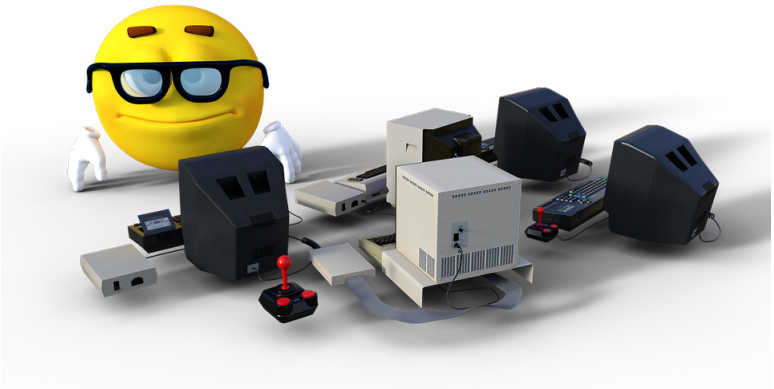
Westfälische Hochschule

XXX

Alles wichtige auf einen Blick

XXX Semesterzusammenfassung



Author(s):
-

Supervisor(s):
Prof. -

4. November 2025

Vorwort

Dieses Dokument ist zu der Vorlesung aus dem Jahr 2025 entstanden. Inhate können im vergleich zu anderen Jahren abweichen. Es wird keine Garantie auf richtigkeit der Inhalte gegeben. Dieses Dokument ist eine ergänzung für die Vorlesungen und kein vollständiger ersatz.

Inhaltsverzeichnis

Teil I

Semesterzusammenfassung

1. Einführung

1.1 Projektbasierte vs. Produktbasierte Software

1.1.1 Projektbasierte Software

Bei der Projektbasierten Software entwickelt der *Auftragnehmer* für den *Auftraggeber*. Der Auftraggeber hat die Geschäftsidee oder das Problem und erstellt die Vision der Software. Der Auftragnehmer verdient an der Software, nicht an der Vision.

Der Auftraggeber kommuniziert also seine Vision an den Auftragnehmer, welcher dann sicherstellt, dass die Software der Vision des Auftragsgebers entspricht.

1.1.2 Produktbasierte Software

Bei der Produktbasierten Software gibt es keine klare Trennung zwischen Auftraggeber und Auftragnehmer. Der Entwickler verdient an der Geschäftsidee. Es gibt hier keine formale Abnahme, sondern nur Akzeptanz am Markt. Die Software ist also nie richtig fertig.

2. Vorgehensmodelle

2.1 Phasen eines Softwaresystems

Softwaresysteme durchlaufen im Wesentlichen einen Lebenszyklus, der sich in Phasen aufteilen lässt. Eine gängige Gliederung umfasst:

1. **Idee** (Was ist das Problem/die Vision?)
2. **Konzept** (Was soll die Software tun? Anforderungen.)
3. **Lösung** (Wie soll es umgesetzt werden? Architektur, Design.)
4. **Realisierung** (Die eigentliche Implementierung/Codierung.)
5. **Test** (Verifikation und Validierung.)
6. **Betrieb** (Auslieferung, Wartung und Weiterentwicklung.)

Diese Phasen müssen durchlaufen werden. Da es hierfür nicht *die* eine Vorgehensweise gibt, wurden verschiedene Vorgehensmodelle entwickelt. Je nach Projekt und dessen Anforderungen eignet sich ein anderes Modell.

2.2 Arten von Vorgehensmodellen (Überblick)

Vorgehensmodelle sind wie *Schablonen*, die man verwenden und anpassen kann, um einen Software-Lebenszyklus zu erstellen. Sie geben beispielsweise an, welche Phasen wie, wann und wie oft durchlaufen werden. Im Englischen: **Software Life Cycle Model (SLCM)**.

Die wichtigsten Arten von Vorgehensmodellen sind:

- **Planbasiert (z.B. Wasserfall)**
  - Phasen werden sequentiell aufgeführt.
  - Detaillierte Planung am Anfang.
  - Vollständiges Produkt erst am Ende.
  - Kein Wert für den Kunden vor Projektende.
- **Inkrementell**
  - System wird in Teilen gebaut. Noch nicht fertige Teile bekommen Dummy-Funktionalität.

  - Jede Iteration ergibt ein lauffähiges Produkt mit Teilfunktionalität.
  - Fertiggestellte Systeme werden in späteren Iterationen typischerweise nicht mehr verändert.

