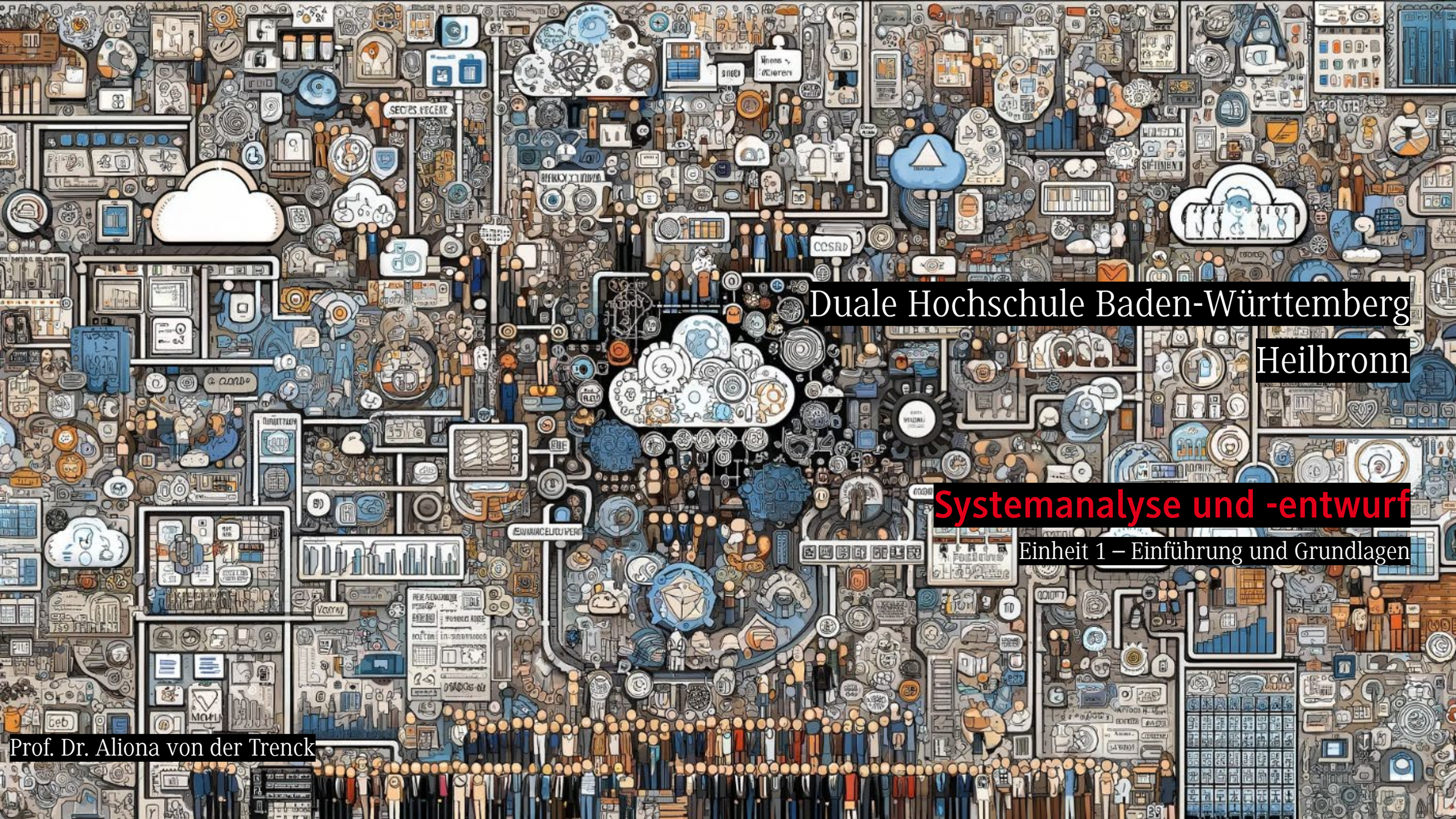


Duale Hochschule Baden-Württemberg

Heilbronn

Systemanalyse und -entwurf

Einheit 1 – Einführung und Grundlagen

Prof. Dr. Aliona von der Trenck

Motivation der Veranstaltung / Lernziele

Modulbeschreibung

Qualifikationsziele und Kompetenzen	
Fachkompetenz	Nach Abschluss des Moduls können die Studierenden wichtige Aufgaben und Systeme der Wirtschaftsinformatik benennen und grundlegend erläutern. Sie können reale Informationssysteme in die Taxonomie der Wirtschaftsinformatik einordnen und als soziotechnische System beschreiben. Die berufliche Rolle als Systemgestalter und Vermittler zwischen betriebswirtschaftlichen Fachanforderungen und Informatik wurde verstanden. Wichtige Aufgaben der Systemanalyse und des Systementwurfs können benannt und erläutert werden.
Methodenkompetenz	Die Studierenden können reale Informationssysteme in die Taxonomie der Wirtschaftsinformatik einordnen und als soziotechnisches System beschreiben. Sie können Systemanforderungen insbesondere auf Basis der Unified Modeling Language (UML) beschreiben und objektorientierte Systeme grundlegend auf Basis der UML entwerfen.
Personale und Soziale Kompetenz	Die Studierenden haben den sozialen Charakter ihrer Vermittlerrolle in der Systemanalyse und im Systementwurf verstanden und können dies in ersten Ansätzen umsetzen. Sie können selbständig nicht zu komplexe Teilaufgaben in der Systemanalyse und im Systementwurf bearbeiten und können die notwendigen Kommunikationstechniken einsetzen, z. B. um Lösungen mit anderen Personen zu diskutieren. Sie sind in der Lage, verschiedene soziale Perspektiven auf Problemstellungen zu erkennen und zu berücksichtigen.
Übergreifende Handlungskompetenz	Die Studierenden können ihre Kenntnisse auf praxisorientierte Fragestellungen im Rahmen von Systemanalyse und -entwurf anwenden, selbständig Problemlösungen erarbeiten und diese im sozialen Prozess erläutern und abstimmen.

Systemanalyse und -entwurf Definition und Ziel von Systemanalyse und Systementwurf – Notwendigkeit eines strukturierten Vorgehens – Grundkonzepte der Objektorientierung (inkl. Darstellung in UML) – Objektorientierte Analyse und objektorientierter Entwurf mit UML (wichtige Modelle und zentrale Konzepte mit Querbezügen zur objektorientierten Programmierung)	**36,0**	**54,0**

