

Bachelor of Science (BSc) in Informatik  
Modul Software-Entwicklung 1 (SWEN1)

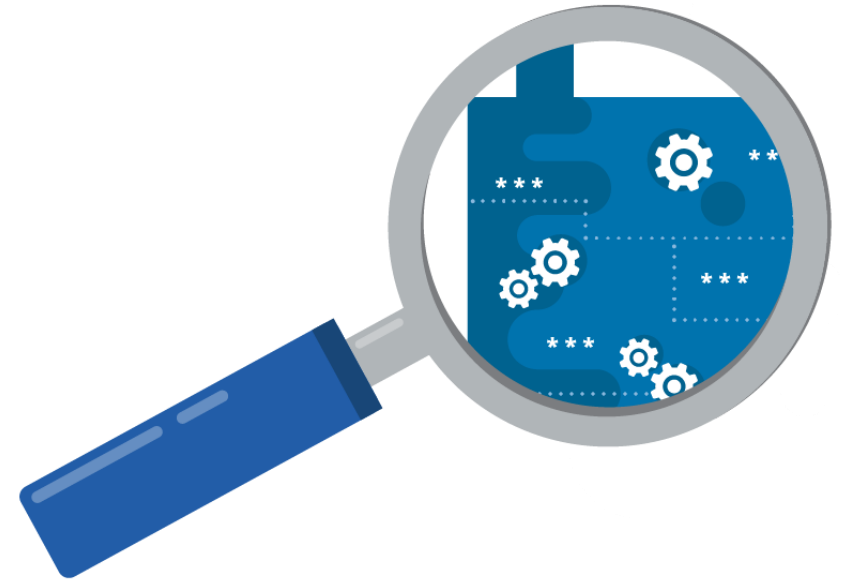
# LE 01 – Einführung und Überblick

SWEN1/PM3 Team:  
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

Ausgabe: HS24

# Um was geht es?

- Was lerne ich in diesem Modul?
- Was ist ein Softwareentwicklungsprozess und welche Artefakte werden im Laufe eines Projektes erstellt?
- Was und warum modelliere ich mit der UML in der Analyse und dem Design?



# Lernziele LE 01 – Einführung und Überblick

---

- Sie sind in der Lage:
  - die Lernziele, Inhalte und den Ablauf des Moduls zu erläutern,
  - die Charakteristiken von Wasserfall, iterativ-inkrementellen und agilen Softwareentwicklungsprozessen darzustellen,
  - den Zweck und den Nutzen von Modellen in der Softwareentwicklung zu diskutieren,
  - die in der Vorlesung thematisierten Artefakte in einem iterativ-inkrementellen Prozess zu illustrieren und einzuordnen.

# Agenda

---

1. Lernziele, Inhalte und Ablauf des Moduls
2. Überblick Software Engineering und Softwareentwicklungsprozesse
3. Modelle und Modellierung mit der UML
4. Ablauf, Rollen und Artefakte in einem iterativ-inkrementellen Prozess
5. Wrap-up und Ausblick

# Lernziele SWEN1 (1/2)

- Sie sind in der Lage:
  - für einen vorgegebenen, **iterativ-inkrementellen Softwareentwicklungsprozess** den Ablauf und die **Artefakte** zur **Entwicklung einer objektorientierten Softwareapplikation** zu erläutern,
  - die **Begriffswelt des Anwenders** durch geeignete Vorgehensweisen zu erfassen und zu einer fachlichen Terminologie **zu verdichten (Domänenmodell)**,
  - eine **Softwareapplikation** sinnvoll abzugrenzen,
  - **systematisch die funktionalen Anforderungen mit Use Cases** sowie **Qualitätsanforderungen und Randbedingungen** zu erheben und zu kommunizieren,

# Lernziele SWEN1 (2/2)

- basierend auf den Anforderungen eine **geeignete Softwarearchitektur** und ein **objektorientiertes Design** - Klassen mit Verantwortlichkeiten - für die darin enthaltenen Komponenten **der fachlichen Logik zu entwerfen**,
- für die **Modellierung und Kommunikation von Artefakten** im Softwareentwicklungsprozess **standardisierte Notationen** (wie UML) zu benutzen,
- bewährte **Analyse, Architektur und Design Patterns** adäquat für eine Problemstellung einzusetzen.