- **Evolutionär / Agil (z.B. Scrum)**
  - Initiale Grob-Planung zu Beginn.
  - In jeder Iteration: Anforderungsanalyse, Design, Implementierung, Testen, Delivery.
  - Änderungen sind erwünscht und willkommen.
  - Fertiggestellte Teile des Systems können, wenn nötig, jederzeit verändert werden.
  - Jede Iteration soll mehr Wert für den Kunden bringen.
  - Kunde ist früh in die Entwicklung eingebunden und kann diese besser beeinflussen.

2.3 Wasserfall und V-Modell

2.3.1 Wasserfall Modell

Das Wasserfallmodell ist das klassische, lineare und **streng sequentielle** Vorgehensmodell im Software-Engineering. Der Name leitet sich davon ab, dass der Prozess wie Wasser in einem Wasserfall Stufe für Stufe "nach unten"fließt. [Image of Waterfall software model diagram]

Das Kernprinzip ist, dass jede Phase (z.B. Idee, Konzept, Lösung, ...) genau einmal durchlaufen wird. Eine neue Phase beginnt erst dann, wenn die vorherige Phase **vollständig abgeschlossen** und formal abgenommen (z.B. durch ein SSign-off") wurde.

Es gibt in der reinen Lehre dieses Modells **keine Rücksprünge** (Iterationen oder Loops) zu früheren Phasen. Einmal getroffene Entscheidungen (z.B. im Konzept) werden in späteren Phasen (z.B. Realisierung) nicht mehr grundlegend geändert.

2.3.2 V-Modell

Das V-Modell ist eine Erweiterung des Wasserfallmodells. Es ergänzt das lineare Phasenmodell um den Aspekt der **Qualitätssicherung** parallel zu jeder Entwurfsphase.

Die charakteristische V-Form entsteht, indem den Phasen des **Entwurfs** (linker Ast des "V") spezifische Phasen der **Verifikation und Validierung** (rechter Ast des "V") gegenübergestellt werden.

Der linke Ast beschreibt die Detaillierung des Systems (vom Groben zum Feinen):

- **Anforderungsdefinition:** Was soll das System tun?

- **Systementwurf:** Wie ist die Gesamtarchitektur (Grobentwurf)?
- **Komponentenentwurf:** Wie sind die einzelnen Module aufgebaut (Feinentwurf)?
- **Implementierung:** Die Codierung der Module am tiefsten Punkt des "V".

Der rechte Ast beschreibt die Integration und Prüfung (vom Feinen zum Groben):

- **Modultest (Komponententest):** Testet einzelne Module gegen den Komponentenentwurf.
- **Integrationstest:** Testet das Zusammenspiel der Module gegen den Systementwurf.
- **Systemtest:** Testet das Gesamtsystem gegen die Anforderungsdefinition.
- **Abnahmetest:** Der Auftraggeber validiert das System.

Das Kernprinzip ist, dass jede Entwurfsphase auf der linken Seite die Grundlage für die Testspezifikation der gegenüberliegenden Testphase auf der rechten Seite liefert. So werden beispielsweise die Abnahmetestfälle bereits während der Anforderungsdefinition geplant.

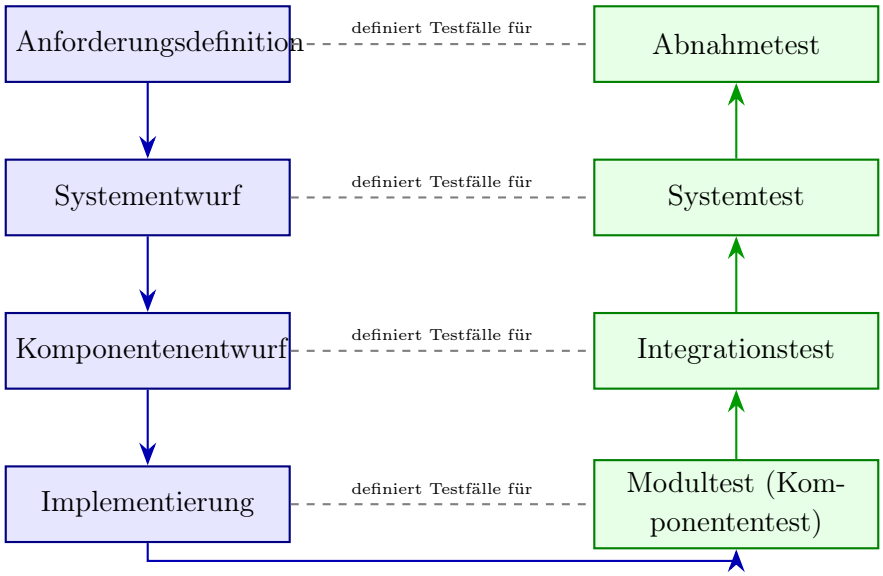


Abbildung 2.1: Schematische Darstellung des V-Modells

2.3.3 Annahmen von Wasserfall und V-Modell

Diese Modelle basieren auf einigen Annahmen, die in der Praxis oft problematisch sind:

- **Perfekte Vorkenntnisse:** Der Auftraggeber und Auftragnehmer müssen ganz genau wissen, was sie wollen und was zu tun ist. Oft ist am Anfang eines Projektes aber nicht genau klar, was gebraucht wird. Der Auftraggeber kann sich nicht genau vorstellen, wie sein System aussehen soll und vermeidet deswegen, Entscheidungen zu treffen.

- **Vollständige Analyse:** Alle Probleme müssen am Anfang des Projektes vollständig analysiert werden. Dies ist nur selten möglich und noch seltener erwünscht, da dies sehr viel kostet. Oft ist nicht genug Zeit, um am Anfang Wochen oder Monate in die Analyse zu stecken.
- **Geringe Flexibilität:** Die Software ist sehr unflexibel und es werden nur sehr wenig Änderungen erwartet. Der Auftraggeber ist oft nur über Dokumentation in das Projekt eingebunden. Die Kommunikation ist darüber sehr schwierig und es kommt oft zu Missverständnissen. Der Auftraggeber realisiert oft erst in der Testphase, was noch fehlt und was er braucht.

2.3.4 Schwächen von Wasserfall und V-Modell

- Das Wasserfallmodell macht die Annahme, dass jede Phase vollständig und fehlerfrei beendet wird und auf die nachfolgende Phase aufbaut.
- Ist dies nicht der Fall (was es oft ist), ist Nacharbeit einer bereits beendeten Phase nötig.
- Diese Nacharbeit ist kein Teil vom Wasserfall und damit nicht Teil der Projektplanung. Hierdurch entstehen also zusätzliche Kosten und Zeitbedarf.
- Beim V-Modell ist dies ähnlich: Obwohl Tests früher geplant werden, erfolgt die *Durchführung* der wichtigen Tests (System-, Abnahmetest) erst sehr spät im Prozess.

2.3.5 Stärken von Wasserfall und V-Modell

Der Wasserfall (und das V-Modell) eignet sich gut für Projekte mit **stabilen Anforderungen**, in denen der Auftraggeber ganz genau weiß, was er will. Beispielsweise für die Anpassung der Steuersoftware des Finanzamts für ein neues Steuerjahr.

Außerdem eignet er sich gut für **Kritische Systeme**:

- **Safety-critical:** Softwarefehler können Menschen verletzen oder töten, z.B. in der Luftfahrt und Medizin.
- **Mission-critical:** Softwarefehler können das gesamte System zerstören, z.B. in der Raumfahrt.
- **Business-critical:** Softwarefehler können in hohen Kosten resultieren, z.B. Trading-Software.

Zudem eignet sich der Wasserfall gut für sehr große Systeme, da er eine klare Struktur und definierte Meilensteine (Phasenabschlüsse) vorgibt.

2.4 Agilität

2.4.1 Das Agile Manifest: 12 Prinzipien

1. Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.
2. Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
3. Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
4. Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.
5. Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.
6. Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.
7. Funktionierende Software ist das wichtigste Fortschrittsmaß.
8. Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
9. Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.
10. Einfachheit — die Kunst, die Menge nicht getaner Arbeit zu maximieren — ist essenziell.
11. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
12. In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

2.4.2 Scrum

Scrum ist eine agile Methode. Scrum ist, per Definition, *kein* starres Vorgehensmodell, sondern ein **Framework** (Rahmenwerk). Es wird in der Praxis aber oft genutzt, um den Software-Lebenszyklus zu strukturieren.

Bei Scrum wird jeder in Rollen eingeteilt:

- **Scrum-Master**: Verantwortlich dafür, dass Scrum korrekt ausgeführt wird und dass Hindernisse (Impediments) für das Team beseitigt werden.
- **Product Owner**: Verantwortlich für die Maximierung des Wertes des zu entwickelnden Produkts; managt das Product Backlog.

- **Developers**: Entwickeln als selbstorganisiertes Team die Software.
- **Scrum-Team**: Scrum Master + Product Owner + Developers (das gesamte Team).

Scrum besteht aus mehreren Kern-Elementen (Events und Artefakten):

Product Goal

Das Product Goal beschreibt das langfristige Ziel, das das Produkt letztlich erreichen soll.

Product Backlog

Eine priorisierte To-do-Liste von Bugs, Features und Verbesserungen für das gesamte Produkt.