Motivation Lernziele

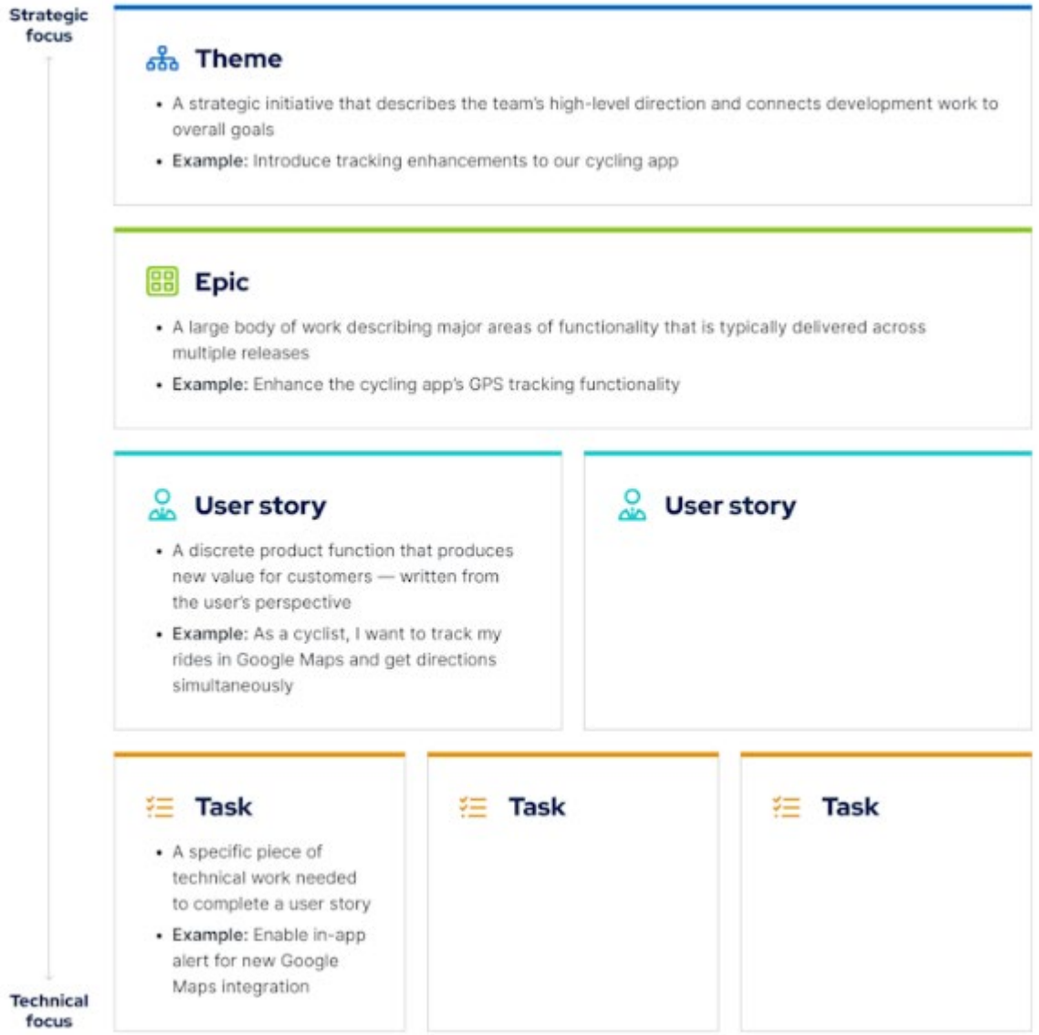
Themes* der Veranstaltung

- Verstehen der Rolle als Gestalter und Vermittler zwischen Fachanforderungen und Informatik
- Benennen und erläutern der wichtige Aufgaben der SEA
- Beschreiben von Systemanforderungen auf UML-Basis und entwerfen der objektorientierten Systeme auf Basis der UML
- Selbständig nicht zu komplexe Teilaufgaben in SEA bearbeiten
- Einsetzen der notwendigen Kommunikationstechniken (Teamarbeit)
- Anwenden von Kenntnissen auf praxisorientierte Fragestellungen im Rahmen von SEA
- Erarbeiten von Problemlösungen selbständig und in Teams (sozialer Prozess)
- Sensibilisieren, was es bedeutet Software „im großen Stil“ zu entwickeln und welche modernen Praktiken dafür notwendig sind
- Verinnerlichen des SCRUMS

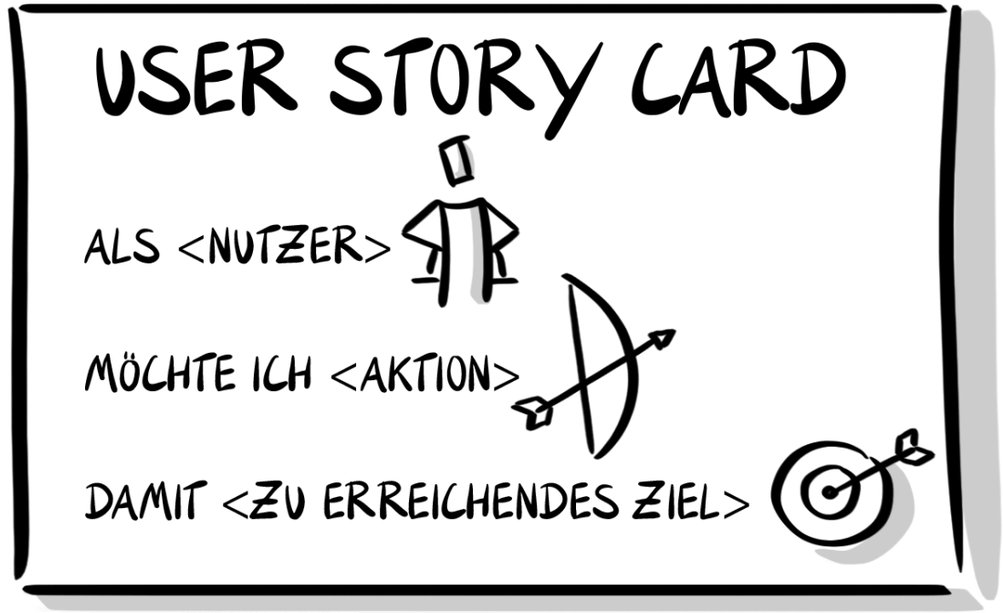
* Theme – die größte Arbeitseinheit in der agilen Entwicklung (alle zugehörigen Epics, Stories und Aufgaben fallen drunter)

** Systemanalyse und des Systementwurfs

SCRUM: Begriffe



Quelle: https://www.aha.io/roadmapping/guide/agile/themes-vs-epics-vs-stories-vs-tasks



Quelle: https://www.agile-academy.com/de/product-owner/meine-lieblingsstruktur-fuer-user-stories/

Literatur

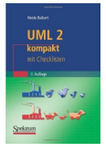
Viele Inhalte und Anwendungsbeispiele der Veranstaltung sind diesen Lehrbüchern entnommen:

- **Applying UML and Patterns** (Craig Larman): Fokus auf Anforderungsanalyse mit gut eingegrenzter UML-Teilmenge

- **Objektorientierte Softwareentwicklung** (Bernd Oestereich): Einführung in objektorientierte Analyse und Design mittels UML

- **UML @ Classroom** (Martina Seidl et al.): UML Sprachkonzepte und Notation der Diagrammarten zur Struktur- und Verhaltensmodellierung auf Basis eines durchgängigen Beispiels.

- **UML 2 glasklar** (Christine Rupp et al.): Umfassender Überblick über alle UML Diagrammtypen. Sehr guter Bezug zur Praxis bzw. dortiger Verwendung von UML.



- **UML 2 kompakt** (Heide Balzert): UML Zusammenfassung, Nachschlagewerk für den (Projekt-)Alltag.

Organisatorisches

Allgemein

- Gestaltung der Pausen
- Vorlesungsform Präsenz und Anwesenheitsliste
- Umgang mit Moodle
  - Bearbeitete (Gruppen-)Aufgaben hochladen und den anderen Teilnehmern zur Verfügung stellen
- Klausur:
  - Dauer: **72 Minuten**
  - Inhalt: Wissens- und Verständnisfragen zu Inhalten der Vorlesung, Aufgaben zu/mit UML-Diagrammen, Modellierung

Organisatorisches

Tool-Einsatz

- Modellierung von UML-Diagrammen ist ein wichtiger Bestandteil der Veranstaltung
- Möglich mit Stift und Papier, wird für die Klausur zum Einsatz kommen
- Tools möglich:
  - **Visual Paradigm** (Online): https://online.visual-paradigm.com/de/
  - **SAP Signavio** für Use Cases und Klassendiagramme:
    - mit der DHBW-E-Mail anmelden
      https://academic.signavio.com/p/register?link=c2a427edd81c4b66abc7ebfd395e898a
  - **Miro**: https://miro.com/app/dashboard/innovation-workspace (nur Visualisierung)

Agenda Systemanalyse und -entwurf

	Thema	Inhalte	(Geplante r) Termin
1	Einführung	Einführung in Veranstaltung. Modulhandbuch, Lernziele und Überblick, empfohlene Literatur. Requirements Game. Überblick Anforderungsmanagement und UML. Aufteilung in Teams und Themen.	31.03 NM
2	Systemanalyse	Das **UML-Anwendungsfalldiagramm** (Use Cases). Sein Zweck und seine Bestandteile, erläutert mit Beispielen. Umsetzung eines Beispiels in Kleingruppen. Das **UML-Aktivitätsdiagramm**. Sein Zweck und seine Bestandteile, erläutert mit Beispielen.	02.04. NM
3	Entwurf	Das **UML-Aktivitätsdiagramm**. Umsetzung eines Beispiels in Kleingruppen. Das **UML-Klassendiagramm**. Sein Zweck und seine Bestandteile, erläutert mit Beispielen. Umsetzung eines größeren Beispiels in Kleingruppen. Verteilung Projektthemen und Gruppeneinteilung.	03.04. VM
4	Action! Anforderungssammlung	**Anforderungssammlung** für Projekte (13:30-15:00) mit Kunden oder Brainstorming. Danach (zusammen): Reflexion, Arbeit in Teams (Tooling und Modellierung)	15.04. NM
5	Guest Lecture	Guest Lecture „Moderne Software-Entwicklung in großen Unternehmen: von der Idee zum Produkt am Beispiel der SAP Internen Development Plattform (IDP) Hyperspace“. Erwartungen an Präsentationen	17.04. VM
6	Präsentationen	**Präsentationen** und Integration der Projekte, Reflexion	14.05 NM
7	Integration	**Erstellen von Beispielsaufgaben** inkl. Musterlösungen in Teams.	21.05 NM
8	Integration	**Messe der Diagramme, Reflexion** **Verankern** des erlernten Vorgehens mit UML an weiteren Beispielen. Entweder einzeln oder in Kleingruppen.	03.06 VM
9	Abschluss	Fragerunde, Wiederholungen, Vorbesprechung **Klausur**	03.06. NM

Organisatorisches

Vorgehensweise der Vorlesung: eduScrum – Elemente vom Scrum in der Lehre

- Die Vorlesung besteht aus 4 Sprints
- In jedem Sprint werden Ziele als User Stories bearbeitet
- User Stories werden von Prof vorgegeben und dienen der inhaltlichen Orientierung
- Am Ende von jedem Sprint erfolgt ein Review in Form von Tests, Aufgaben oder Präsentationen
- Am Ende von jedem Sprint erfolgt eine Retrospektive

- Studierende arbeiten in Teams
- Teams werden heute festgelegt

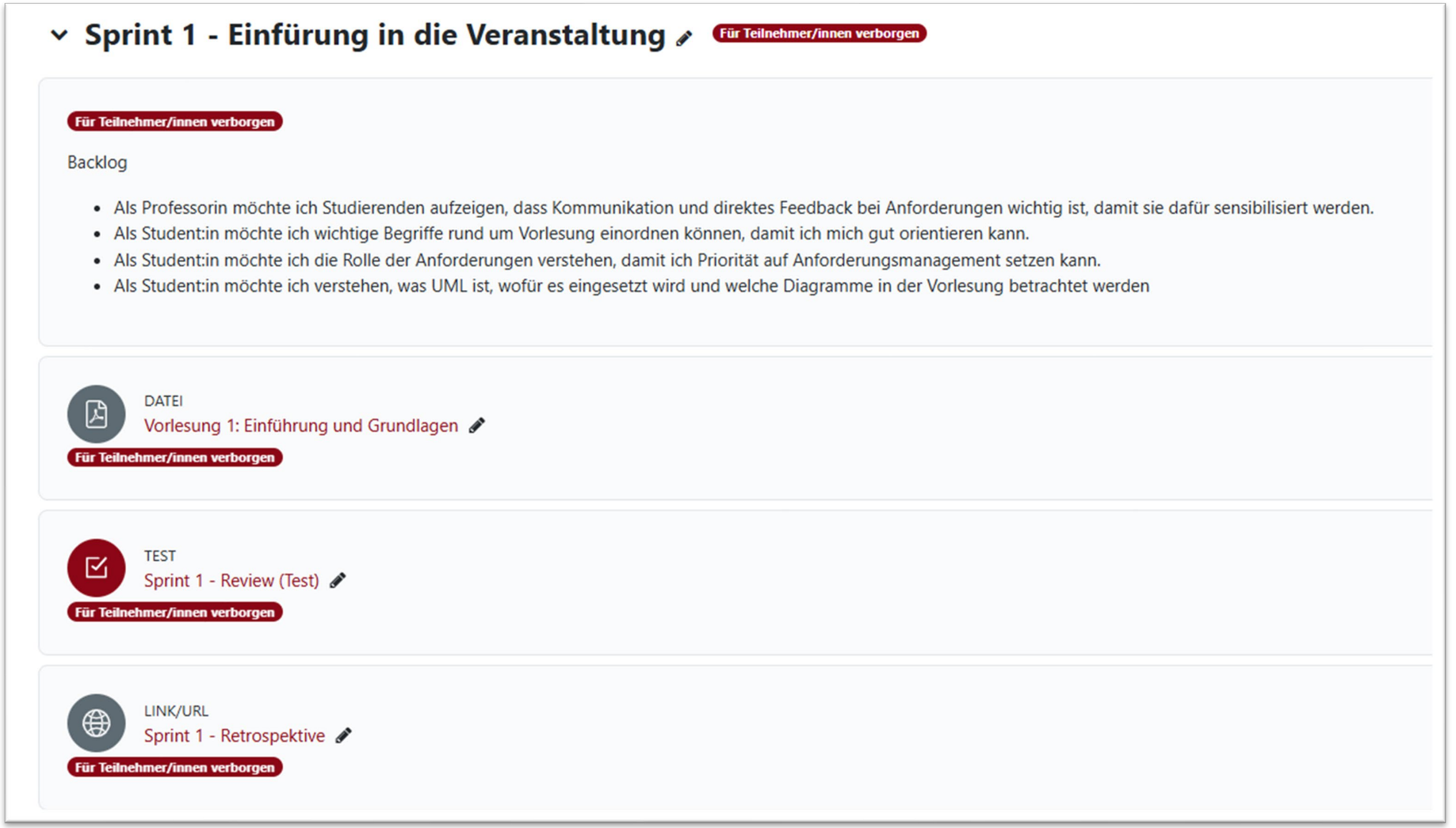
Ansicht der Veranstaltung im Moodle

Willkommen im Sprint 1

Heute

nächste Vorlesung
13:15-13:30

nächste Vorlesung



Ansicht der Veranstaltung im Moodle

Agenda für heute

Inhalte

- Grundlagen und Definitionen
  - System und Software
  - Systemanalyse und -entwurf
  - Softwareentwicklung
- Überblick Anforderungsmanagement und Lebenszyklus der Anforderungen
  - Verständnis für die Bedeutung des Anforderungsmanagements.
  - Überblick über zentrale Begriffe und Methoden.
  - Verbindung zwischen Anforderungen und der späteren Systementwicklung.
- Überblick UML und die wichtigsten Diagramme
  - Was ist UML?
  - Geschichte der UML
  - Die verschiedenen UML-Diagramme

Motivation

Etwa 75 % der Softwareprojekte überschreiten Zeit- und Budgetrahmen und liefern nicht die erwarteten Ergebnisse

- **Fokus auf Technologie und Architektur:** Teams investieren oft viel Zeit in die Auswahl von Technologien und Architekturen, was jedoch selten der Hauptgrund für das Scheitern ist.
- **Vernachlässigung der Fachlichkeit:** Ein häufig übersehener Faktor ist das mangelnde Verständnis der zugrunde liegenden Geschäftsprozesse und Anforderungen.

Empfehlung: Ein tiefes Verständnis der Fachlichkeit und eine enge Zusammenarbeit mit den Fachabteilungen sind entscheidend für den Projekterfolg.

Quelle: https://www.heise.de/blog/75-Prozent-aller-Softwareprojekt-scheitern-was-tun-9979648.html

Definition System und Software

Was ist Software?

- Software („weiche Ware“), Abk. SW, Sammelbezeichnung für Programme, die für den Betrieb von Rechensystemen zur Verfügung stehen, einschl. der zugehörigen Dokumentation (nach **Brockhaus Enzyklopädie**).
- Software: Computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system (nach **IEEE Standard Glossary of Software Engineering Terminology**).

Was ist ein Software-Produkt? (nach Balzert, H.: Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering)

- Ein Produkt ist ein in sich abgeschlossenes, i.a. für einen Auftraggeber bestimmtes Ergebnis eines erfolgreich durchgeführten Projekts oder Herstellungsprozesses. Als Teilprodukt bezeichnen wir einen abgeschlossenen Teil eines Produkts.
- Software-Produkt: Ein Produkt, das aus Software besteht.

Definition System und Software

Was ist ein System? (nach Hesse et al.)

- Unter einem System wird ein Ausschnitt aus der realen oder gedanklichen Welt, bestehend aus Gegenständen (z.B. Menschen, Materialien, Maschinen oder anderen Produkten) und darauf vorhandenen Strukturen (z.B. deren Aufbau aus Teileinheiten oder Beziehungen untereinander) verstanden.

Was ist ein Software-System?

- Ein Software-System ist auf Grundlage der vorherigen Definitionen ein System, dessen Systemkomponenten und Systemelemente aus Software bestehen.

Systemanalyse und -entwurf

Was verstehen wir im Rahmen der Veranstaltung unter der (objektorientierten) Systemanalyse?

- Ein Problem (z.B. Anforderung einer Fachabteilung) wird untersucht
- Die dahinterstehenden Konzepte der Anwendungsdomäne werden erkannt und beschrieben:
  - Abbildung, wie wir als Menschen die Welt sehen
  - Keine Lieferung von Softwareobjekten, sondern von **Konzepten**

Was verstehen wir im Rahmen der Veranstaltung unter dem (objektorientierten) Systementwurf?

- Den Entwurf einer Lösung (z.B. ein Datenbankdesign oder die Klassen einer Software)
- Die Softwareobjekte und ihre Zusammenarbeit zur Lösung der Probleme der Anwendungsdomäne werden definiert
  - **Modellierung von Attributen und Operationen** in Softwareklassen: Diese stellen nicht zwangsweise eine direkte Umsetzung des konzeptionellen Modells dar und müssen sich nicht unmittelbar in Code wandeln lassen
  - **Modellierung des Systemverhaltens:** Z.B. Darstellung des Nachrichtenflusses zwischen Softwareobjekten

Softwareentwicklung

Was unterscheidet Software von anderen Produkten? (nach Balzert, H.)

- **Software ist ein immaterielles Produkt:**
  - Software kann man nicht „anfassen“, nicht „sehen“
- **Software unterliegt keinem Verschleiß, altert aber trotzdem:**
  - Software kann beliebig oft ablaufen, ohne dass Abnutzungserscheinungen auftreten
  - Sie altert aber trotzdem, da die Umgebung, in der eine Software eingesetzt wird, sich ständig ändert
- **Software wird nicht durch physikalische Gesetze begrenzt:**
  - Software ist ein künstliches Produkt des menschlichen Erfindungsgeistes
  - Es basiert nicht auf physikalischen Gesetzen

Softwareentwicklung

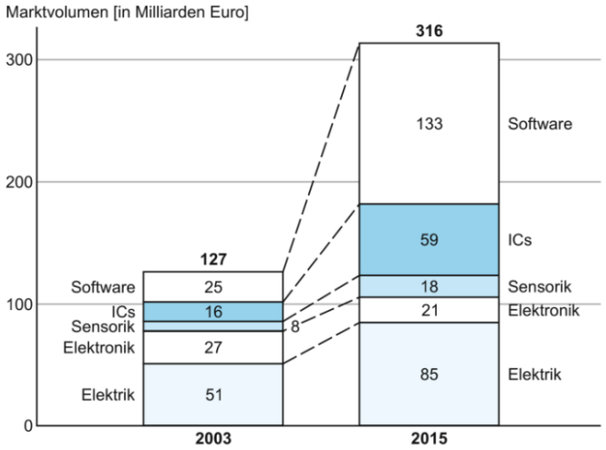
Was unterscheidet Software von anderen Produkten? (nach Balzert, H.)

- **Software ist (im Allgemeinen) leichter und schneller änderbar als ein (physisches) technisches Produkt:**
  - Vorausgesetzt, Software ist gut strukturiert und modularisiert lassen sich Änderungen schnell/einfach durchführen
  - Für Veränderungen technischer Produkte oft neues Werkzeug notwendig
- **Für Software gibt es keine Ersatzteile**
  - Bei technischen Produkten werden defekte Produktteile (oder gleich das ganze Produkt) ausgetauscht und durch in der Regel vorproduzierte Ersatzteile ersetzt. → nur eingeschränkt gültig für Backups
  - Bei Software gibt es keinen Verschleiß. Anpassungen werden erst vorgenommen, wenn Fehler auftreten und nicht „auf Halde“ entwickelt. Nicht zu verwechseln mit allgemeiner Weiterentwicklung/Verbesserung der Software
- **Software ist schwer zu „vermessen“**
  - Technische Produkte kann man i.d.R. sehr exakt vermessen. Sie können sowohl untereinander als auch mit anderen Standards verglichen werden. Das ist bei Software nur bedingt möglich

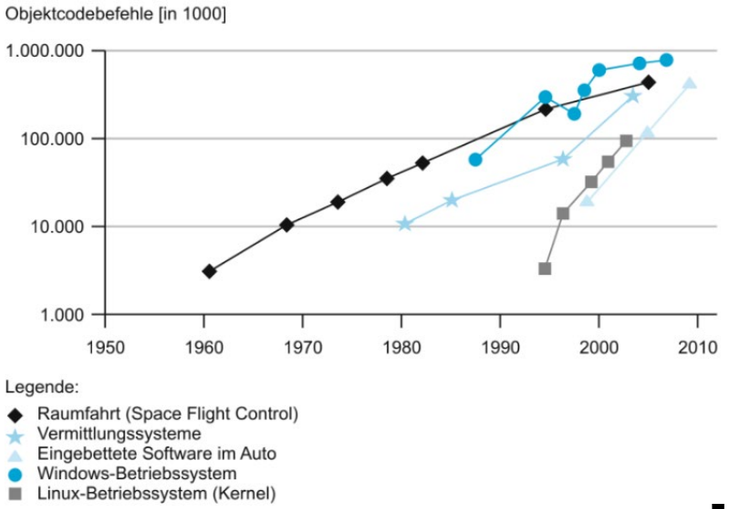
Softwareentwicklung

Welche Veränderungen der letzten Jahre (Jahrzehnte) beeinflussen die Softwareentwicklung? (nach Balzert, H.)

- **Zunehmende Bedeutung von Software**
  - Ohne Software sind heute die wenigsten Produkte denkbar:
    „fundamentaler Werkstoff des Informationszeitalters“
  - Beispiel: Tesla /Automobilbranche allgemein (autonomes Fahren, Smart Car, etc.)



- **Wachsende Komplexität**
  - Softwareeinsatz zum Lösen immer schwierigerer Aufgabenstellungen
  - Ermöglicht durch Fortschritte im Hardwarebereich (z.B. Prozessoren für KI)
  - Beispiel: Tesla / Automobilbranche allgemein (autonomes Fahren)



Softwareentwicklung

Welche Veränderungen der letzten Jahre (Jahrzehnte) beeinflussen die Softwareentwicklung? (nach Balzert, H.)

- **Zunehmend „Altlasten“**
  - Anwendungssoftware wird oft >20 Jahre eingesetzt
  - Einsatzumgebung der Anwendungssoftware ändert sich ständig
    → die Software muss ebenfalls ständig angepasst werden
  - Permanente Anpassungsprozesse verursachen oft 2/3 aller Software-Kosten
  - Beispiel: Übergang von SAP R3 auf S/4 HANA
- **Mehr Standardsoftware und zunehmende „Außer-Haus-Entwicklung“**
  - Individualsoftwareentwicklung teuer und fehleranfällig, Wartung schwieriger / verteilt sich auf wenige Schultern
  - Unternehmen kaufen „passendste“ Standardsoftware und passen sich bzw. ihre Arbeitsweise an diese an (oder häufiger: passen die Standardsoftware zu gewissem Umfang so an, das bestehende Prozesse in der Standardsoftware abgedeckt werden
    → ebenfalls teuer und fehleranfällig)
  - „Früher“ hatten Anwenderunternehmen oft eigene Softwareteams, mittlerweile häufig an Dienstleister ausgelagert

500-Millionen-Euro-Projekt scheitert: Lidl bläst SAP-Software ab (2018)

- Sieben Jahre hat Lidl mit SAP an der Einführung eines neues Warenwirtschaftssystem gearbeitet.
- Jetzt erklärt der Lebensmittel-Discounter das Projekt für gescheitert

Quelle: https://t3n.de/news/500-millionen-euro-projekt-scheitert-lidl-blaest-sap-software-ab-1095673/

Softwareentwicklung

Welche Veränderungen der letzten Jahre (Jahrzehnte) beeinflussen die Softwareentwicklung? (nach Balzert, H.)