# Themen und Ablauf des Moduls SWEN1

LE#	Thema
01	Einführung und Überblick
02	Anforderungsanalyse I
03	Anforderungsanalyse II
04	Domänenmodellierung
05	Quiz 1: Analyse Softwarearchitektur und Design I
06	Softwarearchitektur und Design II
07	Use-Case Realisierung
08	Entwurf mit Design Patterns I
09	Entwurf mit Design Patterns II
10	Implementation, Refactoring und Testing
11	Quiz 2: Design Vertiefung 1: Thema gemäss Abmachung
12	Vertiefung 2: Thema gemäss Abmachung
13	Vertiefung 3: Thema gemäss Abmachung
14	Wrap-up der Vorlesung

# Motivation SWEN1 (1/2)

- Analyse- und Entwurfs-Kompetenzen sind zentral für BSc Informatik,
  - um grössere und komplexere Softwaresysteme entwickeln zu können.
- Zur Analyse-Kompetenz gehören vor allem der Wille und die Fähigkeit,
  - mit Aufgabenstellern und zukünftigen Systemnutzern die Anforderungen auszuhandeln, zu kommunizieren und zu dokumentieren,
  - sich schnell in neue Anwendungskontexte einarbeiten zu können.
- Entwurfs-Kompetenzen
  - Sie sollten bekannte Problemstellungen im Anwendungskontext erkennen können und mit den zugehörigen Lösungsmustern vertraut sein.
  - Sie sollten Inkonsistenzen erkennen und mit unklaren Anforderungen umgehen können.



# Motivation SWEN1 (2/2)

- Entwurfs-Kompetenzen (Forts.)
  - Als BSc in Informatik wird von Ihnen erwartet, dass Sie **komplexe Domänen modellieren und grosse Anwendungsprobleme** durch geeignete Schnittstellen in Teilprobleme zerlegen können.
  - Sie benötigen die Fähigkeit, Systeme aus Hard- und Software so zu entwerfen, dass sie die **Anforderungen vollständig erfüllen**.
  - Hierfür ist **Abstraktionsfähigkeit** genauso unverzichtbar wie solide Kenntnisse in der **Softwarearchitektur**.
  - Zentral ist bei Ihrem Entwurf ist die **Umsetzung nicht funktionaler Anforderungen, wie Sicherheit, Performanz, Skalierbarkeit, Wartbarkeit, Erweiterbarkeit und Zuverlässigkeit**.

# Didaktisches Konzept (1/2)

## **Vorlesung** (1.5 Lektionen)

- Vermittlung der Grundlagen anhand einer Zusammenfassung
- Ein umfangreicher Foliensatz steht für die nachfolgenden Praktika zur Verfügung

## **Praktikum** (2.5 Lektionen integriert in der Vorlesung)

- Verschiedene Aufgaben mit Bezug zu PM3, Mini-Fallstudien mit Lernaufgaben, Präsentationen zu Themen aus der Vorlesung

## **Selbststudium**

- Wissenssicherung bzw. weitergehende Aufgaben zur Vorlesung und dem Praktikum
- Weitergehende empfohlene Lektüre und abgegebene Fachartikel



# Didaktisches Konzept (2/2)

## **Vorlesung** (ca. 1.5 Lektionen)

- 15 Min.: Repetition letzte Lerneinheit
- 15 Min.: Besprechung der fakultativen Lernaufgaben und der Wissenssicherung
- 40 Min.: Input Dozent (Einleitung, Ablauf, Motivation, Begriffe)

