Analyse

- **Stakeholder und Anforderungsquellen:**
  - Stakeholder ist jede Person oder Institution, die ein Projekt beeinflussen kann oder durch ein Projekt beeinflusst wird.
  - Stakeholder ist ein „Anteilhalter", der ein Interesse im Projekt vertritt und damit eine Anforderungsquelle darstellt.
  - Stakeholdergruppen sind Gruppen von Stakeholdern, die gleiche Interessen vertreten. Sie haben eine eigene Sicht auf das Projekt und einen eigenen Jargon.
  - Es kann zu Konflikten zwischen den Stakeholdern kommen, daher ist eine Priorisierung der Anforderungen wichtig.
  - Typische Stakeholder sind:
    - Kunde, Auftraggeber, Geldgeber,
    - Geschäftsführung, Abteilungsleitung,
    - Legacy Owner (Besitzer eines Altsystems oder Altdaten),
    - Betreiber/Entwickler von Schnittstellensystemen (Systeme mit denen integriert werden soll),
    - Benutzer,
    - Projektleiter,
    - Projektsteuergruppe,
    - Systemarchitekten, IT-Strategie,
    - Softwareentwickler,
    - Qualitätssicherung und Test,
    - Betrieb,
    - Wartung.
  - Weitere Anforderungsquellen sind:
    - Gesetze,
    - Standards und Normen,
    - Datenschutz,
    - Projektumfeld: Funktionalität von Konkurrenzprodukten, Unternehmenskultur,
    - verdeckte Regeln wie Glaubensgrundsätze, Tabus, Macht, Werte, Einstellungen, Status.

- **Vision und Scope**:
  - **Vision**: üblicherweise stabil; Zweck, Nutzen, Business Case.
    Beispiel: Im Jahr 2020 werden infrastrukturelle Änderungen notwendig, die Anpassungen an einem end-of-life Softwareprodukt erfordern. Von dem neuen Produkt wird erwartet, dass die 30 Jahre fachliches Wissen der alten Lösung enthalten sind, sowie dass möglichst rasch die Lizenzkosten für das alte System eingespart werden können.
  - **Scope**: (= Geltungsbereich) üblicherweise veränderbar; (fachlicher) Umfang des Projektes.
    Beispiel: Funktionsliste der existierenden Software, Sammlung von umzusetzenden Use-Cases
    Achtung: Fachlicher Scope != Scope des gesamten Projektes (beinhaltet z. B. Deployment, PM-Overhead, QS etc.
  - Nachdem sich der Umfang im Projekt ändern kann, wird üblicherweise eine Projektsteuergruppe für die Änderung des Scopes eingerichtet. → Aufgabe des Projektmanagements

- **Anforderungsvolatilität**:

- **Stabile Anforderungen**: Anforderungen, die sich (voraussichtlich) nicht oder nur geringfügig ändern werden.
- **Volatile Anforderungen**: Anforderungen, die sich (voraussichtlich) während der Anforderungsanalyse oder während des Betriebes des Systems ändern werden.
- **Systemdesignauswirkungen**:
  - Stabile Anforderungen können im simpleren Design bzw. „hard-wired" implementiert werden.
  - Volatile Anforderungen erfordern ein adaptives Systemdesign „soft-wired".
  - Stabile Anforderungen können früher detaillierter dokumentiert bzw. modelliert werden.
  - Stabile Anforderungen können im Entwicklungsprozess früher implementiert werden.

- **Baseline und Änderungen daran**:
  - Es existiert niemals ein echtes Einfrieren (Freeze) der Anforderungen. Trotzdem sind stabile Zustände in der Analyse unabdingbar, weshalb man die sogenannte Baseline einsetzt. Dabei handelt es sich um eine Spezifikation oder ein System, das offiziell reviewed und auf das sich geeinigt wurde, danach als Basis für weitere Entwicklung dient und nur durch formelle Change-Control-Procedures geändert werden kann.
  - Nach dem Setzen der Baseline wird bei Änderungen daran eine Change-Impact-Analyse und ein Re-Costing durchgeführt.
  - Achtung: Änderungen der Baseline sind Teil des Projektmanagements

- **Klassifikation von Anforderungen**:
  - **Funktionale Anforderungen**:
    - Beschreibung der Funktionen/Services, die ein System zur Verfügung stellen soll. Dabei muss geklärt werden, wie das System auf bestimmte Eingaben und Situationen reagieren soll.
    - Liefern Mehrwert für den Kunden.
  - **Nichtfunktionale Anforderungen**:
    - Anforderungen an die Funktionen/Services, die nicht deren Verhalten, sondern Vorgaben an z. B. Qualitätseigenschaften, Entwicklungsprozess, einzuhaltende Standards und Normen, gesetzliche Rahmenbedingungen beschreiben.
    - Availability, Usability, Wartbarkeit, Skalierbarkeit etc.
    - „Verschlüsseltes Abspeichern von Passwörtern", „Abfragedauer < 2 Sekunden." Etc.
  - **Domänenanforderungen**:
    - Funktionale oder nichtfunktionale Anforderungen, die von der Domäne des Systems (= Systemumfeld) definiert werden und die besondere Eigenschaften/Einschränkungen dieses Gebietes darstellen. Domänenanforderungen sind oft inhärent (= einer Sache innewohnend) und nicht explizit pro Projekt dokumentiert.
  - **Gute Anforderungen** sollen in erster Linie **komplett** und **konsistent** sein:
    - **Vollständigkeit**: Alle Anforderungen sollen abgedeckt sein.
    - **Konsistenz**: Die Anforderungen enthalten keine Konflikte und keine Widersprüche.

- **Überprüfen von Anforderungen**:
  - **Anforderungsreview**:
    - Strukturierte und systematische Überprüfung der Qualitätseigenschaften von Anforderungen.

        - Mittels Priorisierung gilt es Konflikte zwischen Stakeholdern zu erkennen und zu qualifiziert aufzulösen.
        - Die Beteiligung unterschiedlicher Stakeholder (fachliche und technische) ist erforderlich.
    - **Prototyping**:
        - Dabei wird überprüft, ob die Anforderungen umsetzbar und insbesondere auch eindeutig und vollständig sind.
    - **Abnahmetestfallerstellung**:
        - Dadurch wird die Testbarkeit (Testability) von Anforderungen erreicht und eine Prüfung auf Eindeutigkeit und Vollständigkeit ermöglicht.

- **Typische Probleme bei Anforderungen**:
    - **Verständlichkeit**: Anforderungen werden in der Sprache der Applikationsdomäne repräsentiert, die oft von Softwareingenieuren nicht verstanden wird.
    - **Selbstverständlichkeit**: Domänespezialisten sind mit der Domäne so vertraut, dass sie bestimmte Anforderungen als selbstverständlich erachten und nicht explizit darstellen und kommunizieren.
    - **Wenig Präzision**: Es ist schwierig, Anforderungen präzise zu formulieren, ohne diese schwer lesbar zu gestalten.
    - **Vermischung**: Funktionale und nichtfunktionale Anforderungen werden häufig vermischt.
    - **Zusammenführung**: Verschiedene Anforderungen werden gemeinsam beschrieben, ohne diese genau zu identifizieren und von einander abzugrenzen.
    - **Mehrdeutigkeit**: Spezifikationen in natürlicher Sprache sind oft mehrdeutig.
    - **Überflexibilität**: Die Beschreibungsart der Anforderungen (z. B. natürliche Sprache) ist oft mehrdeutig interpretierbar und somit zu flexibel, um exakte Beschreibungen zu erzeugen.

- **Qualitätsattribute von Anforderungen**:
    - **Vollständig**: Anforderungen müssen explizit beschrieben sein, es darf keine impliziten Annahmen eines Stakeholders über das zu entwickelnde System geben.
    - **Eindeutig definiert/abgegrenzt**: Präzise Definitionen helfen, Missverständnisse zwischen Stakeholdern (z. B. zwischen Entwicklern und Auftraggebern) zu vermeiden.
    - **Verständlich beschrieben**: Dadurch können alle Stakeholder mit vertretbarem Aufwand die gesamte Anforderung lesen und verstehen.
    - **Atomar**: Es darf nur eine Anforderung pro Anforderungs-ID beschrieben sein.
    - **Identifizierbar**: Jede Anforderung muss eindeutig identifizierbar sein (z. B. über eine ID).
    - **Einheitlich dokumentiert**: Die Anforderungen und ihre Quellen sollten in denselben Dokumenten stehen und gleiche Struktur haben.
    - **Notwendigkeit**: Klärung, ob die Anforderung unabdingbar ist.
    - **Nachprüfbar**: Die Anforderungen sollten mit Abnahmetestfällen überprüft werden, damit bei der Abnahme auch geprüft werden kann, ob die Anforderungen erfüllt wurden.
    - **Rück- und vorwärts verfolgbar**: Dadurch ist einerseits erkennbar, ob jede Anforderung erfüllt wurde, und andererseits ist für jede implementierte Funktionalität erkennbar, aus welcher Anforderung sie resultiert, sodass nichts Überflüssiges entwickelt wird.
    - **Priorisiert**: Die Priorität muss festgelegt werden, damit diese im Entwicklungsprozess berücksichtigt werden kann (z. B. durch die Entwicklungsreihenfolge) bzw. konkurrierende Anforderungen aufgelöst werden.