Product Backlog Beispiel

Priorität	**Typ**	**Product Backlog Item (User Story / Task)**
1	Feature	Als Lehrer möchte ich definieren können, welche Lern-Werkzeuge für welche Klassen verfügbar sind.
2	Feature	Als Elternteil möchte ich die Hausaufgaben meiner Kinder einsehen, die ihnen von ihrem Lehrer aufgegeben wurden.
3	Feature	Als ein Lehrer von jungen Kindern möchte ich ein User-Interface mit Bildern haben für Kinder, die noch nicht gut lesen können.
4	Entwicklungs-Aktivität	Kriterien ausarbeiten für die Auswahl und Bewertung von Open Source Software, die für Teile des Systems genutzt werden könnte.
5	Verbesserung	Refactoring des User Interface Codes für bessere Lesbarkeit und Performance.
6	Verbesserung	Implementierung von Verschlüsselung für persönliche Daten von Usern.

Tabelle 2.1: Beispiel-Ausschnitt eines Product Backlogs

Sprint

Scrum ist in Sprints aufgeteilt. Ein Sprint ist ein fester Zeitrahmen (Timebox) von in der Regel 2-4 Wochen. Er stellt den Rahmen für alle Aktivitäten in Scrum.

Increment

Am Ende jedes Sprints steht ein lauffähiges Ïncrement"(Produkt-Zuwachs), das potenziell auslieferbar ist und mehr Wert hat als vor dem Sprint.

Sprint Planning

Das Sprint Planning ist ein Scrum-Event zu Beginn des Sprints (ca. 4-8 Stunden). Es werden 3 Themen behandelt:

1. **Warum?** Das Ziel des Sprints (Sprint Goal) wird definiert.
2. **Was?** Auswahl aus dem Product Backlog: Was kann in diesem Sprint erreicht werden?
3. **Wie?** Definition von Tasks: Wie kann die Arbeit durch die Developers erreicht werden?

Sprint Backlog

Enthält die Ergebnisse des Sprint Planning. Dazu zählen:

- Das Ziel des Sprints (Sprint Goal)
- Die ausgewählten Product Backlog Items
- Die detaillierten Entwicklungs-Tasks

Daily Scrum

Tägliches Meeting (Timebox: max. 15 Min.) für die Developers. Typische Themen zur Synchronisierung:

- Was habe ich gestern getan, um das Sprint Goal zu erreichen?
- Was werde ich heute tun?
- Welche Hindernisse (Impediments) sehe ich?

Definition of Done (DoD)

Definiert, wann ein Product Backlog Item wirklich "fertigïst (Qualitätsstandard). z.B.:

- Code gemerged
- Alle Tests sind grün (Unit-, Integrationstests)
- Code-Review erfolgt
- 80% Code Coverage erreicht
- Dokumentation aktualisiert

Sprint Review

Ein Scrum-Event am Ende des Sprints (ca. 2-4 Stunden). Das Scrum-Team stellt das Increment den Stakeholdern (z.B. Kunde, Management) vor. Gemeinsam wird überlegt, ob man auf dem richtigen Weg ist und was als Nächstes zu tun ist (Anpassung Product Backlog).

Sprint Retrospective

Ein Scrum-Event ganz am Ende des Sprints (ca. 1-3 Stunden). Nur das Scrum-Team diskutiert, was im letzten Sprint gut und was schlecht gelaufen ist (Prozess, Zusammenarbeit). Das Team entscheidet selbst, was es ändern muss, um im nächsten Sprint effektiver zu werden.

Backlog Refinement (oder Grooming)

Findet durchgehend während des Sprints statt (oft als separates Meeting). Product Backlog Items werden vom Team (oft mit Product Owner) priorisiert, detailliert, in kleinere Items zerlegt und auf ihren Aufwand hin abgeschätzt.

2.4.3 Eigenschaften agiler Teammitglieder

- **Selbstorganisation:** Fähigkeit, Aufgaben eigenständig zu priorisieren und umzusetzen.
- **Kollaborationsfähigkeit:** Offene Kommunikation, aktives Zuhören, Wissensaustausch.
- **Verantwortungsbewusstsein:** Übernahme von Verantwortung für Ergebnisse (Outcome) und nicht nur für die Erledigung von Aufgaben (Output).
- **Lernbereitschaft:** Offenheit für Feedback, kontinuierliches Lernen.
- **Transparenz:** Arbeitsergebnisse und Fortschritte sichtbar machen.
- **Technische Exzellenz:** Fundierte Fähigkeiten, oft in mehreren Fachgebieten (T-Shaped / Full-Stack).