- **Zunehmende Qualitätsanforderungen**
  - Software wird immer mehr in Bereichen eingesetzt, wo Fehler nicht nur zu wirtschaftlichen Verlusten führen, sondern auch Menschenleben gefährdet sind.
  - Geschätzt sind für 50 Prozent der Ausfälle im industriellen Sektor Software-Fehler verantwortlich.

  - Verdeutlichung anhand (alter) Beispiele, was das Akzeptieren eines 0,1%-Defektniveaus (kritischer Fehler) bedeutet:
    - pro Jahr: 20.000 fehlerhafte Medikamente, 300 versagende Herzschrittmacher
    - pro Woche: 500 Fehler bei medizinischen Operationen
    - pro Tag: 16.000 verlorene Briefe in der Post, 16 Flugzeugabstürze
    - pro Stunde: 22.000 Schecks nicht korrekt gebucht

Agenda für heute

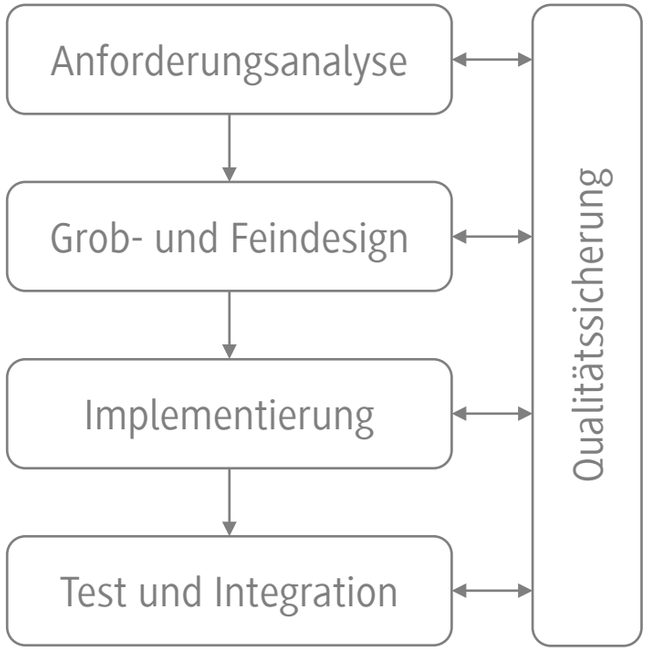
Inhalte

- Grundlagen und Definitionen
  - System und Software
  - Systemanalyse und -entwurf
  - Softwareentwicklung
- Überblick Anforderungsmanagement und Lebenszyklus der Anforderungen
  - Verständnis für die Bedeutung des Anforderungsmanagements.
  - Überblick über zentrale Begriffe und Methoden.
  - Verbindung zwischen Anforderungen und der späteren Systementwicklung.
- Überblick UML und die wichtigsten Diagramme
  - Was ist UML?
  - Geschichte der UML
  - Die verschiedenen UML-Diagramme

Phasen der Softwareentwicklung

Jede Komponente durchläuft grundsätzliche Phasen, unabhängig vom Vorgehensmodell

- Anforderungsanalyse
  - Ziel: verstehen was Kunde will; maßgeblich für Projekterfolg
  - Kunde kann nicht alles was er will zu Papier bringen
  - Intensiver Dialog zwischen Kunde und Entwicklung notwendig
- Designphase
  - Grob: Anforderungen in SW-Modell zu verwandeln, SW und HW Architektur
  - Fein: innere Struktur der Software, Oberflächen-Design, Schnittstellen
- Implementierung
  - Eigentliche Programmierung mit dem Ziel laufende SW oder SW-Inkrement
- Test und Integration
  - Prüfung der SW, Integration mit dem Rest des Systems, Abnahmen
- Qualitätssicherung
  - Ziel: für jedes Teilprodukt im ganzen Entwicklungszyklus sicherzustellen, dass Qualitätskriterien erfüllt sind



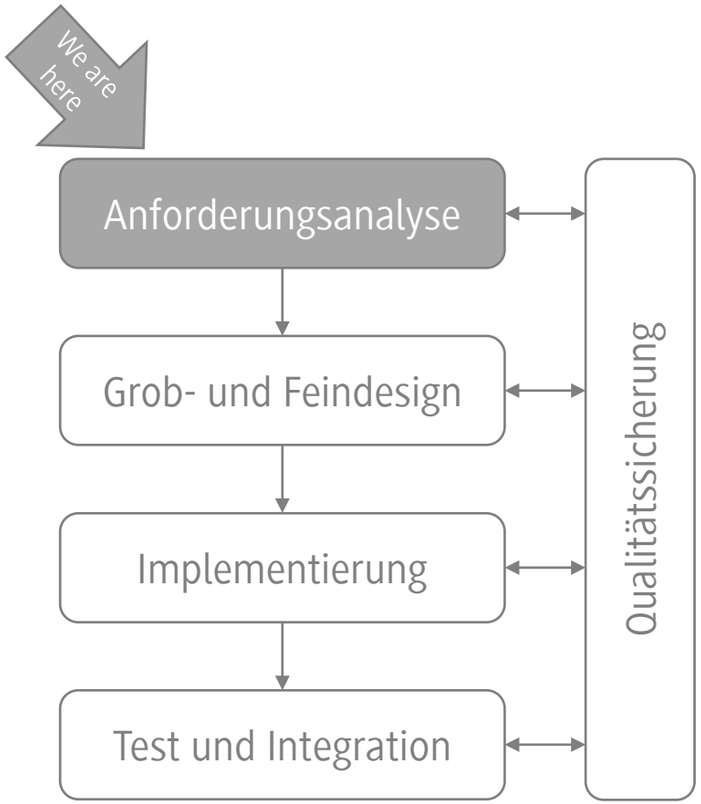
Entwicklungsphasen im Überblick (Grundkurs Software Engineering mit UML, Kleuker)

Phasen der Software Entwicklung

Gut gesammelten Anforderungen bedeuten nicht, dass das Ergebnis gut wird. Schlecht gesammelten führen meistens zu keinem Projekterfolg.

- Die Methoden schlechte Anforderungsanalyse zu verstecken **(so bitte nicht!)**
  - Anforderungen werden zu grob definiert
  - Anforderungen werden zu technisch definiert
- Wer definiert Anforderungen?
  - Kunde
  - Stakeholder*

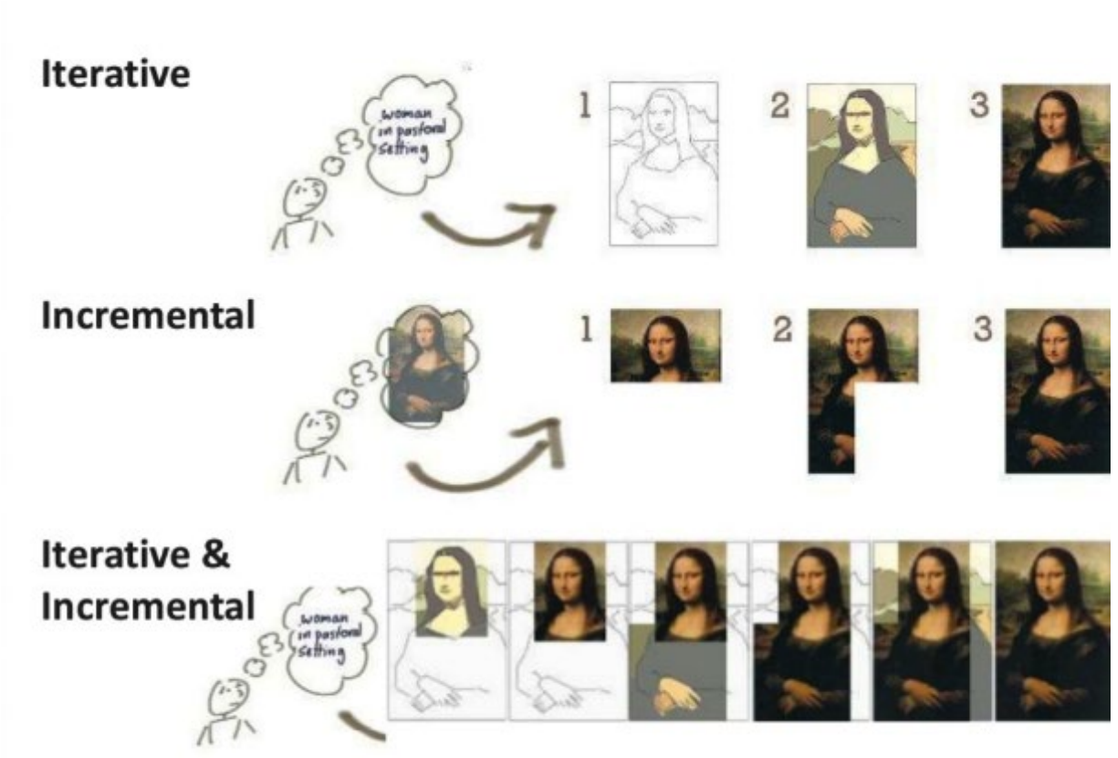
Das Ziel ist es, ganzheitlich Anforderungen zu erfassen und zu dokumentieren



Entwicklungsphasen im Überblick (Grundkurs Software Engineering mit UML, Kleuker)

* Stakeholder - jede Person oder Organisation die von einem Projekt in positiver oder negativer Weise beeinflusst werden kann

Iterativ vs. Inkrementell



Quelle: https://www.everyday.design/guides/whats-the-difference-between-incremental-and-iterative-development

Iterativ vs. Inkrementell

Inkrementelle Entwicklung

Definition:

- In Scrum bezeichnet ein „Inkrement“ die abgeschlossene Arbeit am Ende eines Sprints.
- Es ist ein fertiges, nutzbares Stück Arbeit, das an den Kunden ausgeliefert werden könnte und einen Wert liefert.

Merkmale:

- Jedes Inkrement ist in sich abgeschlossen und funktionsfähig.
- Das gesamte Produkt ist noch nicht fertig, aber dieser Teil kann unabhängig genutzt werden.
- „Potentiell auslieferbar“ bedeutet nicht, dass es sofort veröffentlicht werden muss – nur, dass es möglich wäre.

Beispiele:

- Entwicklung von abgeschlossenen Software-Funktionen pro Sprint.
- Bereitstellung fertiger Design-Assets oder abgeschlossener Marketingkampagnenelemente.

Iterative Entwicklung

Definition:

- Im Gegensatz zur inkrementellen Entwicklung wird bei einer iterativen Vorgehensweise das gesamte Produkt von Anfang an bereitgestellt – jedoch in einer unfertigen Form.
- Es wird in mehreren Durchläufen weiterentwickelt, basierend auf Feedback und Erkenntnissen.

Merkmale:

- Unfertige Versionen eines gesamten Produkts oder einer Kampagne werden bereitgestellt.
- Ziel ist es, Nutzerfeedback einzuholen und das Produkt schrittweise zu verbessern.
- Die Entwicklung erfolgt in mehreren Iterationen, bis das Produkt vollständig ausgereift ist.

Beispiele:

- Ein Low-Fidelity-Prototyp einer App mit allen Funktionen sichtbar, aber nicht funktionsfähig.
- Ein MVP (Minimum Viable Product) einer Marketingkampagne, um Nutzerreaktionen zu testen und darauf basierend Anpassungen vorzunehmen.

Agenda für heute

Inhalte

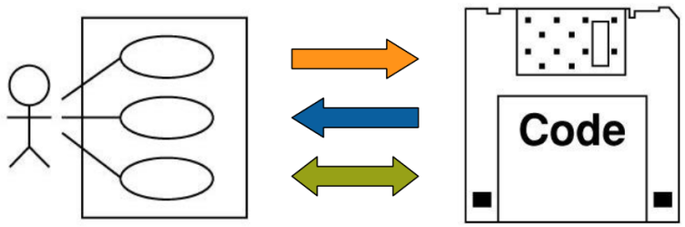
- Grundlagen und Definitionen
  - System und Software
  - Systemanalyse und -entwurf
  - Softwareentwicklung
- Überblick Anforderungsmanagement und Lebenszyklus der Anforderungen
  - Verständnis für die Bedeutung des Anforderungsmanagements.
  - Überblick über zentrale Begriffe und Methoden.
  - Verbindung zwischen Anforderungen und der späteren Systementwicklung.
- Überblick UML und die wichtigsten Diagramme
  - Was ist UML?
  - Geschichte der UML
  - Die verschiedenen UML-Diagramme

Unified Modeling Language

Wie kann UML im Umfeld der Softwareentwicklung helfen und was ist UML überhaupt? (nach Steinpichler und Kargl)

- UML ist eine standardisierte **grafische Darstellungsform** zur Visualisierung, Spezifikation, Konstruktion und Dokumentation von (Software-)Systemen.

- UML bietet ein Set an **standardisierten Diagrammtypen**, mit denen **komplexe Abläufe** und Systeme übersichtlich und **verständlich dargestellt** werden können.
  - UML ist **keine Vorgehensweise und auch kein Prozess**
  - UML ist ein „Wörterbuch“ an Symbolen mit definierter Bedeutung
  - UML bietet Diagrammtypen für die objektorientierte Analyse, Design und Programmierung
  - UML kann nahtlosen Übergang von den Anforderungen an ein System bis zur fertigen Implementierung gewährleisten (oder aus Code abgeleitet werden).



Steinpichler und Kargl: Projektabwicklung mit UML und Enterprise Architect

Unified Modeling Language

Wie kann UML im Umfeld der Softwareentwicklung helfen und was ist UML überhaupt? (nach Steinpichler und Kargl)

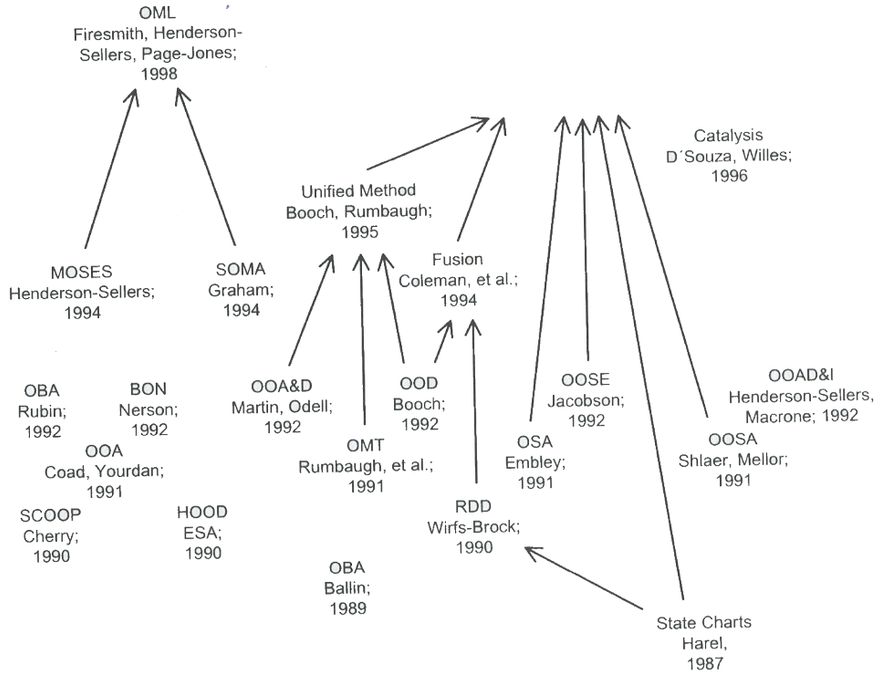
- Wichtiger Einsatzzweck der UML ist es, Diagramme für die Verwendung in der Projektdokumentation zu erstellen, z.B.
  - Use Case Diagramme zur Beschreibung der funktionalen Anforderungen in der Anforderungsdefinition
  - Klassendiagramme zur Beschreibung der Softwarearchitektur im Designdokument
- Ziel von UML als „**gemeinsame Sprache**“: Verbesserung der Zusammenarbeit zwischen Technikern und Nicht-Technikern.
- UML hilft allen Beteiligten dadurch:
  - Systeme besser zu verstehen
  - Möglichkeiten der Vereinfachung und/oder Wiederverwendbarkeit aufzudecken
  - mögliche Risiken besser zu erkennen
  - Fehler frühzeitig (in der Analyse- bzw. Designphase) zu erkennen
  - Kosten während der Projektlaufzeit im Allg. und der Implementierungsphase im Bes. geringer/planbar zu halten.
- UML ist prinzipiell auch für Organisationsprojekte einsetzbar. Prozesse können visualisiert und im Anschluss analysiert und verbessert werden

Unified Modeling Language

Geschichte der UML (nach Seidl et al.: UML@Classroom)

- In den 1980ern/90ern: Entwicklung einer Vielzahl von (objektorientierten) Modellierungssprachen
- Problem: Jeder verwendete eine andere Sprache, Modelle von Projektpartnern **nicht zueinander kompatibel**, Wiederverwendbarkeit in neuen Projekten nicht gegeben
- Enormer Aufwand über unterschiedliche Notationen zu diskutieren – statt über das eigentlich zu modellierende und zu lösende Problem
- Unklar, in welche Sprache investiert werden sollte, da ständig neue auftauchten und bestehende verschwanden

- Diese Zeit wird rückblickend „**Methodenkrieg**“ genannt



Auf der Suche nach der „Lösung“ – der Methodenkrieg (UML 2 glasklar, Christine Rupp et al.)

Unified Modeling Language

Geschichte der UML (nach Seidl et al.: UML@Classroom)

- Mitte der 1990er die Einsicht: So kann es nicht weitergehen

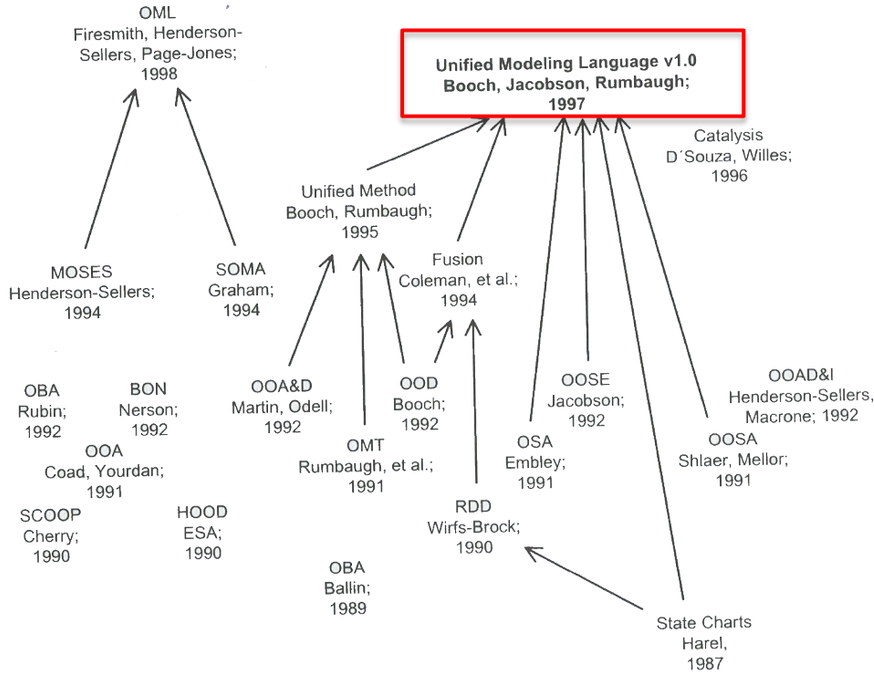
- 1996 ruft die **Object Management Group** (OMG, wichtigstes Standardisierungsgremium für objektorientierte Softwareentwicklung) zur Spezifikation eines **einheitlichen Modellierungsstandards** auf

- Parallel dazu arbeiten Booch, Jacobson, Rumbaugh (die „drei Amigos“) an Vereinheitlichung ihrer Ansätze
- Sie stecken sich dabei u.a. folgende Ziele:
  - Verwendung von **Objektorientierung zur Repräsentation kompletter Systeme**, nicht nur Teilen von Software
  - Explizite Bindung zwischen Modellierungskonzepten und ausführbarem Programmcode
  - Die Modellierungssprache soll sowohl **durch Maschinen verarbeitbar** als auch **für Menschen lesbar** sein

Unified Modeling Language

Geschichte der UML (nach Seidl et al.: UML@Classroom)

- Resultat ihrer Arbeit: **die Unified Modeling Language**
- Einreichung in Version 1.0 auf OMG-Aufruf
- Ab Version 1.1: Beteiligung weiterer Autoren, erschienen 1998
- In den Folgejahren: Stetige Weiterentwicklung, Integration weiterer Funktionen und Technologien (z.B. XML Metadata Interchange Format)
- 1999: OMG übernimmt Copyright an UML
- 2005: Veröffentlichung UML 2. Ziel: Stabilisierung und Komplexitätsreduzierung der Modellierung
- **Aktuelle Version: 2.5.1** (erschienen 2017)



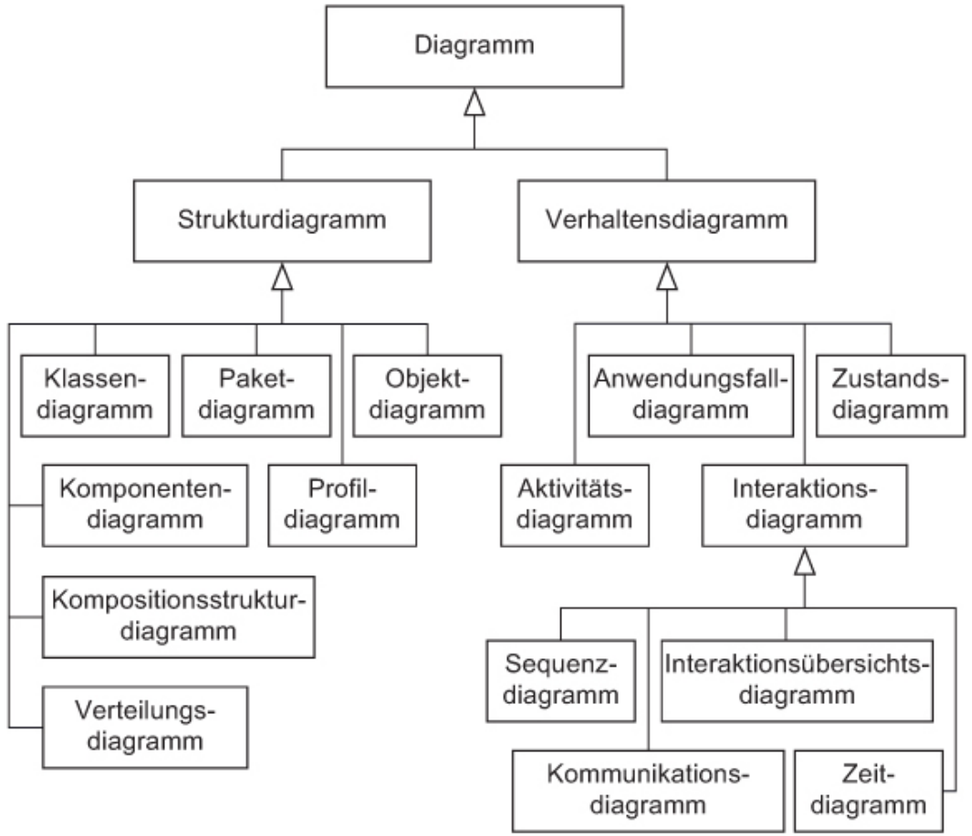
Auf der Suche nach der „Lösung“ – der Methodenkrieg (UML 2 glasklar, Christine Rupp et al.)

- UML ist heute **eine der am weitesten verbreiteten**, grafischen, objektorientierten **Modellierungssprachen**. Ihre Entwicklung „stagniert“ jedoch – oder ist sie einfach „ausgereift“?

Unified Modeling Language

UML-Diagramme (nach Seidl et al.: UML@Classroom)

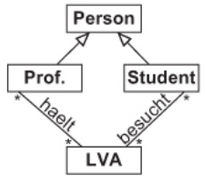
- UML-Diagramme unterteilen sich grundsätzlich in
  - **Strukturdiagramme:** Können zur Modellierung der **Statik** des Systems verwendet werden. Dynamisches Verhalten (Änderungen über die Zeit) wird in diesen Diagrammen nicht berücksichtigt.
  - **Verhaltensdiagramme:** Definieren das Verhalten von Objekten, welche Folgen Aktionen von Objekten haben. Die Diagramme beschreiben also, wie sich die **Zustände** von Objekten über die Zeit hinweg **verändern**.