- **Anforderungsdokument**:
  - „Muss“: Absolute Festlegung einer Anforderung
  - „Darf nicht“: Absoluter Ausschluss einer Anforderung
  - „Soll“: Empfehlung für eine Anforderung. Abweichungen sind in begründeten Fällen möglich.
  - „Soll nicht“: Empfehlung für den Ausschluss einer Anforderung. Abweichungen sind in begründeten Fällen möglich.
  - „Kann“: Die Anforderung ist optional.

- **Anforderungsnachverfolgbarkeit (Traceability)**:
  - **Nachverfolgbarkeit der Quelle**: Zusammenhang zum Stakeholder, der die Anforderung eingebracht hat
  - **Nachverfolgbarkeit zu anderen Anforderungen**: Zusammenhänge zwischen interdependenten Anforderungen
  - **Nachverfolgbarkeit zu abhängigen Artefakten**: Zusammenhänge der Anforderung z. B. zu Elementen des Systemdesigns oder auch die Nachverfolgbarkeit zu den Testfällen, die diese Anforderung abdecken.

- **UML**: grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Softwareteilen und anderen Systemen.
  - **Verhaltensmodelle**: Beschreiben das dynamische Verhalten eines Systems.
    - Anwendungsfalldiagramm
    - Aktivitätsdiagramm
    - Zeitverlaufdiagramm
    - Zustandsdiagramm
    - Kommunikationsdiagramm
    - Interaktion-Übersichtsdiagramm
    - Sequenzdiagramm
  - **Strukturmodelle**: Beschreiben die statische Struktur eines Systems.
    - Klassendiagramm
    - Paketdiagramm
    - Komponentendiagramm
    - Verteilungsdiagramm
    - Kompositionsstrukturdiagramm
    - Objektdiagramm

- **User-Stories**:
  - **Vorteile**:
    - User-Stories sind kurz und prägnant und damit gut wartbar.
    - User-Stories fördern die Diskussion/Abstimmung zwischen Kunden und Entwickler während der gesamten Projektdauer.
    - User-Stories reduzieren damit das Risiko von Fehlentwicklungen aus Benutzersicht.
    - User-Stories funktionieren insbesondere in Kombination mit einer Metapher (gemeinsame Sicht auf das System).
  - **Nachteile**:
    - Ohne zugehörige Akzeptanztestfälle sind User-Stories sehr stark interpretierbar und können ein Risiko für eine erfolgreiche Abnahme darstellen.
    - User-Stories setzten eine starke Involvierung des Kunden/Benutzers während des gesamten Prozesses voraus, was jedoch oft unrealistisch ist.
    - User-Stories bzw. die erforderliche Kundeninvolvierung sind personenzentriert.

- User-Stories können kaum als Vertragsgrundlage dienen.

Entwurf

- **Ziele und Aufgaben des Entwurfs**:
  - Entwurf eines Systems, das die Anforderungen des Kunden optimal abdeckt
  - Sorgt für Umsetzung der nichtfunktionalen und funktionalen Anforderungen
  - Beschreibung des Systems auf technischer Ebene
  - Ausgangspunkt für Implementierung
  - Festlegung auf Architektur des Systems auf Bausteinebene und Gesamtsystemebene
  - Kommunikationsmittel zwischen Projekt-Stakeholdern
  - Aufteilung des Systems in funktional getrennte Blöcke: Notwendig, um Anforderungen an Skalierbarkeit und Wartbarkeit zu erfüllen.

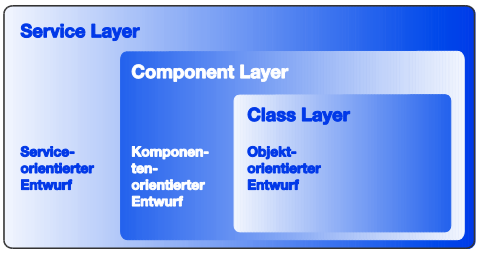
- **Softwarearchitektur**:
  - **Definition Architektur**: Eine Architektur ist die grundlegende Organisation eines Systems, verkörpert durch seine Komponenten, deren Beziehungen zueinander und zur Umwelt, sowie die Prinzipien hinter dem Design und der Evolution des Systems.
  - **Architektur**: Strategischer Entwurf
    - Architektur im Großen
    - Entwurf auf Systemebene
    - Entscheidungen über Technologie, Gesamtaufbau des Systems etc.
    - Gibt Rahmen für den taktischen Entwurf vor
  - **Design**: Taktischer Entwurf
    - Architektur im Kleinen
    - Entwurf im klassischen Softwareengineering-Sinn auf Bausteinebene und nicht auf Systemebene
    - Entwurf über den Aufbau der einzelnen Bausteine (z. B. Entwurf der Klassen etc.)
  - **Von der Anforderung zur Architektur**:
    - Nichtfunktionale Anforderungen (z. B. Zuverlässigkeit, Performanz, Sicherheit etc.) sind Faktoren hinter dem strategischen Entwurf bzw. der Architektur.
    - Funktionale Anforderungen (Systemverhalten z. B. Verarbeitung von Eingaben, Erstellung von Ausgaben etc.) sind Faktoren hinter dem taktischen Entwurf bzw. des Designs.
    - Ziel des Entwurfes ist die Abbildung eines Systems, das die Anforderungen optimal beschreibt.

- **Objektorientiertes Design**:
  - Modelle aus der objektorientieren Analyse werden übernommen und erweitert
  - Möglichst genaue Abbildung des später zu implementierenden Systems
  - Erleichtert direkte Umsetzung mit objektorientierter Sprache
  - Bei Ausschnitten/Ansichten kein Anspruch auf Vollständigkeit
  - **Verwendung folgender Diagramme**:
    - Klassendiagramme: Festlegung der Struktur; steht im Mittelpunkt
    - Zustands- und Sequenzdiagramme: Verhalten des Systems
    - Komponentendiagramme: Abbildung der Systemkomponenten und deren Zusammenhang
  - **Vorgehensweise**:
    - Identifizierung von Klassen und Objekten auf jeder Abstraktionsebene
    - Festlegung der Semantik zwischen Klassen und Objekten

        - Identifizierung der Beziehungen zwischen Klassen und Objekten
        - Spezifikation der Implementierung der Klassen

    - **Ergebnisse des Entwurfs**:
        - Festlegen einer Architektur für die Implementierung (logische und physische Architektur)
        - Policies für die Systemkomponenten festlegen

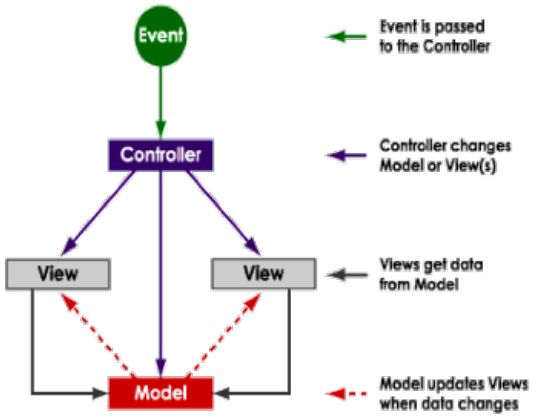
- **Serviceorientiertes Design/Serviceorientierte Architektur**:
    - Umspannt **mehrere Abstraktionsebenen** mit je unterschiedlichen Notationen:
        - Service Layer: Webservice; serviceorientierter Entwurf
        - Component Layer: Komponentendiagramme; komponentenorientierter Entwurf
        - Class Layer: UML; objektorientierter Entwurf



    - **Paradigmen der Serviceorientieren Architektur SOA**:
        - **Lose Kopplung**: Niedrige Kopplung der Services ermöglicht hohe Wiederverwendbarkeit und Flexibilität
        - **Abstraktion**: Reduktion von Neuentwicklungen – Spannungsfeld Abstraktion vs. Technische Anforderungen (z. B. können manche technische Services nicht abstrakt gelöst werden.)
        - **Standardisierung**: Erlaubt die Zusammenarbeit von Services
        - **Composable**: Standardisierte Services können zu Komponenten zusammengefasst werden.
    - **Serviceorientiere Architektur:**
        - **Business driven/Geschäftsprozessorientiert**:
            - Serviceorientierte Architekturen sind business-driven, das heißt, sie sind an Geschäftsprozesse angelehnt.
            - Im serviceorientieren Entwurf wird ein Schritt aus einem Geschäftsprozess extrahiert und gekapselt. Dabei handelt es sich um ein sogenanntes Business Service.
            - Business Porcess Modelling (BPM) und SOA sind eng miteinander verknüpft. Anwendungen werden maßgeblich von der Geschäftsprozessmodellierung beeinflusst.
        - Anwendungen werden durch mehrere Services gelöst.
        - Services aus einer Anwendung sollen auch aus anderen Anwendungen heraus aufgerufen werden können: Wiederverwendbarkeit, Modularisierung.
        - Zusammenstellung von Services in bestimmten Abläufen zur Umsetzung von Geschäftsprozessen (Orchestrierung)
        - Services existieren auf mehreren Abstraktionsebenen (Business-Service, Component-Service, technisches Service usw.).
        - Beschreibung einer Architektur, keiner Technologie

- **Patterns**:

- Patterns stellen allgemeine Lösungsstrategien für wiederkehrende Probleme der Softwareentwicklung dar.
- Patterns sind:
  - Abstrakt
  - Weitestgehend technologieunabhängig
  - Wesentlicher Input in Entwurfsentscheidungen
- **Architekturpatterns**:
  - Sollen den Entwurf vereinfachen und bekannte Probleme effizient lösen
  - Muster auf architektonischer Ebene (Makroarchitekturebene)
  - Schablonen für konkrete Softwarearchitekturen
  - Spezifizieren systemweite, strukturelle Eigenschaften einer Anwendung
  - Großer Einfluss auf die Architektur der eigenen Subsysteme
  - Fundamentale Designentscheidung
  - Z. B.: Modell-View-Controller, Pipes and Filters, Layered Architecture
  - **Modell-View-Controller-Pattern**:
    - Szenario: Interaktive Anwendungen mit einer flexiblen Schnittstelle zwischen Mensch und Computer
    - Problem: GUI wird oft geändert
    - Lösung: Aufteilung in Verarbeitung, Output und Input
    - Modell: Kapselt Daten und Funktion
    - View: Stellt Informationen dar
    - Controller: Empfängt Events



- **Designpatterns**:
  - Muster auf Klassenebene (Mikroarchitekturebene)
  - Beschreiben wiederholt auftretende Strukturen von miteinander kommunizierenden Komponenten
  - Dienen dazu, ein allgemeines Problem innerhalb eines bestimmtes Kontexts zu lösen
  - Bieten für bestimmte Programmierprobleme bewährte Lösungen
  - Patterns auf Abstraktionsebene unter Architekturpatterns
  - **Creational Patterns**: Abstract Factory, Builder, Singleton, Prototype, ...
  - **Structural Patterns**: Adapter, Bridge, Decorator, Facade, Proxy, ...
  - **Behavioral Patterns**: Command, Iterator, Observer, Strategy, ...
- **Pattern-Language**:
  - Sammlung von Entwurfsmustern
  - Schaffung einer einheitlichen Sprache für Probleme und Lösungen

Implementierung

- **Ziele und Aufgaben der Implementierung**:
  - Umsetzung der Anforderung des Kunden
  - Umsetzung der Architektur in eine Software
  - Zielgerechte Lieferung der Software
  - Entwicklung einer wartbaren Software

- **Voraussetzungen für die Implementierung**:
  - Vorhandensein einer ausführlichen Architektur
  - Definition eines Teams
  - Festlegung von Entwicklungsprozess, Programmiersprache, Tools, Zeitplan

- **Notwendigkeit einer Architektur**:
  - Architektur gibt Bauplan des Projektes vor
  - Der Bauplan teilt das Produkt in Komponenten mit Verantwortlichkeiten. Typischer weise arbeiten mehrere Personen an verschiedenen Komponenten. Daher sorgt der Bauplan für eine saubere Umsetzung.
  - Das Entwicklungsteam hat eine „Anleitung“ zur Orientierung.

- **Wahl der Programmiersprache**:
  - Kenntnisse und Erfahrung des Teams
  - Eignung der Sprache für die Problemdomäne
  - Verfügbarkeit von Entwicklungstools, Frameworks und Libraries
  - Verbreitung der Sprache und Verfügbarkeit von Entwicklern
  - Kundenvorgabe

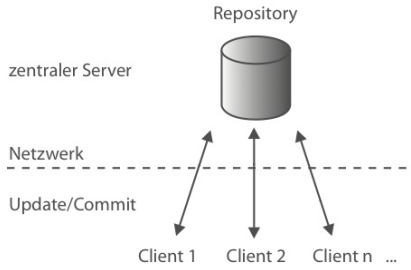
- **Framework**:
  - Applikationsskelett, das vom Entwickler angepasst/spezialisiert wird
  - **Vorteile**:
    - Wiederverwendung
    - Kürzere Entwicklungszeit
    - Geringere Kosten
    - Erhöhung der Softwarequalität
    - Die Entwicklung kann auf die Spezialisierung von vorgegebenen Strukturen reduziert werden.
  - **Nachteile**:
    - Komplexität: Black/Magic Box, daher Einarbeitung und Erfahrung nötig
  - **Auswahlkriterien**:
    - Deckt das Framework die Anforderungen ab?
    - Ist die Wartung und Weiterentwicklung gesichert? Höhere Verbreitung bedeutet meist gute Wartung, Weiterentwicklung und Stabilität.
    - Support, Garantien
    - Kosten

- **Library vs. Framework**:
  - **Library (Bibliothek)**:
    - Wiederverwendbare Funktionalität
    - Wird vom Entwickler-Code aufgerufen.
  - **Framework**:
    - Wiederverwendbares Verhalten
    - Ruft Entwickler-Code auf (Inversion-of-Control, Hollywood-Prinzip)

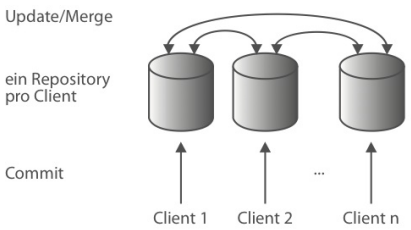
- **Versionskontrollsysteme VCS**:

- Computersystem, das Änderungen an Dateien über die Zeit hinweg archiviert
- Erlaubt jederzeit das Laden der aktuellen und jeder archivierten Version einer Datei
- **Repository**: Datenbestand, aus dem die Arbeitskopie generiert wird
- **Check-out**: Übertragung/Abholung der Daten aus dem Repository
- **Check-in/Commit**: Übertragung der Daten in das Repository
- **Verzweigung/Branching**: Anlegen einer Kopie von Objekten
- **Zentrales VCS**:
  - Client-Server-Ansatz
  - Zentraler Server mit Repository
  - Z. B. SVN
- **Dezentrales VCS**:
  - Kein zentrales Repository
  - Jeder Entwickler besitzt ein eigenes Repository
  - Änderungen werden zwischen den Repositories ausgetauscht
  - Z. B. Git

Zentrales VCS:



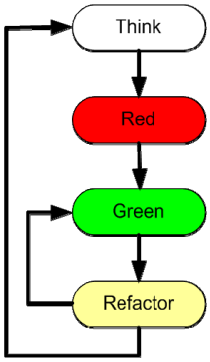
Dezentrales VCS:



- **Build-Management-Systeme**:
  - Werkzeug zum automatisierten Bauen von Applikationen oder Teilen davon
  - Stellt sicher, dass benötigte (Sub-) Module verfügbar sind
  - Stellt die Abhängigkeiten zwischen den Komponenten sicher
  - Z. B. Maven, Gradle, Ant, Make

- **Kopplung und Kohäsion**:
  - **Kopplung**:
    - Maß der Abhängigkeit zwischen Komponenten
    - Hohe Kopplung ist schlechter Programmierstil
  - **Kohäsion**:
    - Maß der Zusammengehörigkeit innerhalb einer Komponente
    - Hohe Kohäsion durch Methodengruppierung nach Zweck gewährleistet
    - **Single-Responsibility-Prinzip**: Eine Komponente erfüllt genau eine Aufgabe.
    - **Don't-Repeat-Yourself-Prinzip**: Nur diese eine Komponente erfüllt diese Aufgabe.
  - **Ziel: Geringe Kopplung, erreicht durch hohe Kohäsion**

- **Schnittstellen und Information-Hiding**:
  - Saubere Definition von Interfaces zwischen Komponenten (Klassen, Modulen etc.)
  - Saubere Schnittstellen sind auch für die Testbarkeit eine Komponente wichtig.
  - **Information Hiding**:
    - Eine Komponente soll von einer anderen Komponente nur so viel als nötig wissen (z. B. gehaltene Daten, Datenstrukturen, Algorithmen).
    - Kommunikation zwischen Komponenten über Interfaces
    - Siehe Datenkapselung bei OOP



- **Test-driven-Development TDD:**
  1. **Think**:
     - Auswahl der zu implementierenden Anforderungen
     - Spezifikation der Testfälle
  2. **Red**:
     - Implementierung und Ausführung aller Testfälle
     - Alle Tests müssen fehlschlagen.
  3. **Green**:
     - Implementierung der Komponenten und Klassen und Ausführung aller Testfälle
     - Testfall erfolgreich: Weiter bei Schritt 4
     - Testfall schlägt fehl: Weiter bei Schritt 3
  4. **Refactor**:
     - Änderung und Optimierung der Implementierung ohne Änderung der Funktionalität; Ausführung der Testfälle
     - Testfälle dürfen nicht fehlschlagen.
     - Weiter bei Schritt 1 und der Auswahl der nächsten Anforderung

Integration und Testen

- **Stellenwert von Softwaretests**:
  - Keine Software ist frei von Fehlern.
  - Testen gehört zu den wichtigsten Aktivitäten des Entwicklungsprozesses.
  - Systematisches Testen ist notwendig.
  - Eine gut geplante Testphase ist die Grundlage für ein erfolgreiches Projekt.
  - Ziel ist es, Fehler so früh als möglich zu finden.
  - Softwaretests sind eine dynamische und produktorientierte Qualitätssicherungstechnik.
  - Aufgrund wachsender Komplexität und hoher Abhängigkeiten von Softwaresystemen wird der Softwaretest strategisch immer bedeutender.

- **Grundsätze des Softwaretests**:
  - Testen zeigt Anwesenheit von Fehlern
  - Vollständiges Testen ist nicht möglich
  - Mit dem Testen frühzeitig beginnen
  - Häufung von Fehlern
  - Wiederholungen haben keine Wirksamkeit
  - Testen ist abhängig vom Umfeld
  - Trugschluss: Keine Fehler bedeutet ein brauchbares System

- **Ziele und Methodik von Softwaretests**:
  - **Ziele**:
    - **Verifikation**:
      - „Bauen wir das Produkt richtig?“
      - Prüfung des Systems gegen seine Spezifikation
      - Umsetzung im Vergleich zur Spezifikation in vorangegangenen Phasen
      - Test der Umsetzung gegen die Spezifikation
    - **Validierung**:
      - „Bauen wir das richtige Produkt?“
      - Prüfung des Systems gegen die (Kunden-) Anforderung

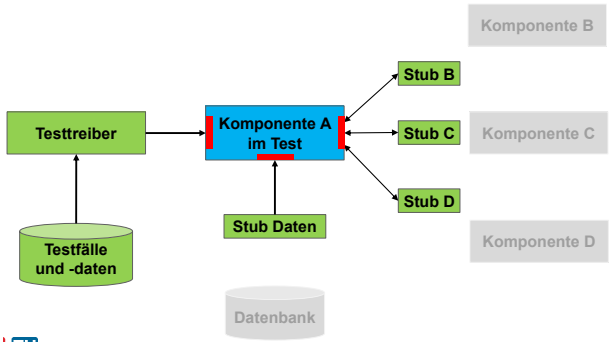
- Entspricht die Lösung und damit die Spezifikation dem, was sich der Kunde wünscht?
- Test der Umsetzung gegen die Nutzeranforderungen

- **Methodik**:
  - Jene Testfälle identifizieren, mit denen die höchste Wahrscheinlichkeit gegeben ist, festzustellen, ob das Softwaresystem korrekt funktioniert.
  - Möglichst hohe funktionale oder nichtfunktionale Abdeckung

- **Test- und Integrationsstufen**:

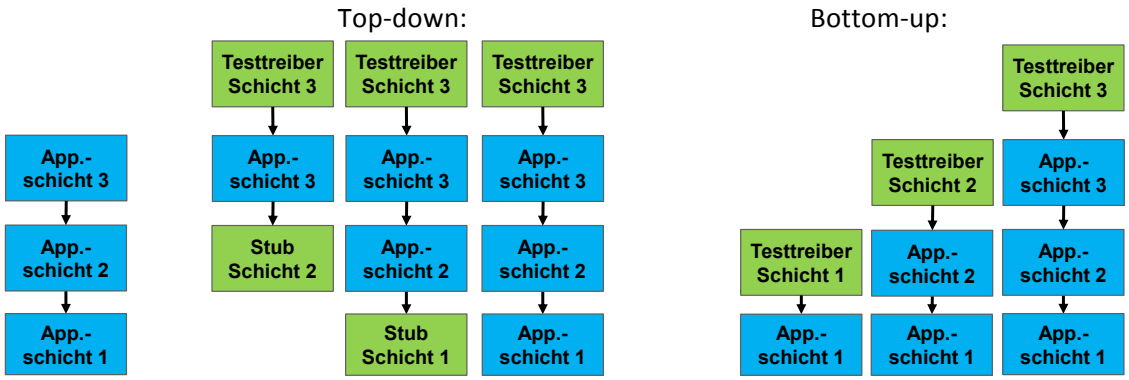
Projektphase	Teststufe
Architekturdesign	Systemintegrationstest
Technisches Design	Komponentenintegrationstest
Modul-/Klassendesign	Komponenten-/Unit-Test

- **Komponententest**:
  - Testen von funktionalen und nichtfunktionalen Aspekten einer Komponente (Modul, Klasse, ...), z. B.: Robustheit, strukturelle Prüfung (z. B. Zweigüberdeckung), Umgang mit Ressourcen, ...
  - Grundlage für Komponententests:
    - Dokumentation
    - Datenmodell
    - Quellkode
    - Kooperation mit Entwicklern
  - Test-Setup:
    - Isolation der Komponente vom Rest des Systems
    - Erst saubere Schnittstellen ermöglichen (isoliertes) Testen



- **Integrationstest**:
  - Prüft Schnittstellen und Interaktion zwischen Komponenten
  - **Big-Bang-Integration**:
    - Nichtiterative Integrationsform
    - Gleichzeitige Kombination aller Komponenten zu einem Gesamtsystem
    - **Vorteile**:
      - Keine Stubs
      - Keine Testtreiber
    - **Nachteile**:
      - Erst spät im Projekt möglich (Risiko)
      - Fehleridentifizierung und –lokalisierung schwierig
  - **Horizontale Integration**:
    - Integration Schicht für Schicht
    - Iterative, schichtorientierte Integrationform

- **Top-down**:
  - **Vorteil**: Externe Schnittstellen früh verfügbar
  - **Nachteil**: Hoher Simulationsaufwand (Stubs)
- **Bottom-up**:
  - **Vorteil**: Keine Stubs notwendig, dafür mehr Testtreiber, die allerdings wendiger Aufwand sind
  - **Nachteil**: Externe Schnittstellen spät verfügbar

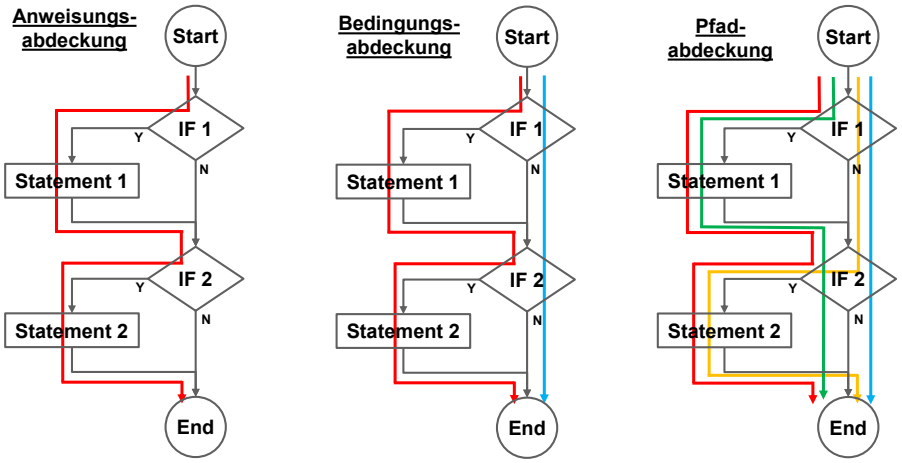


- **Vertikale Integration**:
  - Iterative, funktionsorientierte Integrationsform
  - Die Integration von Komponenten erfolgt:
    - anhand von Szenarien/Use-Cases,
    - mit den Komponenten, die für die zu testende Funktionalität notwendig sind,
    - über Systemschichten hinweg.
  - **Vorteile**:
    - Funktionalität für Szenario/Use-Case ist früher verfügbar/abnehmbar
    - Nach jeder Iteration: Lauffähiges, testbares Teilsystem vorhanden (Scrum)
  - **Nachteile**:
    - Alle Schichten müssen Funktionalität unterstützen
    - Synchronisierung in der Entwicklung notwendig

- **Systemtest, Akzeptanztest (Abnahmetest)**:
  - **Systemtest**:
    - Testet das spezifizierte Verhalten des Gesamtsystems
    - Testumgebung äquivalent zu Produktivumgebung
    - Abdeckung von funktionalen und nichtfunktionalen Anforderungen
  - **Akzeptanztest**:
    - Meist vom Auftraggeber durchgeführt
    - Prüft, ob Auftragnehmer die vereinbarten Leistungen umgesetzt hat
    - Verschiedene Arten:
      - Anwender-AT: Prüft Benutzbarkeit für Anwender
      - Betrieblicher AT: Backup- und Restore-Möglichkeit, Benutzerverwaltung, ...
      - Regulatorischer/vertraglicher AT: Gesetzes-, Standardkonformität
      - Feldtest

- **Funktionale Softwaretests**:
  - **Ziel**: Funktionale Verifikation des Systems

- **Methodik**: Mittels ausgewählter Testfälle möchte man eine hohe Abdeckung des Systems (Coverage) erreichen, die mit hoher Wahrscheinlichkeit Fehler findet.
- **Strukturelle Abdeckung**: **White-Box-Test**
  - Testfälle werden aus der inneren Struktur des Systems heraus abgeleitet.
  - Analyse des Kontrollflussgrafen (Quellcode)

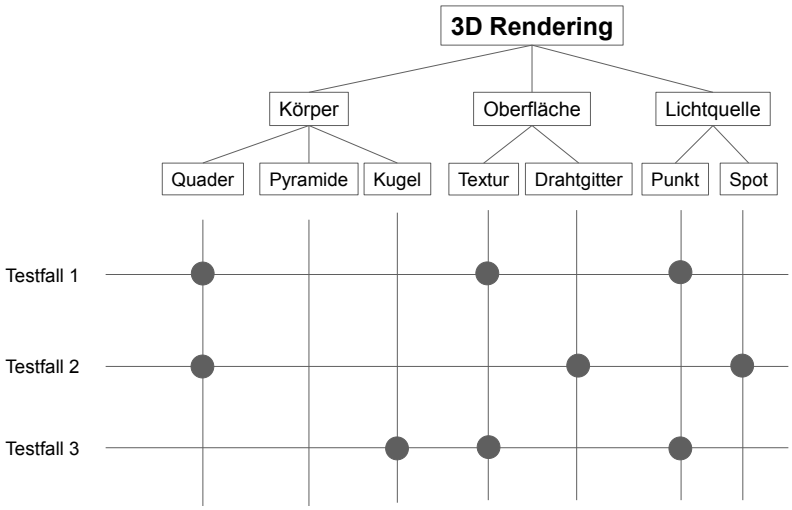


1 Testfall 2 Testfälle 4 Testfälle

  - **Anweisungsüberdeckung**:
    - **Ziel**: Abdeckung aller Knoten im Kontrollflussgrafen
    - 100 % Anweisungsüberdeckung erreicht, wenn sämtliche Anweisungen mindestens einmal durchlaufen werden
    - Aufspürung von nicht ausführbaren Anweisungen (totem Code) möglich
    - Zu schwaches Kriterium für sinnvolle Testausführung: Auftreten der Fehlerwirkung kann an die Ausführung mit bestimmten Testdaten gekoppelt sein
  - **Bedingungsüberdeckung**:
    - **Grundidee**: Gründliche Überprüfung zusammengesetzter Entscheidungen, Teilentscheidungen
    - **Einfacher Bedingungsüberdeckungstest**: Fordert den Test aller atomaren Teilentscheidungen gegen true und false
    - **Mehrfacher Bedingungsüberdeckungstest**: Fordert den Test aller Wahrheitswertekombinationen der atomaren Teilentscheidungen
  - **Pfadüberdeckungstest**:
    - Fordert die Ausführung aller unterschiedlichen Pfade der Software
    - Ein Pfad ist eine Sequenz von Knoten.
    - Für reale Softwaremodule meist nicht durchführbar, da sie eine unendlich hohe Anzahl von Pfaden besitzen können
    - Bei endlicher Anzahl von Pfaden ist der Aufwand meist trotzdem zu hoch.

- **Funktionale Abdeckung**: **Black-Box-Test**
  - Ableitung der Testfälle aus den Spezifikationen
  - **Äquivalenzklassenanalyse**:
    - Einteilung der möglichen Eingabewerte in Äquivalenzklassen mit gleichem erwartbaren Systemverhalten
    - Durchführung der Tests für einen Repräsentanten der Klasse

- **Grenzwertanalyse**: Spezialfall von Äquivalenzklassenanalyse
  - Fehler treten oft an den Grenzen der Äquivalenzklassen auf.
  - Identifikation der Werte, die die Grenze zwischen den Äquivalenzklassen bilden
  - Grenzwerte werden als Klassenrepräsentanten gewählt. Je Klassengrenze ein Testfall.
- **Zustandsbasierte Testfälle**:
  - Ableiten der Testfälle aus dem Zustandsautomat (z. B. UML-Diagramm)
  - Alle Zustände und alle Zustandsübergänge durchlaufen
  - Alle Auslöser von Zustandsübergängen durchführen
- **Klassifikationsbaummethode**:
  - Einteilung der Testfälle in Klassifikationen (testrelevante Aspekte)
  - Jede Klassifikation in Äquivalenzklassen zerlegen
  - Testfall ist eine Kombination aus Äquivalenzklassen; eine Äquivalenzklasse je Klassifikation.



- **Informelle Testmethoden**:
  - Keine oder nur teilweise systematische Ableitung von Testfällen

- **Nichtfunktionale Softwaretests:**
  - Testen der nichtfunktionalen Eigenschaften des Systems
  - Test abhängig von der untersuchten Eigenschaft
  - Mehr Expertenwissen notwendig
  - Z. B. Benutzbarkeit (Usability), Leistungsfähigkeit (Performance), Sicherheit (Security)

- **Testautomatisierung:**
  - Automatisiertes Ausführen von Softwaretests
  - Initialaufwand deutlich höher als bei manuellen Tests
  - Testdurchführungsaufwand deutlich niedriger als bei manuellen Tests
  - Hoher Wartungsaufwand
  - **Automatisierter Komponententest**:
    - Tests in der Programmiersprache des Zielsystems
    - Gute Framework-Unterstützung (z. B. JUnit, TestNG)
  - **Automatisierter GUI-Test**:
    - Simulation eines Benutzers
    - Capture/Replay-Verfahren

        - **Nachteile**: hoher Wartungsaufwand, Testobjekt muss bereits verfügbar sein
        - **Alternative**: Scripting, hybrider Ansatz
    - **Regressionstest**: Ist die Systemfunktionalität nach Änderungen noch gewährleistet? Z. B. nach Änderung an Quellcode oder Konfiguration, Tausch einer Komponente etc.
    - **Kontinuierliche Integration (Continuous Integration)**: Automatisiertes Bauen und Testen des Systems nach jeder Änderung

Inbetriebnahme

- **Anwenderakzeptanz,** Integration der Systemanwender wird erreicht durch:
    - **Anwendergerechte Dokumentation**: Benutzerhandbuch, online Hilfe
    - **Schulungen**: Meist mehrere Benutzergruppen mit unterschiedlichen Funktionalitäten/Berechtigungen
    - **Rollout**:
        - Big-Bang-Rollout ist zu vermeiden
        - Testbetrieb mit einer kleinen Benutzergruppe bereits während der Entwicklung
        - Parallelbetrieb

- **Migration**:
    - Ablöse eines bestehenden Systems durch ein neues System
    - Laufenden Betrieb nicht beeinflussen (unterbrechungsfrei)
    - **Arten von Migration**:
        - **Hardwaremigration**
        - **Softwaremigration**
        - **Datenmigration**: Transfer von Daten in ein neues System
        - **Kombination**: insbesondere Software- und Datenmigration

- **Inbetriebsetzung** und laufender Betrieb:
    - Planung!
    - Eigene Maschine(n)?
    - Datenbank
    - Integration von Fremdsystemen
    - Verantwortlichkeiten klären:
        - Installation der Software
        - Einrichten der Datenbank
        - Benutzerkonten, Netzwerkfreigaben
    - Dokumentation:
        - Release-Checkliste
        - Release-Drehbuch
    - Best-Practises:
        - Zusammenarbeit Entwicklung und Betrieb
        - Verantwortlichkeiten festlegen
        - Frühzeitiges Testrollout
    - Anti-Patterns:
        - Manuelle Installation der Software
        - Installation erst nach Abschluss der Entwicklung
        - Manuelle Konfiguration
        - Änderungen direkt in der Betriebsumgebung („Hot-Fixes“)

Wartung

- **Definition Softwarewartung**: Sämtliche Änderungen einer Software zur
  - Fehlerbehebung,
  - Verbesserung gewisser Eigenschaften (z. B. Performance),
  - Anpassung an geänderte Umgebung.
  - Änderungen erfolgen nach der Lieferung.
  - Änderungen verändern die Funktionalität nicht.

- **Auslöser von Wartungstätigkeiten**:
  - Gefundene Fehler
  - Performance-Verbesserungen
  - Updaten der verwendeten Frameworks und Bibliotheken
  - Behebung von Sicherheitslücken
  - Optimierung der Konfiguration

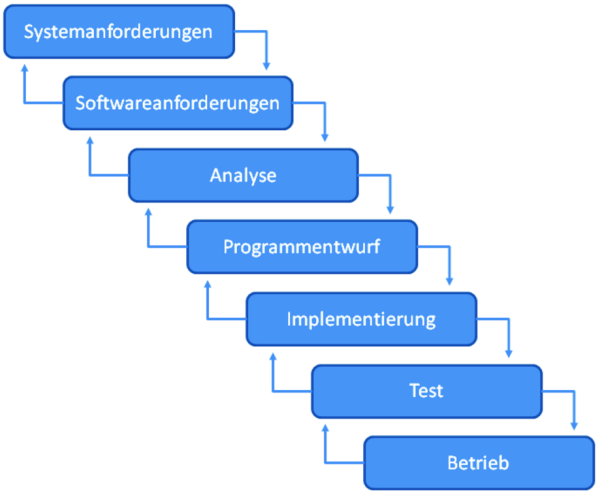
- **Wartungstypen**:
  - **Korrektive Wartung**: Ausbessern von aufgetretenem Fehler
  - **Präventive Wartung**: Ausbessern von Fehler, bevor dieser auftritt
  - **Adaptive Wartung**: Anpassung an eine geänderte Umgebung, Investitionsschutz
  - **Perfektionierende Wartung**: Verbesserung des Systems z. B. Performance, Usability, Speicherverbrauch etc.

- Softwarewartung **How-To's**: Typische Maßnahmen während der Wartung
  - **Reengineering**:
    - Neuentwicklung eines (funktionierenden) Systemteils bei gleichbleibender Funktionalität
    - Ziel: Qualitätssteigerung, Vorbereitung für Weiterentwicklung
  - **Refaktorisierung**:
    - Umstrukturierung bei gleichbleibender Funktionalität
    - Ziel: Erhaltung/Erhöhung der Wartbarkeit
  - **Eliminierung von Anti-Patterns**:
    - Erkennen und Entfernen von schlechten Softwareteilen

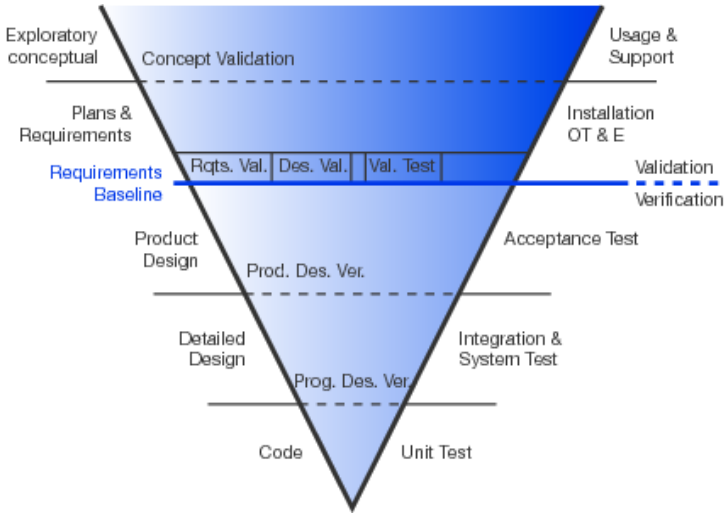
Vorgehensmodelle

- **Grundlagen und Bedeutung**:
  - Softwareprodukte werden immer umfangreicher und komplexer.
  - Effektive und effiziente Softwareentwicklung sind wesentlich am Geschäftserfolg des Softwareunternehmens beteiligt.
  - Ziel ist es, die Softwareentwicklung auf eine etablierte, ingenieurmäßige Basis zu stellen, um Verbesserungen in der Produktivität, Qualität und Vorhersagbarkeit zu erreichen.
  - Es existiert kein optimaler Entwicklungsprozess.

- **Traditionelle Modelle**:
  - **Wasserfallmodell nach Royce** (1970):
    - In der Grundform vollständig sequenziell
    - Lässt in einer Erweiterungsstufe den Rückschritt von einer Phase auf den direkten Vorgänger zu.
    - Ein Arbeitsschritt kann erst dann abgeschlossen werden, wenn alle vorgesehenen Produkte fertiggestellt wurden.
    - Berücksichtigt wesentliche Dokumente und Review-Schritte im Modell
    - Wird in der Softwareentwicklung als risikoreich erachtet.



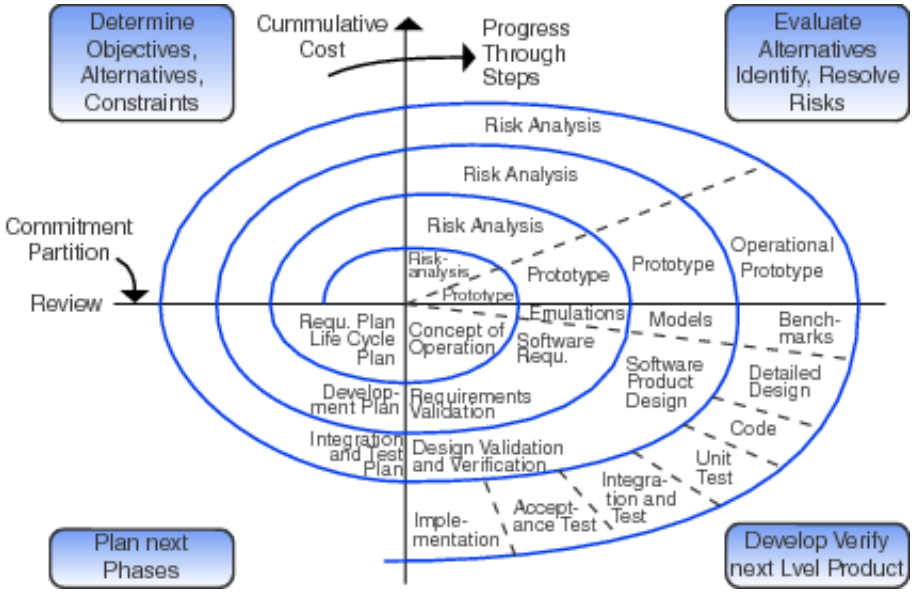
- **Fundamentales V-Modell nach Böhm**:
  - Basiert auf korrelierenden Phasen sowie Verifikation und Validierung
  - Anforderungen steuern Entwicklungsprozess
  - Technische Realisierung wird pro Ebene immer detaillierter
  - **Linke Seite**:
    - Funktionale und fachliche Spezifikationen
    - Ausbau zu technischer Spezifikation und Implementierungsgrundlage
    - Je tiefer, umso detaillierter
  - **Spitze**:
    - Implementierung
  - **Rechte Seite**:
    - Test gegen die jeweilige Spezifikation auf der linken Seite



- **V-Modell 97**:
  - Entwicklung erfolgt inkrementell
  - Zu Beginn wird die Gesamtfunktionalität abgegrenzt und die einzelnen Funktionen des Softwaresystems werden priorisiert.
  - Die einzelnen Funktionen werden gemäß ihrer Priorität Ausbaustufen zugeordnet und im Rahmen dieser umgesetzt.
  - Das Ergebnis einer Stufe wird den Endanwendern zur Verfügung gestellt.

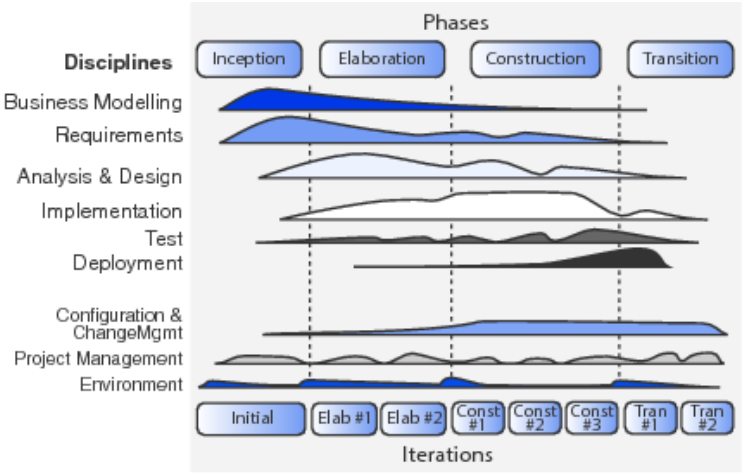
- Der Umfang des Systems wächst ausgehend von einer Basisfunktionalität stetig an.

- **V-Modell XT**: eXtreme Tailoring
  - Salierbarkeit hinsichtlich verschiedener Projektgrößen (reduzierte Menge an Aktivitäten und Produkte für kleinere Projekte)
  - Mehr Orientierung Richtung agilem Vorgehen: Keinerlei Vorschriften über die Zeitliche Abfolge von Vorgehensbausteinen
- **Spiralmodell nach Böhm** (1988):
  - Es berücksichtigt die möglichen Projektrisiken der sequenziellen Entwicklung und ist daher iterativ aufgebaut.
  - Der gesamte Prozess ist in vier Phasen gegliedert:
    1. **Definition von Zielen**
    2. **Risikoanalyse**
    3. **Durchführen der Arbeitsschritte**
    4. **Planen der nächsten Phase**
  - Die Anzahl der notwendigen Durchläufe ergibt sich erst im Projektverlauf und ist von den auftretenden Risiken abhängig.

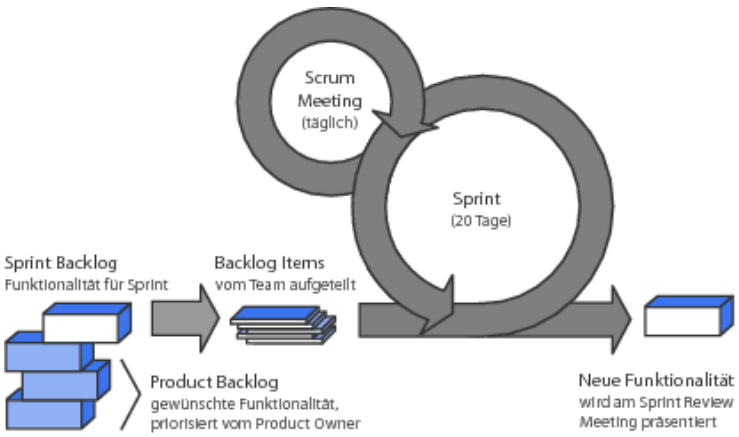


- **Rational Unified Process von IBM** (1998):
  - Industriell gepflegtes und ständig in Weiterentwicklung befindliches Prozess-Framework
  - Eines der am meist verbreitetsten und detailliert beschriebenen Prozessmodelle
  - Beruht auf folgenden Vorgehensweisen:
    - Software iterativ entwickeln
    - Anforderungen managen
    - Komponentenbasierte Architektur verwenden
    - Software visuell modellieren (UML)
    - Softwarequalität kontinuierlich prüfen
    - Änderungen kontrolliert in die Software einbringen
  - Tailoring:
    - Für jedes Projekt bzw. Organisation ist eine selektive Adaption vorzunehmen, um den erwarteten Nutzen auch tatsächlich zu erzielen.

- Für Anpassungen (Tailoring) des RUP bietet sich das kostenpflichtige Tool IBM Rational Method Composer (RMC) an mit u. A. vordefinierten Prozessmodellen
- Die Rolle, den RUP an die jeweiligen speziellen Rahmenbedingungen anzupassen, kommt dem Process Engineer zu.



- **Agile Modelle**:
  - **Agile Manifesto**:
    - *Individuals and interactions* over processes and tools
    - *Working software* over comprehensive documentation
    - *Customer collaboration* over contract negotiation
    - *Responding to change* over following a plan
  - **Scrum**:
    - Hohe Produktivität und Anpassungsfähigkeit
    - Wenig Risiko und Ungewissheit
    - Größerer Komfort für die Projektmitglieder



- **Scrum-Elemente**:
  - **Sprint**: Innerhalb eines Sprints wird versucht, so schnell als möglich zum zuvor definierten Sprint-Goal zu kommen.
  - **Daily Scrum**: Dieses Meeting findet jeden Tag zur selben Zeit am selben Ort statt.

- **Product-Backlog**: Diese nach Dringlichkeit geordnete Liste der Anforderungen kann von allen Projektbeteiligten jederzeit eingesehen werden.
- **Increment**: Bei Scrum entsteht ein Softwareprodukt in Inkrementen, wobei schon der erste, aber spätestens der zweite Sprint ein lauffähiges Softwaresystem (mit Kernfunktionalität) liefert.
- **Burndown-Chart**: In dieses Diagramm werden vom Scrum-Master tagesaktuelle Aufwandsschätzungen für den aktuellen Sprint eingetragen.

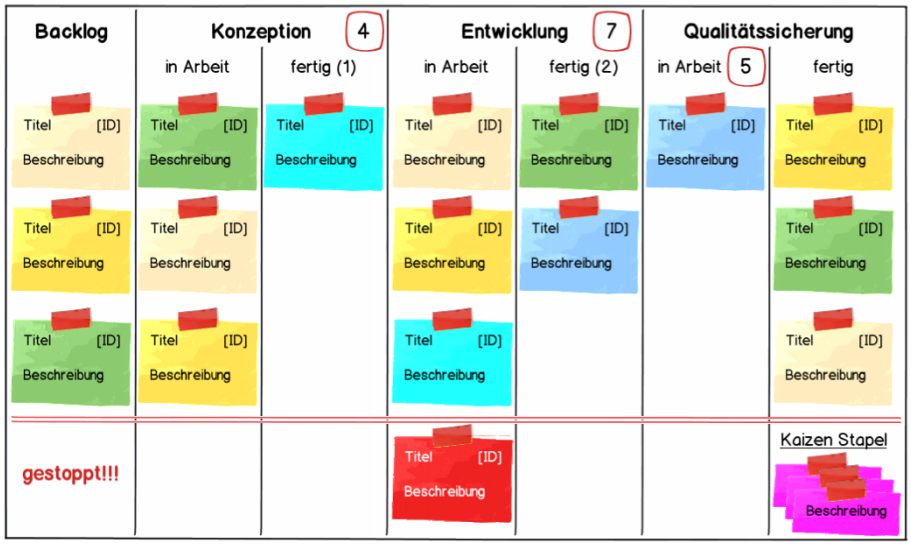
▪ **Scrum-Rollen**:
- **Product-Owner**:
  - Erstellung und Priorisierung des Product- und Release-Backlog
  - Verantwortlich dafür, dass das Team frei von Störungen und Unterbrechungen bleibt
- **Scrum-Team**:
  - Die Mission des Teams ist die Umsetzung der über den Sprint-Backlog geforderten Funktionalität
  - Bzw. die Erreichung des über das Sprint-Goal bestimmten Ziels.
- **Scrum-Master**:
  - Die wesentliche Aufgabe dieser Rolle besteht darin, die Dynamik von Scrum zur ungestörten Entfaltung zu bringen,
  - indem der Scrum-Master einerseits auf die Umsetzung der Grundsätze und Regeln von Scrum achtet
  - und andererseits das Team von äußeren Einflüssen abschottet.