## **Praktika** (ca. 2.5 Lektionen)

- 10 Min: Aufgaben kurz erklären und verteilen
- 50 Min.: Bearbeiten der Aufgaben durch die Studierenden
- 45 Min.: Präsentationen der Resultate durch die Studierenden
- 10 Min.: Wrap-up der Vorlesung

## **Selbststudium** (2-4 Lektionen Wissenssicherung)

Weiterführende Aufgaben aus dem praktischen Teil  
oder Fragen zur Vorlesung (separat beschrieben)



# Leistungsnachweise (1/3)

---

- **Während des Semesters (max. 13 Lerneinheiten)**
  - 30% des gesamten Leistungsnachweises
  - Bestehend aus Leistungsnachweisen pro Lerneinheit: ca. 2/3 von 30%
  - 2 Quizzes: ca. 1/3 von 30%
- **Semesterschlussprüfung (SEP)**
  - 70% des gesamten Leistungsnachweises

# Leistungsnachweise (2/3)

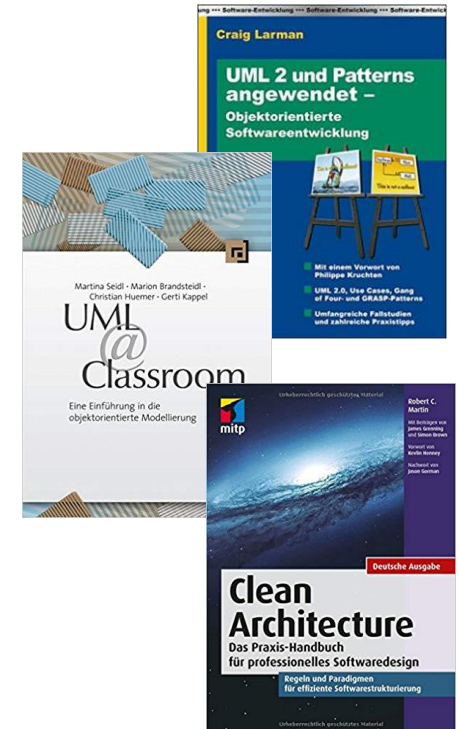
- **Pro Lektionseinheit** maximal 3 Punkte pro Studierenden
  - Präsentationen Lernaufgaben / Wissenssicherung / Inhalte
    - Die Vorbereitung, Durchführung und Präsentation dieser Beiträge durch die Studierenden werden mit Punkten bewertet.
    - Sie dienen der aktiven Aufarbeitung der Inhalte und der engeren Verbindung zu PM3.
  - Punkteschema: 0-3 Punkte
    - Punkte gibt es nur bei Anwesenheit im Klassenzimmer.
      - Bei Abwesenheiten wie Krankheit (Arztzeugnis), Militär, etc. wird die Anzahl Punkte normalisiert.
    - Punkteschema: 0 - nicht anwesend oder unbrauchbar, 1 - brauchbar, 2 - gut, 2+ besser als gut.
    - Falls keine Präsentation in Folge grosser Klassen und begrenzter Zeit möglich ist, können die Abgaben in MS Teams hochgeladen werden. Die Punkte werden nachfolgend erteilt.
    - Bei zwei Präsentationen/Abgaben pro Studierenden pro LE wird der bessere Versuch gewertet.
    - Einige Präsentationen finden als Gruppenpräsentation statt. Dies können PM3-Gruppen oder auch andere Gruppenzusammensetzung sein. Es präsentiert jeweils die ganze Gruppe.