2.4.4 Vergleich Produkt- vs. Projektbasierte Software

- **Produkt-Software** (z.B. eine App, die stetig weiterentwickelt wird)
  - Verwendet so gut wie immer agiles Vorgehen.
  - Ermöglicht frühen und schnellen Gang an den Markt.
  - Erleichtert schnelles Reagieren auf Kundenwünsche oder Marktänderungen.
- **Projekt-Software** (z.B. eine Individualsoftware für einen festen Kunden)
  - Nutzt Agil nur, wenn eine gute Vertrauensbasis mit dem Kunden vorhanden ist.
  - Oft eher inkrementell oder (bei stark vertrags-lastiger Kundenbeziehung) sogar Wasserfall-artig.

Teil II

Übungsaufgaben

3. Übungsblatt 1

3.1 Aufgabe 1

3.1.1 Wiederholungsfragen aus ERP und ORP

Was ist der Unterschied zwischen einer Klasse und einem Objekt?

Ein Objekt ist eine Instanz einer Klasse. Die Klasse gibt den Bauplan vor und das Objekt existiert nach diesen.

Was ist der Unterschied zwischen public, private und protected?

public ist für jede andere Klasse sichtbar, private nur innerhalb der Klasse und protected innerhalb der Klasse und allen Kindern der klasse.

Was ist der Unterschied zwischen einem Interface und einer abstrakten Klasse?

Eine abstrakte klasse gibt vor wie sich eine Kindklasse verhalten muss und ein Interface gibt ein versprechen, wie sich eine Klasse verhält.

Was ist der Unterschied zwischen statischen und nicht-statischen Methoden?

Eine Statische methode kann nicht überschrieben werden. Außerdem ist sie in jeder Instanz der klasse identisch.

Was ist Kapselung? Finden Sie ein Beispiel im Code.

...

Was ist Polymorphie? Finden Sie ein Beispiel im Code.

...

Was ist Vererbung? Finden Sie ein Beispiel im Code.

...

Finden Sie den Bug bei der Verwendung von Heiltränken?

Wenn Energy hinzugefügt wird wodurch die Energy größer ist als die max-Health wird der Aufruf einfach ins void geschickt und es passiert nix.

3.1.2 Implementieren Sie (schriftlich, keine echte Implementierung):

Was ist Kapselung? Finden Sie ein Beispiel im Code.

---

ein Gift, mit dem der Character 20 Lebenspunkte verliert.

void poisen(Character char) char.getHealth().addHealth(-20);

---

einen Heiltrank, mit dem der Character komplett geheilt wird.

void healFull(Character char) Health h = char.getHealth(); h.addHealth(h.getMaxHealth()
- h.getCurrentHealth());

---

einen Stärkungszauber, der die maximalen Lebenspunkte eines Characters um 30 erhöht.

void increaseMaxHP(Character char) Health h = char.getHealth(); h.setMaxHealth(h.getMaxHealth()
+ 30)

3.1.3 Welche Teile des Codes mussten Sie ändern?

Durch den Character Parameter egal. Wenn der Parameter nicht erwünscht ist muss das ganze in die Character klasse rein.

3.2 Aufgabe 2

3.2.1 Beschreiben Sie die grundsätzlichen Unterschiede zwischen projektbasierter und produktbasierter Software.

Bei der Projektbasierten Software handelt es sich um ein einmaldiges Unterfangen, welches nach Spezifikationen eines Auftragsgebers stattfindet. Produktbasiete Software wird dauerhaft erweitert und sofort an den Endkunden vertrieben.

3.2.2 Wie unterscheidet sich die Motivation eines Herstellers von projektbasierter Software von der eines Herstellers von produktbasierter Software?

Bei Projektbasierter Software soll der Auftragsgeber zufriedengestellt werden und bei Produktbasierter Software verdient der Entwickler mit, die Software soll sich auf den Markt gut vertreiben.

3.2.3 Warum ist es oft wichtig, dass produktbasierte Software schnell auf den Markt gebracht wird?

Bei Produktbasierter Software verdient der Entwickler erst Geld, sobald diese am Markt ist.

3.2.4 Warum ist es eine gute Idee, zuerst einen Prototypen entwickeln, bevor man ein neues Software-Produkt erstellt?

Um zu gucken ob die Software gut am Markt ankommt und um früh Feedback zu bekommen.

3.3 Aufgabe 3

3.3.1 Welche Herausforderungen und Probleme wird Software Engineering in den nächsten 20 Jahren zu bewältigen haben?

- KI
- Das 2035 Problem
- Depression haha (: