1 02_Software-Entwicklungsprozess

1.2 Unified Process

- objektorientierter Software-Entwicklungsprozess
- **UP** ist ein (!) **Standardprozess** für objektorientierte Software
- verwendet **UML** als Notationssprache zur Beschreibung von Ergebnissen (Artefakten)

Eigenschaften/ 3 Prinzipien

1) **Anwendungsfallgesteuert (use-case driven)**
2) **Architekturzentriert (architecture-centric)**
3) **Iterativ und inkrementell (iterative and incremental)**

- UP-Prinzipen liegt als **Grundidee** die **Risikominimierung** zugrunde:
  - frühzeitiges Erkennen (und Lösen) der Projektrisiken, indem die Aufgaben mit dem **größten Risiko so früh** wie möglich angegangen werden

1.2.1 Anwendungsfallgesteuerter Prozess (use-case driven)

Da das System aus Sicht des Anwenders entwickelt wird und dessen Anforderungen im Vordergrund stehen. Die Anforderungen sind als Use-Cases beschrieben und diese steuern den Entwicklungsprozess. Bedeutet, dass die kritischen Use-Cases zuerst analysiert und implementiert werden

UP ist **anwendungsfallgesteuert**

- Idee: Software-System aus der **Perspektive des Benutzers** entwickeln!

  ⇒ Was soll das System für den Benutzer leisten?

1. Use Cases (Anwendungsfälle) beschreiben/ spezifizieren die **Anforderungen**

2. Use-Case driven: Use Cases **steuern** den Entwicklungsprozess

- Use Cases
  - wirken auf **Projektorganisation**, **Aufwandschätzung**, **Vertrag**
  - bestimmen verschiedene **Systemstufen** (= Inkrement, siehe später)
  - » Menge von Use Cases legt Funktionsumfang eines Release fest
  - » geben Meilensteine zur Projektsteuerung vor
    - sind Ausgangspunkt vieler Ergebnisse bei SW-Entwicklung

- Idee: **wichtigste (Kern-)** Use Cases (Anwendungsfälle) zuerst realisieren, da in denen das größte fachliche Risiko steckt !

1.2.2 Architekturzentrierter Prozess

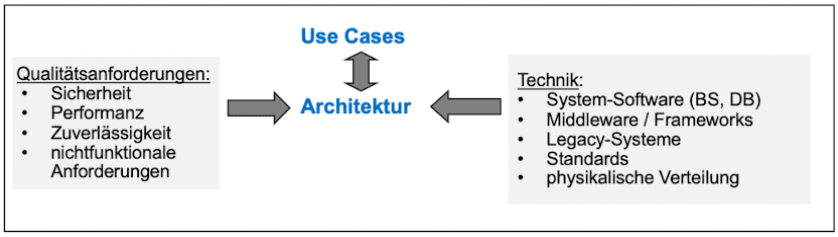
Software-Architektur

- Was ist die **Architektur** eines Software-Systems?
  - Ziel: **Strukturierung des Gesamtsystems** (⇒ dient als exakte Vorgabe für Implementierung)
  - Spezifikation von **Schichten**, **Teilsystemen**, **Komponenten** mit **Schnittstellen** und zugehöriger **Interaktion**

Wie wird eine Architektur dargestellt?

- durch eine Vielzahl von **Modellen** und **Dokumenten** **(nicht der Source-Code)**
- bieten unterschiedliche Sichten: Struktur, Verhalten, Verteilung auf Hardware etc.

Einflussfaktoren auf Architektur



- jeder Gegenstand hat Form (Architektur) und Funktion (Use Cases), die sich gegenseitig beeinflussen

1. Use Cases beeinflussen ggf. Architektur

- Anforderungen der Use Cases wirken auf technische Infrastruktur, Verteilung, Schnittstellen etc.

2. Architektur beeinflusst ggf. die Use Cases

- Architektur führt zu neuer Bewertung/ Priorisierung der Use Cases
  - welche Use Cases sind für Architektur kritisch (= technisches Risiko)?
  - ggf. ergeben sich Änderungen an den Use Cases und ihrer Prioritäten

Architekturzentrierung

Neben der Anwendersicht werden die Technische Aspekte ebenfalls berücksichtigt und haben einen Einfluss auf die Architektur und der Priorisierung der Use Cases

Unified Process ist **architekturzentriert (architecture-centric)**

- Idee: neben **Anwendersicht (fachliche Sicht)** wird **parallel** auch **technische Sicht** auf das System betrachtet

Umsetzung im Unified Process

- früh (= bereits in Elaboration-Phase, siehe später) wird ein **Mini-System** (1. Version bzw. technischer Durchstich) mit allen erforderlichen Modellen realisiert, um **technisches Risiko zu minimieren**
- **Architekturmodelle kontinuierlich während Projektlaufzeit pflegen bzw. anpassen**

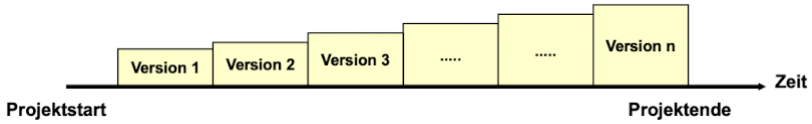
UP adressiert dadurch fachliche Sicht (welche Funktionen soll das System bieten) und **technische Sicht** (wie soll das System gebaut werden) **gleichzeitig**

1.2.3 Iterativer und inkrementeller Prozess

Inkrementell

System wir stufenweise weiterentwickelt!

- **UP ist inkrementeller (incremental) Prozess**
  - Idee: komplexe Software wird **stufenweise** erstellt

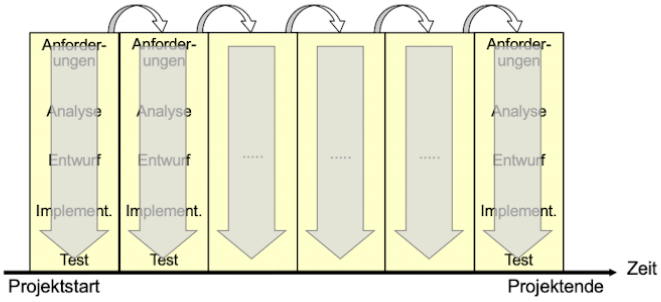


- jede neue Systemstufe erweitert vorhergehende Stufe (= Version) um zusätzliche Funktionalität,
  - d.h. neue Stufe realisiert weitere Use Cases
  - d.h. System „wächst“ stufenweise
- **für jede Stufe** (Version, Release, Build) gibt es ein **Meilenstein**

iterativ

Bei der Entwicklung werden alle Arbeitsschritte andauernd wiederholt!

- **UP ist iterativer (iterative) Prozess**
  - Idee: zur Realisierung jeder Systemstufe werden die **selben Arbeitsschritte** erneut durchgeführt
    Anforderungsanalyse ⇒ Analyse ⇒ Entwurf ⇒ Implementierung ⇒ Test



- Realisierung einer Stufe führt ggf. zu Änderungen in einer vorhergehenden Stufe:
  neue Erkenntnisse erweitern/ revidieren frühere Ergebnisse

Bewertung der inkrementellen Entwicklung

Vorteile

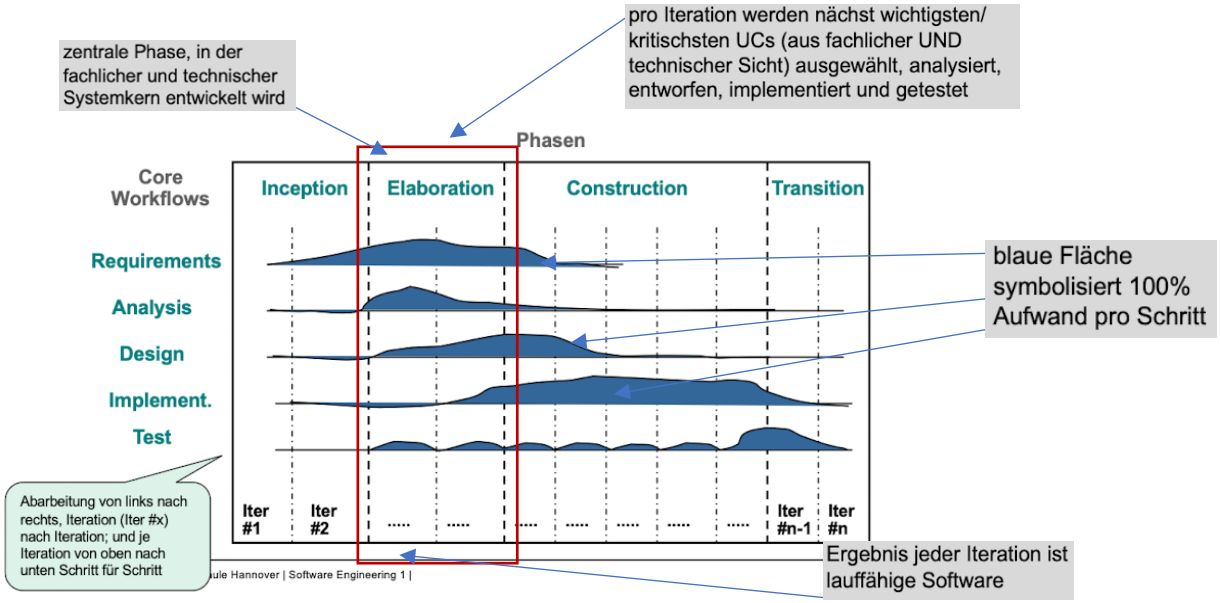
- **besseres Risikomanagement**
  - fachliche und technische Probleme treten schnell zu Tage
  - risikobehaftete (fachliche und technische) Aufgaben zuerst angehen
- **besseres Projekt-Controlling**
  - tatsächliche Aufwände liegen frühzeitig vor
  - frühe Qualitätsüberprüfung möglich
- **bessere Projektsteuerung**
  - anpassen der Stufen bei Budget-/ Zeitproblemen
- **Vorteile für Kunden**
  - erleichtert Benutzerbeteiligung auf Basis der realisierten Stufen
  - ermöglicht frühen produktiven Einsatz

Nachteile

- **vertragliche Gestaltung schwieriger**, da Rückmeldungen der Benutzer zu Änderungen der Anforderungen führen (sollen) →Feedback vom Kunden
- **Abhängigkeiten** zwischen Teilsystemen erst **spät erkennbar**:
  - zusätzlicher Aufwand für Redesign
- **i.d.R. höherer Aufwand** aufgrund von redundanten Aktivitäten, späten Änderungen, Provisorien etc.
- kann zur (zu) schnellen Implementierung ohne Gesamtkonzept verführen

1.3 UP - Vorgehen

Unified Software Development Process



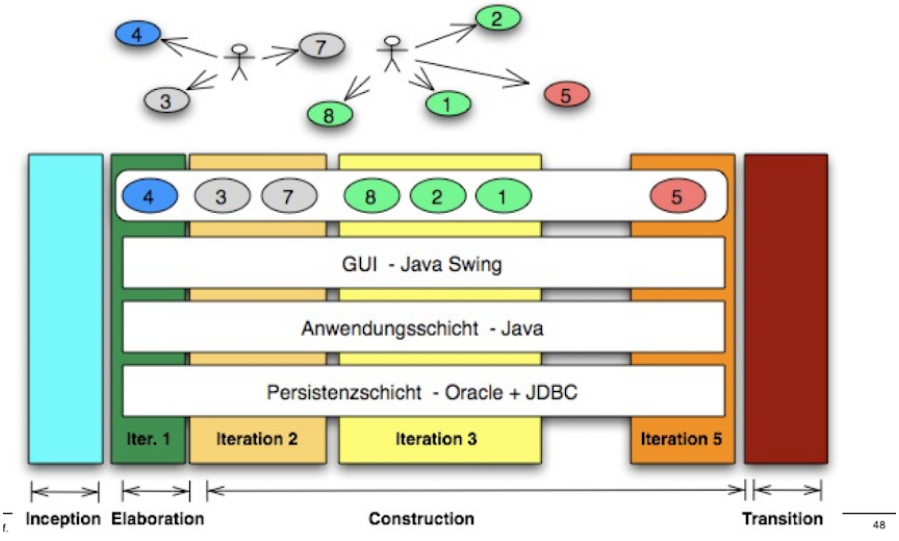
Core Workflows (Arbeitsschritte):

- Schritte der Iterationen ändern sich während der Projektphasen

1) **Anforderungen:**
   - Anforderungsmodell bestehend aus einer Systembeschreibung, einem Domänenmodell, der Anforderungen wie Use-Case Diagramm, Use-Case Beschreibung, nichtfunktionale Anforderungen und ggf. einen GUI-Prototyp
2) **Analyse:**
   - Analysemodell bestehend aus einem Klassendiagramm, Sequenzdiagramm, Paketdiagramm und ggf. einem Aktivitätsdiagramm & Zustandsdiagramm
3) **Design:**
   - Verteilungsdiagramm mit der logischen Struktur sowie Architektur- und Entwurfsmuster festlegen
4) **Implementierung:**
   - Integrationsstrategie festgelegt, Implementierungsmodell und das Build-Management
5) **Test:**
   - Testplan und Testfälle mit Test- und Fehlerbericht

Unified Process ist Use-Case Driven, architekturzentr., iterativ u. inkrementell

- Use Cases bestimmen die Iterationsstufen
- jede Iteration betrachtet nur ihren Teil der GUI, Klassen und DB-Tabellen



Projektphasen:

Projektphasen (I) – Inception (Konzeptionsphase)

- Projektplanung (Organisation, Kosten, …)
- Sammeln der kritischen Use-Cases
- Projektinfrastrukturvorbereitung
- Erste Gedanken zur Systemarchitektur sammeln

→ **kein lauffähiges System**

Projektphasen (II) – Elaboration (Ausarbeitungsphase)

- sammeln aller Use Cases
- Analyse und Implementierung der kritischen Use Cases
- Entwurf und Implementierung des Kerns der Architektur

→ **Anforderungen und Analyse fertig, Kern der Architektur implementiert** und **erste lauffähige Version des Systems**

Projektphasen (III) – Construction (Konstruktionsphase)

- Alle Use Cases analysieren und implementieren
- inkrementell System weiterentwickeln der wichtigsten Use-Cases
  - Dabei pro Iteration Use-Cases neu priorisieren

→ **System fertig entworfen, implementiert und getestet**

Projektphasen (IV) – Transition (Umsetzungsphase)

- Abnahme des Systems
- Abschließende Arbeiten
- Deployment

→ **abschließende Arbeiten, Installationen und Inbetriebnahme**

1.4 Zusammenfassung

Software-Entwicklungsprozess

- **geordneter** und **systematischer Prozess** ist unabdingbar für Projekterfolg
- Prozess bietet Systematik der geordneten Projektdurchführung
- **Arbeitsschritte**
  - **Anforderungsanalyse, Analyse, Entwurf, Implementierung, Test**
  - Objektorientierung durchgängiges Konzept aller Schritte
- **Unified Process** ist Standardprozess für OO-Softwareentwicklung
  1. **anwendungsfallgesteuert** (Anwendersicht)
  2. **architekturzentriert** (technische Sicht)
  3. **inkrementell** und **iterativ** (System entsteht in Stufen)

  ⇒ **Risikominimierung** als grundlegende Idee