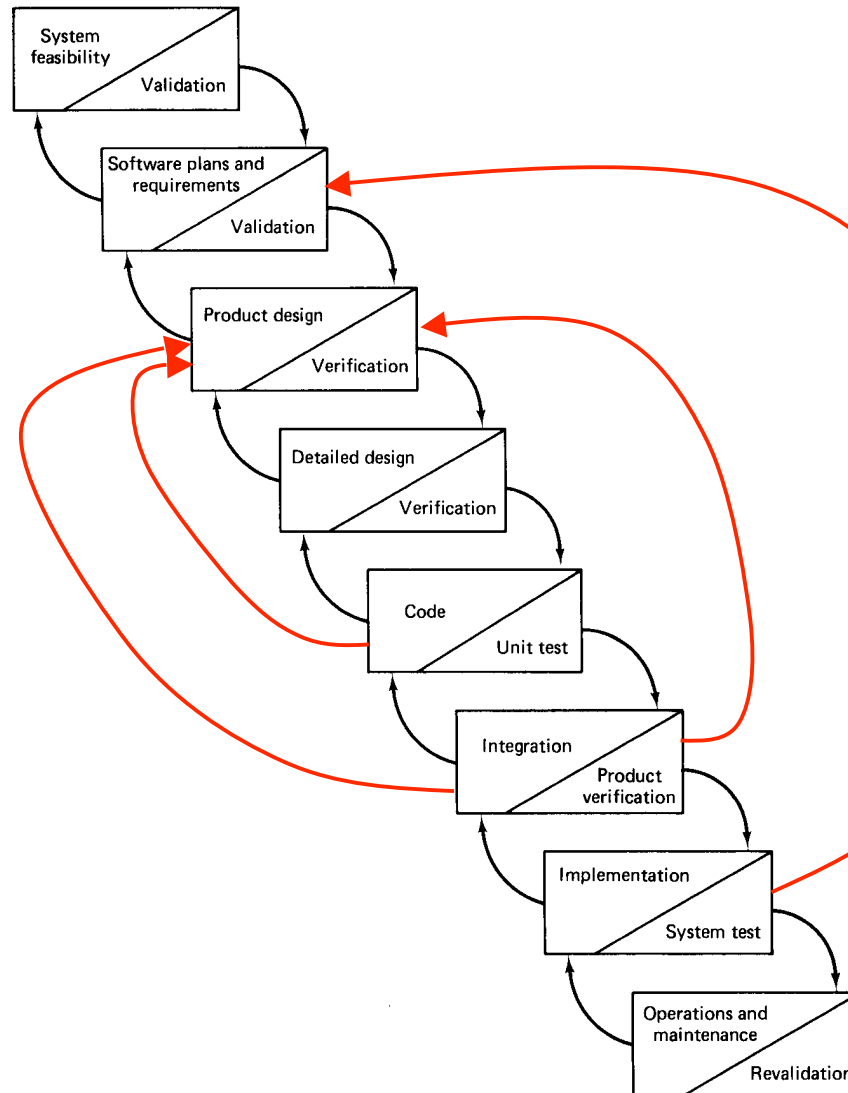


# Wasserfall-Modell – 2: Eigenschaften

---

- Ältestes systematisches Prozessmodell
- Entwicklung als Sequenz aufeinanderfolgender Entwicklungs- und Prüfschritte
- Jede (Haupt-)tätigkeit bildet eine Phase
- Phase wird verlassen, wenn geprüftes und akzeptiertes Ergebnis vorliegt
- Lokale Iterationen möglich
- Es gibt eine Vielzahl so konstruierter Modelle
  
- Hauptproblem: Nicht-lokale Iterationen (erschwert Projektführung!)
- In dieser Form heute besser nicht mehr verwenden

# Wasserfall-Modell – 3: Nicht-lokale Iteration

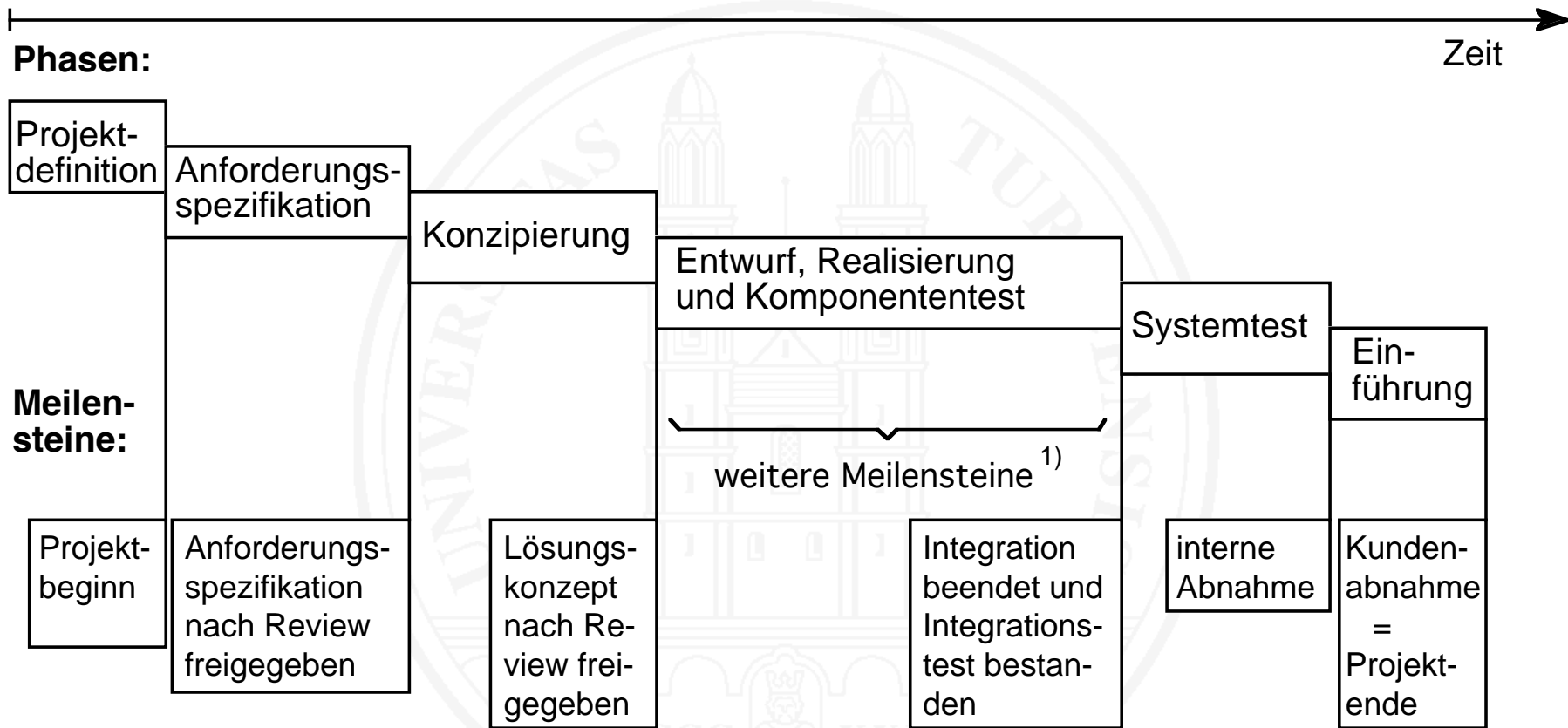


# Ergebnisorientierte Phasenmodelle – 1: Das Prinzip

---

- Orientiert sich am **Software-Lebenslauf**
- Grundidee: Unterteilt Entwicklung in **Sequenz aufeinanderfolgender Zeitabschnitte**
- Je **Phase**: Erarbeitung eines **Teils der Ergebnisse**
- Phase nach Ergebnis / vorherrschender Tätigkeit benannt
- **Keine Iteration**
- Jeder Phasenabschluss ein **Meilenstein**
- Vielzahl von Varianten und Namen
- Wird in der Literatur manchmal auch als Wasserfall-Modell bezeichnet

# Ergebnisorientierte Phasenmodelle – 2: Beispiel



1) zum Beispiel je Modul nach freigegebenem Entwurf und nach bestandenem Modultest



# Ergebnisorientierte Phasenmodelle – 3: Vor- und Nachteile

---

- + leicht verständlich, folgen natürlichem Lebenslauf
- + geeignet zur Projektführung
- + fördern planvolles Vorgehen
- + tragen zur frühzeitigen Fehlererkennung bei
  
- ignorieren die Software-Evolution
- lauffähige Systemteile entstehen erst spät:
  - lange Papier-Durststrecke
  - technische Risiken
  - Adäquatheitsrisiken
- System wird auf einen Schlag in Betrieb genommen

# Ergebnisorientierte Phasenmodelle – 4: Eignung

---

Ergebnisorientierte Phasenmodelle sind besonders geeignet für ...

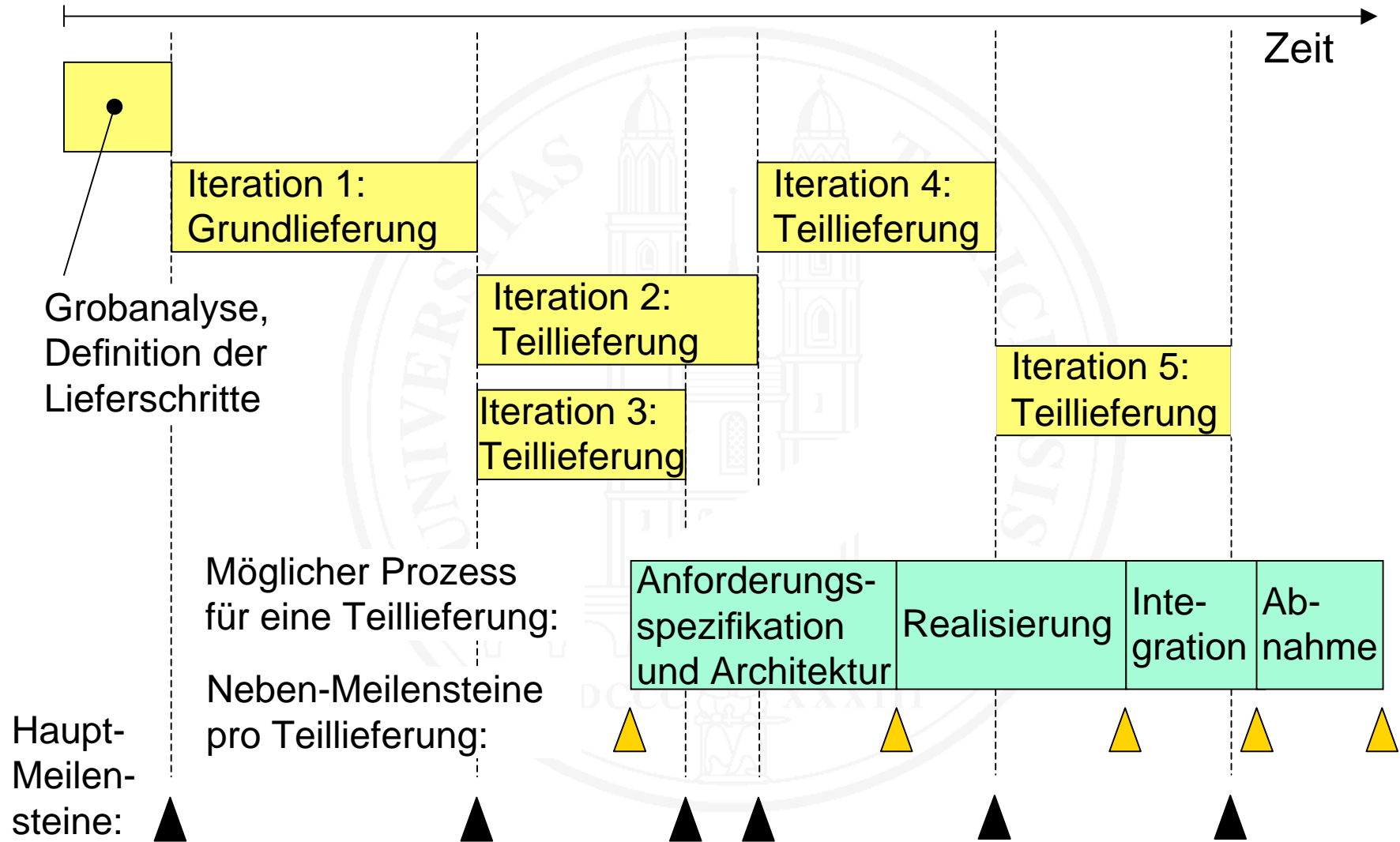
- ... nicht zu große Projekte ...
- ... mit genau definierbaren Anforderungen ...
- ... bei denen die Entwickler über das erforderliche Know-How verfügen ...
- ... und das Entwicklungsrisiko eher gering ist.

# Wachstumsmodelle – 1: Das Prinzip

---

- Leitbild: Software-**Evolution**
- System wird **nicht konstruiert**, sondern **wächst**
- Verschiedene Namen: Versionen-Modell, evolutionäres Modell, inkrementelles Modell
- Grundidee: **Gliederung der zu liefernden Software in aufeinander aufbauende, betriebsfähige Lieferungen**
- Unterteilt Entwicklung in eine **Folge von Iterationen**
- Pro Iteration: **Vollständiges Teilergebnis** mit betriebsfähiger Software
- Lieferungen als **autonome Teilprojekte** organisiert

# Wachstumsmodelle – 2: Beispiel



# Wachstumsmodelle – 3: Planung

---

- Umfang und Reihenfolge der Lieferungen
- Bis zum gewünschten Endzustand
  
- Schrittweiser Ausbau nach verschiedenen **Merkmale**n möglich:
  - **Funktionalität**
  - **Komfort**
  - **Leistung**
  - **Verwendbarkeit**

# Wachstumsmodelle – 4: Kontinuierliche Integration

---

- Normales Wachstumsmodell: betriebsfähige Version des Systems nach jeder Teillieferung
- **Kontinuierliche Integration: Ständig eine betriebsfähige Referenzversion mit dem aktuellen Entwicklungsstand verfügbar**
- Referenzversion wird in kurzen Abständen (täglich, wöchentlich) neu erzeugt (“daily build”)
- Verkürzt die Rückkopplungszyklen und senkt damit die Fehlerkosten
- **Funktioniert** gut, wenn
  - Aufgabe in viele kleine, **autonome Teilaufgaben** gliederbar
  - **Aufwand** für Neuerzeugung der Referenzversion **gering**
- **Gefahr**, dass notwendige Restrukturierungen unterbleiben

# Wachstumsmodelle – 5: Vor- und Nachteile

---

- + modellieren das natürliche Verhalten großer Systeme
- + sehr geeignet zur Projektführung
- + frühe Entstehung lauffähiger Teile
  - + fördert die Motivation der Beteiligten
  - + lindert größte Nöte des Auftraggebers rasch
- + verkürzt die Rückkopplungszyklen
- + schrittweise einführbar
  
- Gefahr der Insel-Bildung
- Gefahr der Zerstörung von Strukturen und Konzepten durch schrittweisen Ausbau

# Wachstumsmodelle – 6: Eignung

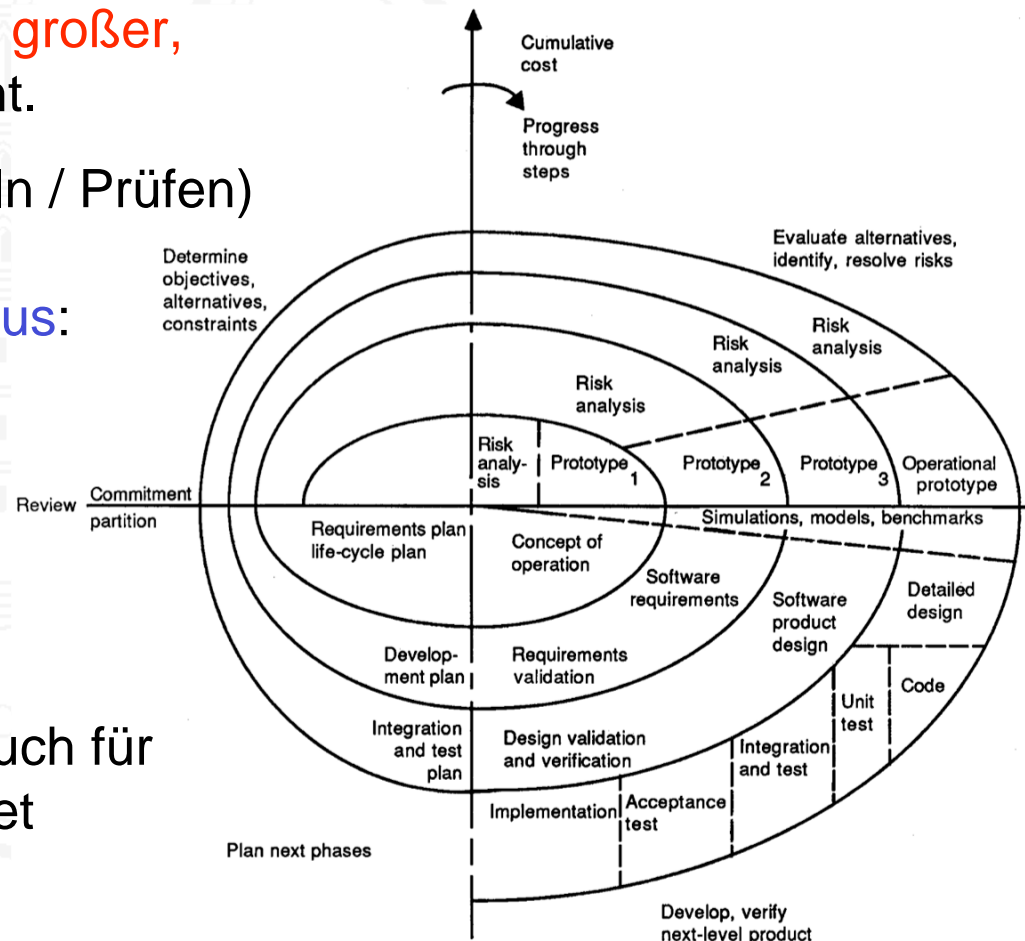
---

- ... das System **groß** ist und eine **lange Entwicklungszeit** hat ...
- ... **nicht die volle Funktionalität** sofort erforderlich ist ...
- ... ein Vorhaben **nicht vorab genau definierbar** ist ...
- ... ein **Basisprodukt** **schnell verfügbar** sein soll.



# Spiralmodell

- Weiterentwicklung des Wasserfall-Modells
- Vor allem für die Entwicklung **großer, risikoreicher** Systeme gedacht.
- Zweiteiliger Zyklus (Entwickeln / Prüfen) des Wasserfall-Modells wird ersetzt durch **vierteiligen Zyklus**:
  1. Planung
  2. Zielsetzung/Zielkorrektur
  3. Risiko-Untersuchung
  4. Entwicklung und Prüfung
- Name manchmal fälschlich auch für Wachstumsmodelle verwendet



# Agile Software-Entwicklung – 1: Idee

---

- **Idee:** Die **Verfahren** für die Entwicklung **von Kleinsoftware** so **auf große Software übertragen**, dass
  - es **erfolgreich** funktioniert
  - die Prozesse möglichst **einfach** und **wenig reglementiert** sind (Zeitersparnis, weniger Prozessbürokratie)
- Entwickelt in den späten Neunzigern; ab 2000 zunehmend populär

# Agile Software-Entwicklung – 2: Prinzipien

---

## Grundlegende **Prinzipien**:

- Je **kleiner** die **Teilaufgaben** und je **kürzer** die **Rückkopplungszyklen**, desto besser
- Der **Kunde** gestaltet das **Produkt** während seiner **Entstehung**
  - **Kundenvertreter** ist/sind **aktiv** am Projekt **beteiligt**
- Je **freier** und **konzentrierter** die Entwickler arbeiten können, desto besser
- Die **Qualität** wird an der **Quelle** sichergestellt
  - **“pair programming”**
  - **rigorose Komponententests**
- **Keine Arbeit auf Vorrat**

# Agile Software-Entwicklung – 3: Erfahrungen

---

- **Erfahrungen:** Agile Software-Entwicklung **funktioniert, wenn**
  - Auftraggebervertreter verfügbar, kompetent und entscheidungsbefugt
  - Problem in hinreichend viele kleine Teilaufgaben unterteilbar
  - Kleines Entwicklungsteam (oder mehrere parallele Teams)
  - Kompetenter Softwarearchitekt
  - Kontinuierliche Integration mit geringem Aufwand möglich
  - Konsequente Qualitätssicherung an der Quelle
  
- Sonst: **Absturz** ins **Chaos** und die **ad hoc Entwicklung**
  
- Vor- und Nachteile: Wie bei Wachstumsmodell, aber verstärkt

# Mini-Übung 13.2: Prozessmodell

---

Gegeben sei das Problem aus Aufgabe 2.5 im Skript.

- a) Welches Prozessmodell schlagen Sie für die Entwicklung der geforderten Software vor? Begründen Sie Ihre Entscheidung.
- b) Angenommen, Ihre Abteilungsleiterin entscheidet unabhängig von Ihrem Vorschlag, dass dieses Projekt nach einem Wachstumsmodell abzuwickeln ist. Skizzieren Sie einen Lieferplan, indem Sie Anzahl und Reihenfolge der Lieferungen festlegen und für jede Lieferung entscheiden, welche Komponenten der geforderten Lösung in welchem Ausbaurzustand darin enthalten sein sollen.

# Der Pflegeprozess

---

**Kontinuierlicher Prozess** mit folgenden Teilaufgaben:

- Erfassung der **Kundenbedürfnisse**
  - **reaktiv**: Problemmeldewesen
  - **proaktiv**: wohin geht die Reise (Märkte, Umwelt, Konkurrenz,...)
- **Analyse** der Kundenbedürfnisse
- **Planung** der Pflegearbeiten, Bildung von Arbeitspaketen
- **Durchführung** der anstehenden Arbeitspakete
- **Auslieferung** der Ergebnisse
- **Verwaltung** der Artefakte (Problemmeldungen, Arbeitspakete, Lieferungen...) [→ Kapitel 23: Konfigurationsmanagement]

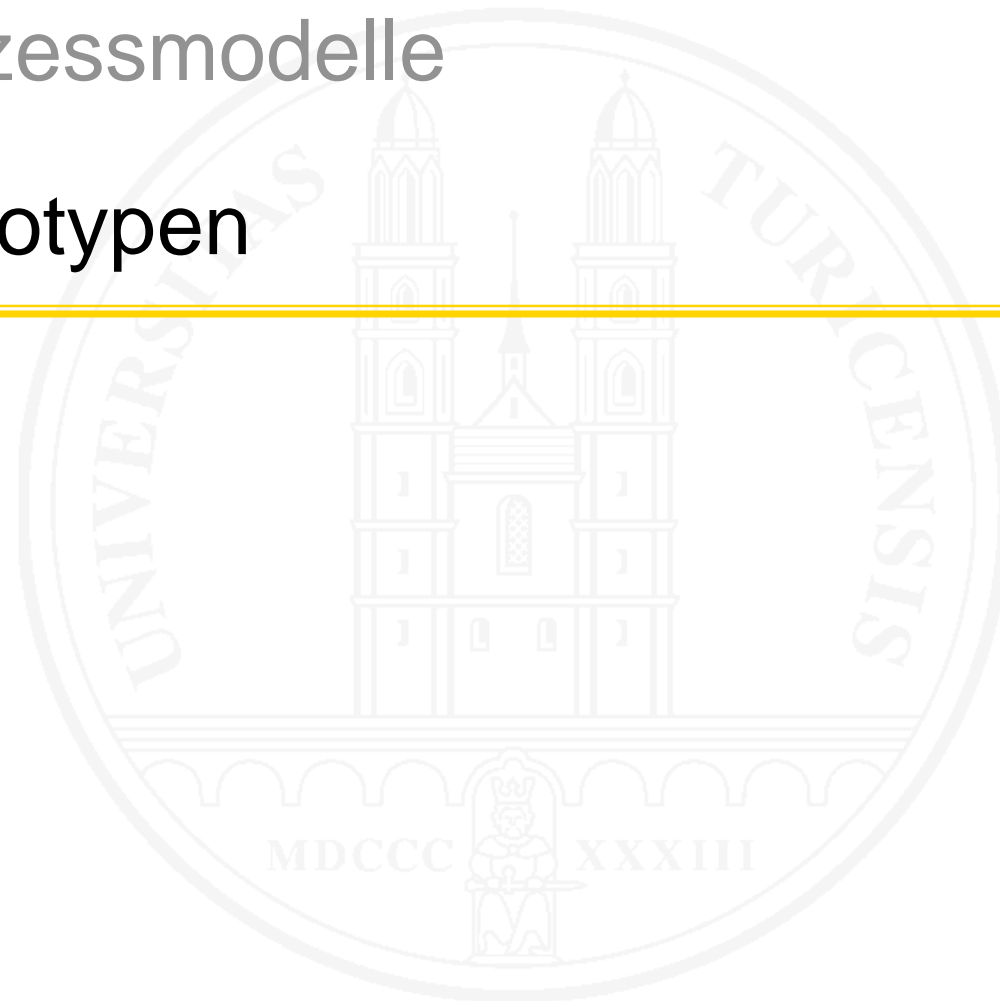
Siehe auch Kapitel 12: Software-Evolution und Reengineering

# 13.1 Grundlagen

# 13.2 Prozessmodelle

# 13.3 Prototypen

---



# Prototypen

---

Zentrales Mittel zur frühzeitigen Erkennung und Lösung von Problemen

**Prototyp (prototype)** – **Lauffähiges** Stück Software, welches **kritische Teile** eines zu entwickelnden Systems **vorab realisiert**.

**Prototyping** – Prozess der Erstellung und Nutzung von Prototypen

Drei **Arten** von Prototyping

- **Explorativ**
- **Experimentell**
- **Evolutionär**



# Exploratives und experimentelles Prototyping – 1

---

## Explorativ

- Klärung und Bestimmung von **Anforderungen**
- **Demonstrationsprototypen**: Demonstration der prinzipiellen **Machbarkeit** und **Nützlichkeit** von Systemideen
- **Prototyp im engeren Sinn**: **Erprobbares** und kritisierbares **Modell** einer geplanten Informatik-Lösung
  - **Angemessenheit** von **Anforderungen**
  - **Tauglichkeit** vorgesehener **Lösungen**

## Experimentell

- Entwicklung von **Labormustern** zur
  - Untersuchung der **Realisierbarkeit** kritischer Systemteile
  - **Bewertung** von **Entwurfalternativen**

# Exploratives und experimentelles Prototyping – 2

---

Demonstrationsprototypen, Prototypen im engeren Sinn und Labormuster sind **Wegwerf-Prototypen**:

- Dürfen **undokumentiert** sein
- Dürfen **schnelle**, softwaretechnisch **unsaubere** Lösungen verwenden
- **Wirtschaftlich**, wenn
  - aufgrund der gewonnenen Erfahrungen mehr Entwicklungskosten **eingespart** werden, als der Prototyp gekostet hat
  - ein **gefährliches Risiko** wesentlich **gemindert** werden kann
- Das tatsächliche Wegwerfen des Prototyps muss sichergestellt sein!

# Evolutionäres Prototyping

---

- Prototyp ist **Pilotsystem**
- Bildet den **Kern** des zu entwickelnden Systems
  - ⇒ **Kein Wegwerf-Prototyp**
  - ⇒ Muss nach allen **Regeln der Kunst** entwickelt werden
- Evolutionäres Prototyping entspricht einem **Wachstumsmodell**

# Prototyping und Prozessmodelle

---

Prototyping ist **kein** eigenständiges **Prozessmodell**

- *Demonstrationsprototypen*
  - Unterstützen Akquisition und Aufsetzen von Projekten
- *Prototypen im engeren Sinn* und *Labormuster*
  - Dienen zur Risiko-Minderung
  - In jedem Prozessmodell zu jedem Zeitpunkt einsetzbar
- Entwicklung *von Pilotsystemen*
  - eine Form des Wachstumsmodells

# Literatur

---

Siehe Literaturverweise im Kapitel 3 des Skripts.

Im Skript [M. Glinz (2005). *Software Engineering*. Vorlesungsskript, Universität Zürich] lesen Sie Kapitel 3.

Im Begleittext zur Vorlesung [S.L. Pfleeger, J. Atlee (2006). *Software Engineering: Theory and Practice*, 3rd edition. Upper Saddle River, N.J.: Pearson Education International] lesen Sie Kapitel 2 und Kapitel 11.1.