# Leistungsnachweise (3/3)

- **2 Quizzes** in der Vorlesung während dem Semester zur formativen Lernkontrolle (jeweils 10 Multiple-Choice-Fragen, 15 Min.)
  - 1 Quiz zur Anforderungsanalyse
  - 1 Quiz zur Softwarearchitektur und Design
- **Semesterendprüfung (SEP)** (90 Min.)
  - Umfang: Vorlesung, abgegebene Unterlagen, Aufgaben aus dem integrierten Praktikum und der Wissenssicherung
  - Prüfungsform: Moodleprüfung. Einige Aufgabe erfordern den Upload eines PDF's oder Scans mittels Smart-Phone Kamera



- Empfohlene Lektüre  
(Bücher sind in DE und EN erhältlich)
  - Larman, C.: UML 2 und Patterns angewendet, mitp Professional, 2005
  - Seidel, M. et al.: UML @ Classroom: Eine Einführung in die objektorientierte Modellierung, dpunkt.verlag, 2012
  - Martin, R. C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design, mitp Professional, 2018
- Das Buch von Larman wird insbesondere Studierenden empfohlen, die noch wenig Vorwissen in der Softwareentwicklung haben.
- Weitergehende Literatur zur Vertiefung eines Themas ist jeweils in der entsprechenden Präsentation aufgeführt.



# Agenda

---

1. Lernziele, Inhalte und Ablauf des Moduls
- 2. Überblick Software Engineering und Softwareentwicklungsprozesse**
3. Modelle und Modellierung mit der UML
4. Ablauf, Rollen und Artefakte in einem iterativ-inkrementellen Prozess
5. Wrap-up und Ausblick



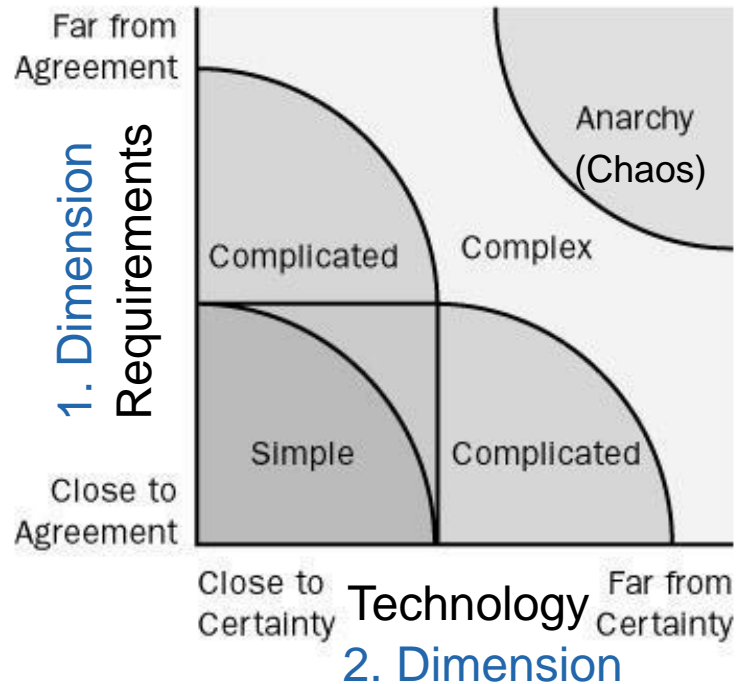
# Software Engineering (2/2)

- Wesentliche **Disziplinen** des Software Engineerings sind:
- **Kerndisziplinen**
  - Anforderungsanalyse (engl. requirements engineering)
  - Softwarearchitektur und Design (engl. software architecture and design)
  - Implementierung (engl. software construction)
  - Softwaretest (engl. software testing)
  - Softwareverteilung (engl. software deployment)
  - Softwareeinführung (engl. software rollout)
  - Wartung/Pflege (engl. software maintenance)
- **Unterstützungsdisziplinen**
  - Projektmanagement (engl. project management)
  - Konfigurationsmanagement (engl. configuration management)
  - Qualitätsmanagement (engl. quality management)
  - Risikomanagement (engl. risk management)

# SWE-Prozess: Motivation und Zielsetzung

- Der Softwareentwicklungsprozess umfasst den gesamten **Produktlebenszyklus**
  - Von der ersten Idee bis zur Ausmusterung der Softwarelösung
- Warum ist eine **strukturierte Softwareentwicklung** notwendig?
  - Strukturierung der wichtigsten Aktivitäten in der Softwareentwicklung
    - Anforderungsanalyse, Software-Design, Implementation, Test
  - Früherkennung von Fehlern
    - Je später Fehler entdeckt werden, desto mehr kostet ihre Behebung (Faktor 10 bis 100)
  - Minimierung von Risiken
    - Projektrisiken sollten so früh wie möglich angegangen werden

# Wie können Software-Entwicklungs-Probleme klassifiziert werden?



## 3. Dimension



Skills, Intelligence Level, Experience,  
Attitudes, Prejudices

Quelle: Agile Project Management with Scrum, Ken Schwaber, 2003

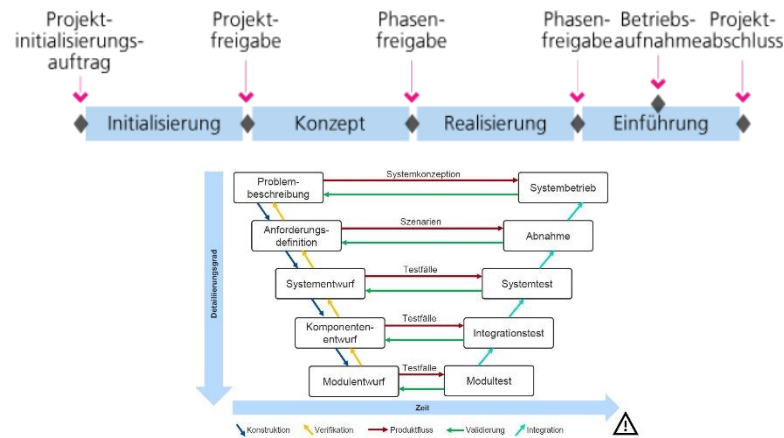
# Ausgewählte Vorgehensmodelle (Familien) zur Lösung von Software-Problemen

---

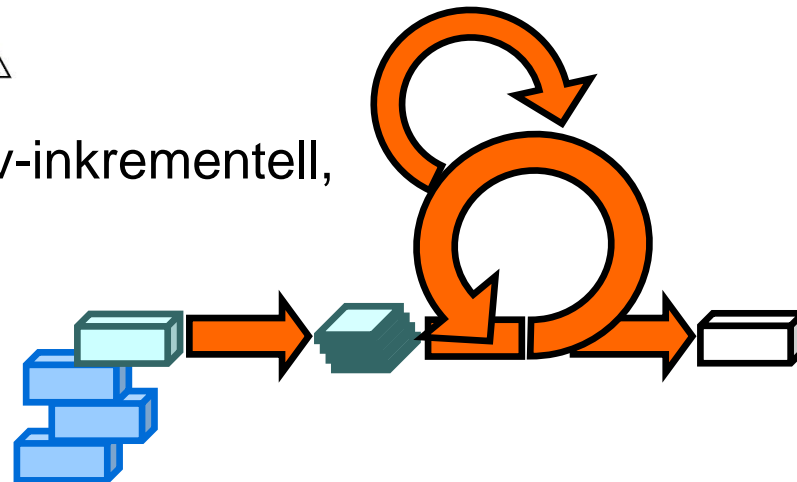
- Code and Fix
- Wasserfallmodell
- Iterative und inkrementelle Modelle