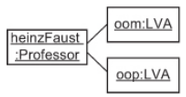
UML-Diagramme (Seidl et al.: UML@Classroom)

Unified Modeling Language

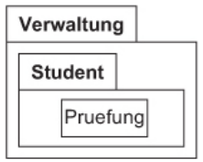
UML-Diagramme (nach Seidl et al. und Rupp et al.)



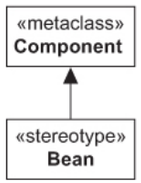
- **Klassendiagramm**: Aus welchen Klassen besteht mein System und wie stehen diese untereinander in Beziehung?



- **Objektdiagramm**: Welche innere Struktur besitzt mein System zu einem bestimmten Zeitpunkt zur Laufzeit? (Klassendiagramm-„Schnappschuss“)



- **Paketdiagramm**: Wie kann ich mein Modell so organisieren, dass ich den Überblick bewahre?

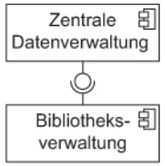


- **Profildiagramm**: Muss ich weitere anwendungsspezifische Konzepte in meine Modellierung einführen?

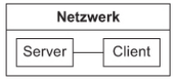
Strukturdiagramme

Unified Modeling Language

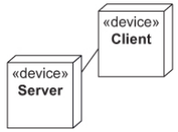
UML-Diagramme (nach Seidl et al. und Rupp et al.)



- **Komponentendiagramm**: Wie werden meine Klassen zu wiederverwendbaren, verwaltbaren Komponenten zusammengefasst? Wie stehen diese miteinander in Beziehung?



- **Kompositionsstrukturdiagramm**: Wie sieht das Innenleben einer Klasse, einer Komponente, eines Systemteils aus?

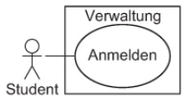


- **Verteilungsdiagramm**: Wie sieht das Einsatzumfeld (Hardware, Server, Datenbanken, …) des Systems aus? Wie werden die Komponenten zur Laufzeit wohin verteilt?

Strukturdiagramme

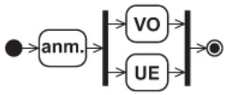
Unified Modeling Language

UML-Diagramme (nach Seidl et al. und Rupp et al.)

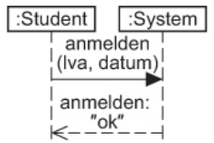


- **Anwendungsfalldiagramm**: Was leistet mein System für seine Umwelt (Nachbarsysteme, Stakeholder)?

- **Zustandsdiagramm**: Welche Zustände kann ein Objekt, eine Schnittstelle, ein Use Case, etc. bei welchen Ereignissen annehmen?



- **Aktivitätsdiagramm**: Wie läuft ein bestimmter flussorientierter Prozess oder ein Algorithmus ab?

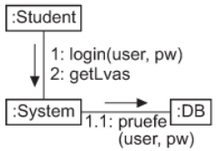


- **Sequenzdiagramm**: Wer tauscht mit wem welche Informationen in welcher Reihenfolge aus?

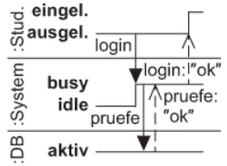
Verhaltensdiagramme

Unified Modeling Language

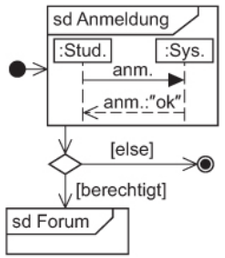
UML-Diagramme (nach Seidl et al. und Rupp et al.)



- **Kommunikationsdiagramm**: Wer kommuniziert mit wem? Wer „arbeitet“ im System zusammen?



- **Zeitdiagramm**: Wann befinden sich verschiedene Interaktionspartner in welchem Zustand?



- **Interaktionsübersichtsdiagramm**: Wann läuft welche Interaktion ab?

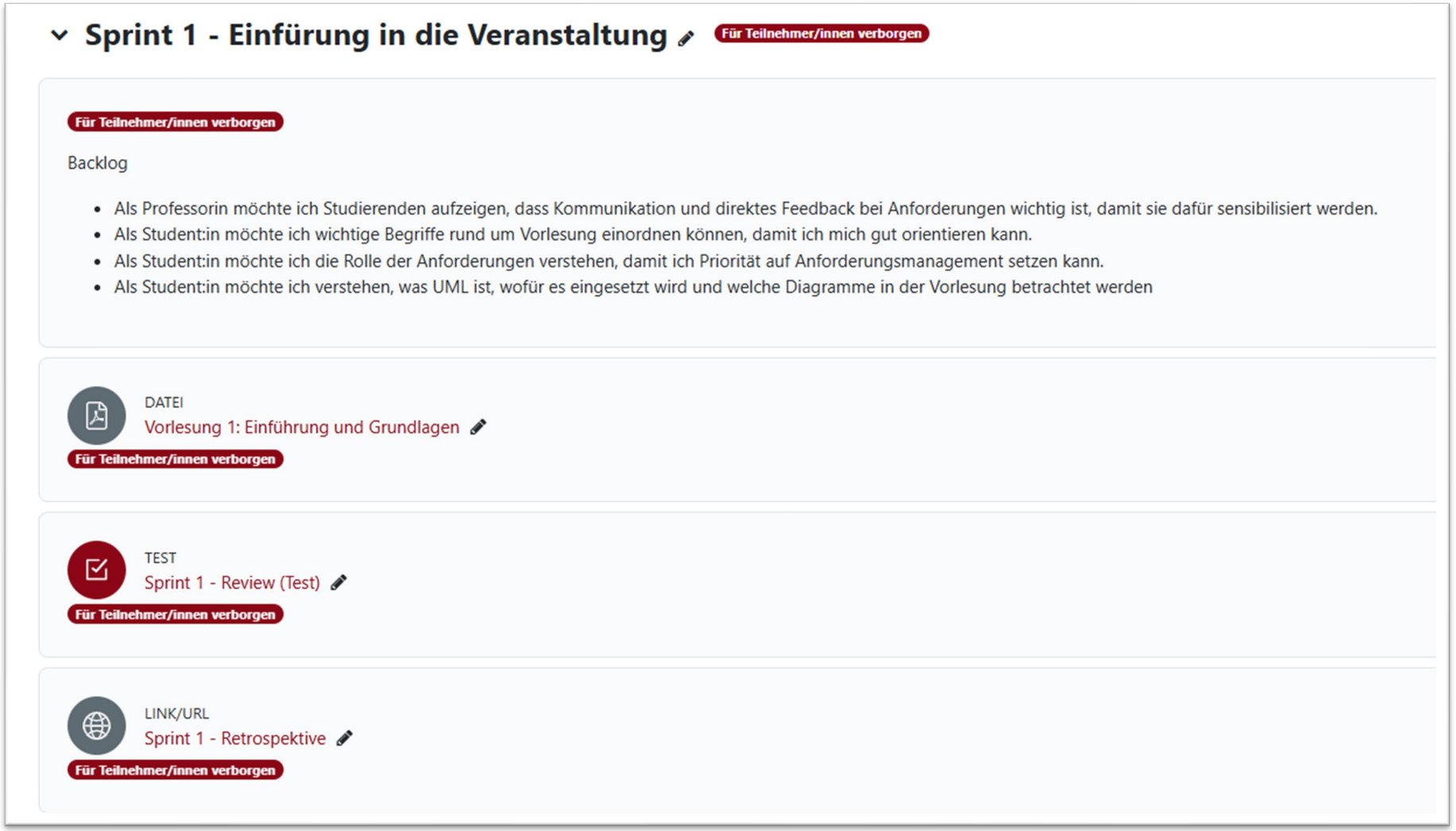
Verhaltensdiagramme

Willkommen im Sprint 1

Heute

nächste Vorlesung -13:35

nächste Vorlesung



Ansicht der Veranstaltung im Moodle