# Charakterisierung von verbreiteten Software-Prozessmodellen

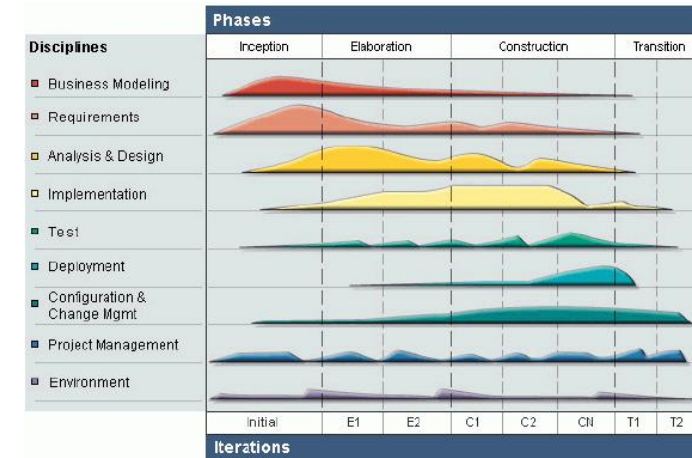
**Hermes:** Wasserfall/V-Modell (historisch),  
definiert, kaum empirisch (Tailoring)



**Scrum:** iterativ-inkrementell,  
empirisch



**(R)UP:** iterativ-inkrementell,  
definiert oder empirisch (Tailoring)



# Agenda

---

1. Lernziele, Inhalte und Ablauf des Moduls
2. Überblick Software Engineering und Softwareentwicklungsprozesse
3. **Modelle und Modellierung mit der UML**
4. Ablauf, Rollen und Artefakte in einem iterativ-inkrementellen Prozess
5. Wrap-up und Ausblick

# Modelle und Modellierung im Software Engineering

- Ein **Modell** ist ein konkretes oder gedankliches **Abbild** eines vorhanden Gebildes oder **Vorbild** für ein zu schaffendes **Gebilde** (hier **Softwareprodukt**).
- Das **Original** ist das abgebildete oder zu schaffende Gebilde.
- **Modellierung** gehört zum **Fundament des Software Engineerings**
  - Software ist vielfach (immer?) selbst ein Modell
  - Anforderungen sind Modelle der Problemstellung
  - Architekturen und Entwürfe sind Modelle der Lösung
  - Testfälle sind Modelle des korrekten Funktionierens des Codes usw.
- «Die Artefakte der Software-Entwicklung sind Modelle» (Jürgen Ebert)
- *Wer Software entwickelt oder pflegt, braucht solides Wissen und Können in Modellierung!*

# Wozu Modelle?

- Verstehen eines Gebildes
- Kommunizieren über ein Gebilde
- Gedankliches Hilfsmittel zum Gestalten, Bewerten oder Kritisieren eines geplanten Gebildes oder von Varianten davon
- Spezifikation von Anforderungen an ein geplantes Gebilde
- Durchführung von Experimenten, die am Original nicht durchgeführt werden sollen, können oder dürfen
- Aufstellen / Prüfen von Hypothesen über beobachtete oder postulierte Phänomene



# Wieviel Modellierung braucht es in einem Softwareprojekt?

Analogie: Planung und Realisierung einer

Hundehütte



vs. Haus



vs. Wolkenkratzer

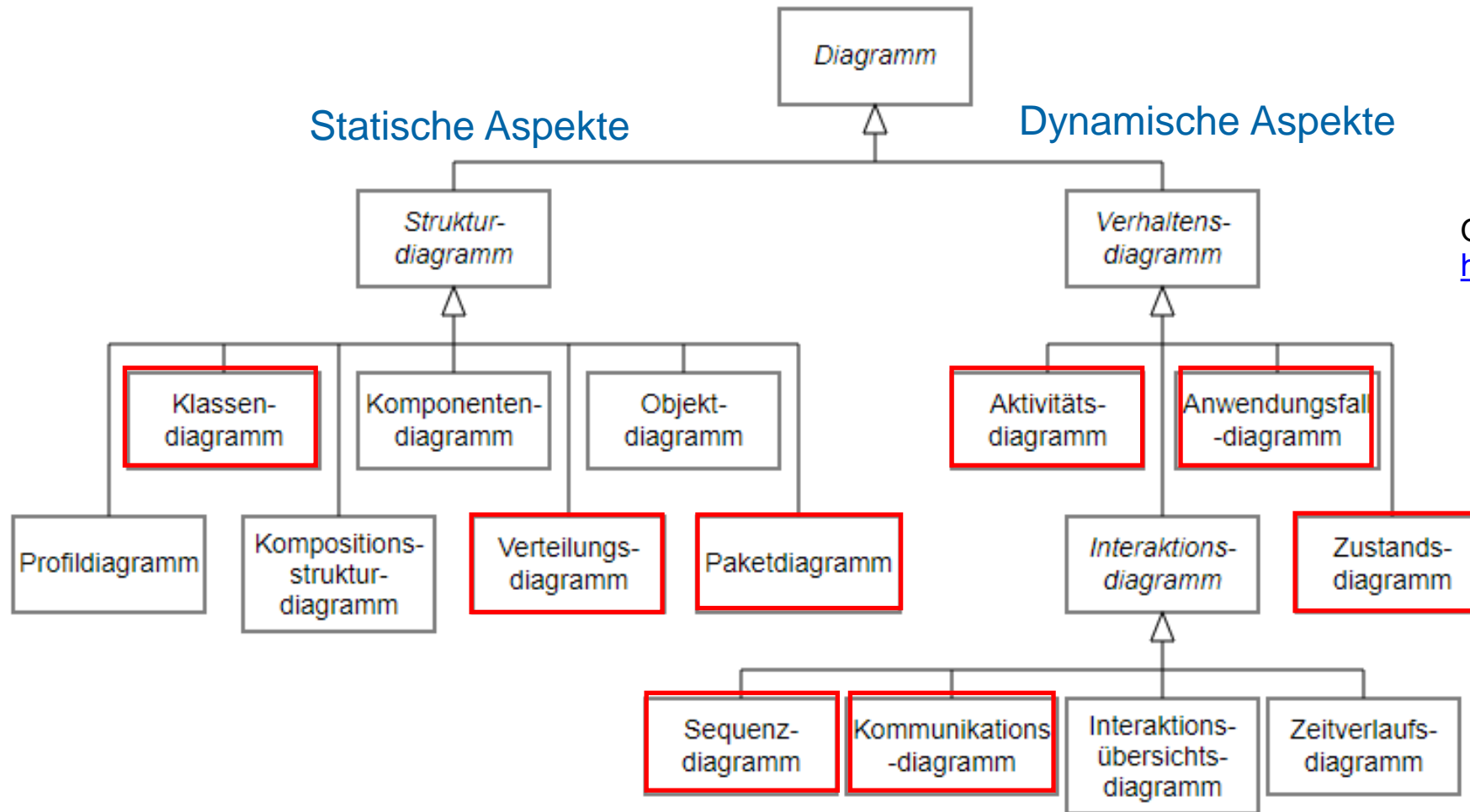


*Wieviel Zeremonie und Aufwand für die Planung, Modellierung, Dokumentation und Realisierung in einem Projekt benötigt wird, hängt von der Problemstellung ab!*

# Die Diagramme der UML



Quelle: UML Specification,  
<https://www.omg.org/spec/UML/>



  für die Modellierung in SWEN1 relevant

# Gebrauch der UML (nach Martin Fowler)

- **UML as a Sketch**

- Informelle und unvollständige Diagramme (z.T. von Hand gezeichnet), um schwierige Teile des Problems oder der Lösung zu verstehen und zu kommunizieren
- Die agile Community bevorzugt diese Anwendungsart von UML

- **UML as a Blueprint**

- Relativ detaillierte Analyse und Design-Diagramme für Code-Generierung oder um existierenden Code besser zu verstehen
- Klassische UML-Tools für ein Forward- und Reverse-Engineering (Round-trip)

- **UML as a Programming Language**

- Komplete, ausführbare Spezifikation eines Software-Systems in UML
- MDA-Tools zur Modellierung und Generierung



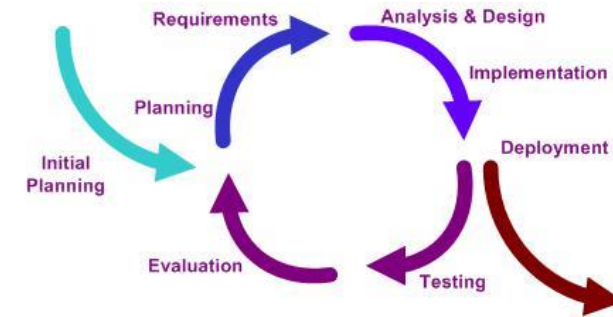
# Agenda

---

1. Lernziele, Inhalte und Ablauf des Moduls
2. Überblick Software Engineering und Softwareentwicklungsprozesse
3. Modelle und Modellierung mit der UML
4. **Ablauf, Rollen und Artefakte in einem iterativ-inkrementellen Prozess**
5. Wrap-up und Ausblick

# Angewendeter iterativ-inkrementeller Softwareentwicklungsprozess in SWEN1/PM3

- Der Softwareentwicklungsprozess wurde so angepasst (engl. tailoring), dass die **wesentlichen Artefakte** in einem **Softwareprojekt** im Kontext eingeführt werden können.
- Die Software wird in **Iterationen** entwickelt (2 Wochen Rhythmus).
- Jede Iteration hat ein Ziel und wird nach Abschluss reviewed.
- Es gibt **drei Meilensteine**, die im Projektverlauf ein besonderes Ereignis darstellen bzw. den Abschluss einer Phase: Projektskizze (M1), Lösungsarchitektur (M2) und Prototyp (M3)
- In **jeder Iteration** werden **Anforderungen, Analyse & Design, Implementation und Testing** gemacht (Software entsteht in Inkrementen).
- Der angewendete Softwareentwicklungsprozess und das **Projektmanagement eines iterativ-inkrementellen Projektes** wird in PM3 noch detaillierter erklärt.



# Agenda

---

1. Lernziele, Inhalte und Ablauf des Moduls
2. Überblick Software Engineering und Softwareentwicklungsprozesse
3. Modelle und Modellierung mit der UML
4. Ablauf, Rollen und Artefakte in einem iterativ-inkrementellen Prozess
5. **Wrap-up und Ausblick**

# Wrap-up

- Zur Entwicklung grösserer Softwareapplikationen werden solide **Analyse- und Entwurfskompetenzen** benötigt, die in dieser Vorlesung vermittelt werden.
- Im Software Engineering wurden im Laufe der Zeit **verschiedene Softwareentwicklungsprozessmodelle** vorgeschlagen.
- Heutige, gängige agile Softwareentwicklungsprozesse verwenden ein **iterativ-inkrementelles Vorgehensmodell**.
- Dieses Vorgehensmodell erlaubt auch **komplexe Problemstellungen** in überschaubaren **kleinen Schritten (Iterationen)** unter Einhaltung von Umfang, Kosten, Zeit und Qualität zu entwickeln.
- **Modelle und Modellierung** in der Softwareentwicklung dienen dazu, die Analyse und Design-Entwürfe zu kommunizieren, zu diskutieren und abzustimmen.

# Ausblick

---

- In der nächsten Lerneinheit werden wir:
  - in die Disziplin Anforderungsanalyse detaillierter einsteigen.



# Quellenverzeichnis

---

- [1] Larman, C.: UML 2 und Patterns angewendet, mitp Professional, 2005
- [2] Seidel, M. et al.: UML @ Classroom: Eine Einführung in die objektorientierte Modellierung, dpunkt.verlag, 2012
- [3] Martin, R. C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design, mitp Professional, 2018