o **eXtreme Programming XP**:
- Kompakte Methode zur Entwicklung von Software in kleinen bis mittelgroßen Teams, deren Arbeit vagen oder sich rasch ändernden Anforderungen unterliegt.
- Das Hauptziel ist das termingerechte Abliefern von auftragsgemäßer Software.
- Charakteristisch für XP sind das eigene Wertesystem, viele Prinzipien und ein System von Techniken, die einander gegenseitig im höchsten Maße unterstützen.
- **Prinzipien**:
  - **Kommunikation**: ständige Kommunikation zwischen allen Beteiligten (Entwickler, Kunden, Sponsoren, Benutzer, ...)
  - **Einfachheit**: das einfachste Design, das die Aufgabe löst
  - **Feedback**: wissen, wo man steht und was man verbessern kann (empirisches Prozessmodell)
  - **Mut**:
    - Zur Offenheit und Transparenz
    - Notwendige Änderungen vorzunehmen
  - **Respekt**: vor sich selbst, dem Team und dem Produkt
- **Kombination mit Scrum**: Scrum als Vorgehensmodell mit XP-Entwicklungstechniken

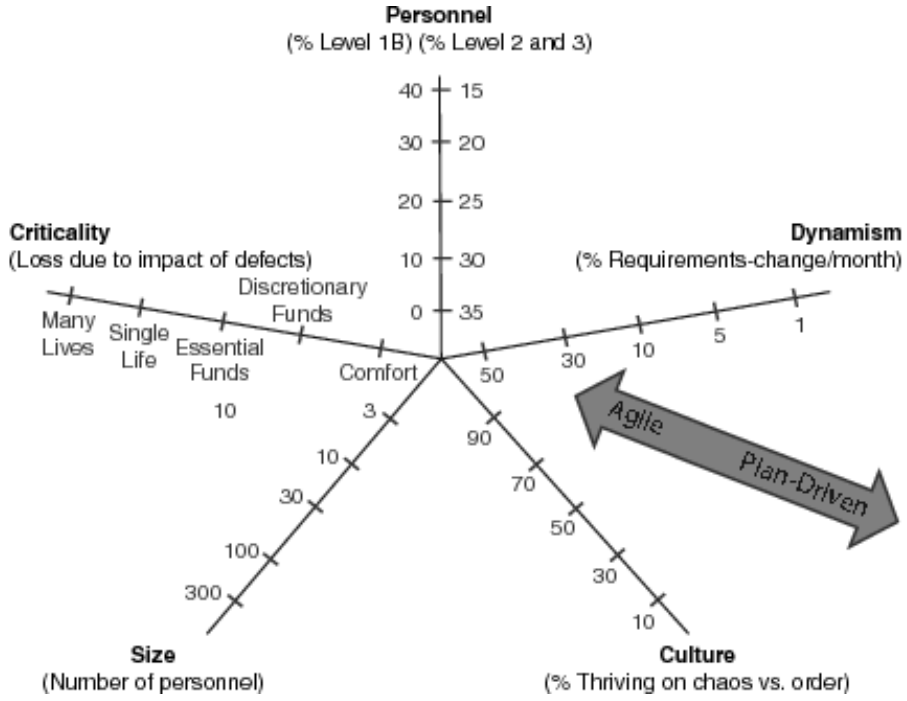
o **Kanban**:
- **Prinzipien**:
  - Eliminate waste
  - Amplify learning

- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrety in
- See the whole

- Keine vorgeschriebenen Rollen, Meetings, Artefakte außer:
  - Kontinuierlicher „Pull"-Workflow
  - Kanbanboard
  - Work-in-Progress-Limit



- **Entscheidungsfaktoren nach Böhm und Turner**:
  - **Teamstruktur (Personnel)**: Die Qualifikation des Teams muss bei agilen Methoden höher sein als bei plangetriebenen Methoden.
  - **Dynamik und Anforderungen (Dynamism)**: Je häufiger Anforderungen geändert werden, umso einfach muss dies möglich sein. (Oft Änderungen → agile Methoden)
  - **Entwicklungskultur (Culture)**: Je nach Organisatorischem Umfeld bzw. Entwicklungskultur können agile Methoden eingesetzt werden. (Chaos → agil, Regeln → Plan)
  - **Teamgröße (Size)**: Agile Methoden sind in kleinen bis mittelgroßen Team effizient einsetzbar.
  - **Kritikalität (Criticality)**: Wenn Menschenleben von einem Softwaresystem abhängig sind, werden höhere Ansprüche an die Prüfung und Nachvollziehbarkeit auf allen Ebenen gelegt. (hoch → Plan, niedrig → agil)



Qualitätssicherung

- **Statische Qualitätssicherung**:
  - **Analytische Aktivitäten** zur Prüfung des Testobjektes, ohne dass dieses dynamisch ausgeführt wird
  - Prüfung von Testobjekten bereits sehr **früh im Projektlebenszyklus** möglich noch bevor ausführbarer Code verfügbar ist
  - Statische Codeanalysen ermöglichen die Prüfung einzelner Programmteile bereits lange **vor der Integration** zu einem Gesamtsystem.
  - Mit Hilfe der **Strukturanalyse** kann die **innere Qualität** ermittelt und über den gesamten Entwicklungsprozess auf hohem Niveau gehalten werden. Basis der Strukturanalyse sind **Abhängigkeitsgrafen** und **Qualitätsmetriken**.
  - Überprüfung der Einhaltung von **Entwicklungsrichtlinien** durch syntaktische Prüfung
  - **Fehlermusteranalyse** prüft den Quellcode gegen typische Fehlermuster.
  - **Softwaremetriken**:
    - Ermöglichen, die Qualität von Software zu interpretieren
    - **Prozessmetriken**: quantitative Daten des Softwareentwicklungsprozesses
      - **Fehler**: Wie schnell werden Fehler im Prozess entdeckt?
      - **Aufwände**: Welche Aufwände verursachen sie? (Mitarbeiter-, Zeit-, Kostenaufwände, ...)
    - **Produktmetriken**: messen Software oder Teile davon
      - **Umfangsmetrik**: Lines of Code
      - **Entwurfsqualität**: Modularität, Kopplung, ...
      - **Produktqualität**: Testergebnisse, Testabdeckung, ...
  - **Reviews**: Formell organisierte Zusammenkunft von Personen zur inhaltlichen oder formellen Überprüfung eines Produktteils nach vorgegebenen Prüfkriterien
    - **Vorteile**:
      - Frühe Durchführung im Entwicklungsprozess
      - Verkürzung der Latenzzeit eines Fehlers
      - Geringere Fehlerbehebungskosten, die Review-Aufwand kompensieren
    - **Rollen**:

- **Manager**:
  - Jener Projektleiter, der den Auftrag zur Erstellung des Testobjektes gegeben hat
  - Verantwortlich für die Freigabe
- **Moderator, Review-Leiter**:
  - Plant, organisiert und leitet das Review
  - Sucht Teilnehmer aus
  - Stellt benötigte Unterlagen des Softwareelements bereit
  - Organisiert die Review-Sitzung
- **Schreiber**:
  - Führt Protokoll
  - Notiert Erkenntnisse des Review-Teams
- **Autor**:
  - Urheber des Testobjekts oder Repräsentant des Teams, das das Testobjekt erstellt hat
- **Gutachter oder Reviewer**:
  - Review-Mitglieder, die in der Vorbereitung das zu prüfende Material begutachten und in der Sitzung ihre Erfahrungen berichten
- **Leser**:
  - Aufbereitung der zu prüfenden Softwareelemente
  - Verantwortlich für zeitlich angemessene Durchführung der Sitzung
  - Geht das zu prüfende Material durch, soll aber explizit nicht Autor sein

- **Typen:**
  - **Stellungnahme**:
    - Der Autor stellt einem oder mehreren Kollegen Kopien zum Lesen zur Verfügung.
    - Diese begutachten die Unterlagen und retournieren die Papiere mehr oder weniger ausführlich kommentiert an den Autor.
    - Anschließend analysiert er die Stellungnahmen und korrigiert, falls er es für notwendig erachtet, seine Arbeit.
  - **Walkthrough**:
    - Weniger formelle Form des Review
    - Regen Diskussionen bzw. zu einer hohen Interaktion zwischen dem Vortragenden und den Teilnehmern an
  - **Technisches Review**:
    - Beurteilung eines Softwareprodukts durch ein qualifiziertes Team hinsichtlich der beabsichtigten Verwendung
    - Identifikation von Abweichungen von der Spezifikation oder von verwendeten Standards
    - Starke Anlehnung an Inspektion
  - **Inspektion**:
    - Formellstes und wirkungsvollstes Review-Verfahren
    - Zweck einer Inspektion ist das Finden und die Identifikation von Anomalien im Quellcode.
    - Es soll ein Softwareelement auf die Erfüllung seiner Spezifikation und die Einhaltung von Richtlinien und Standards untersucht werden.
  - **Round-Robin-Review**:

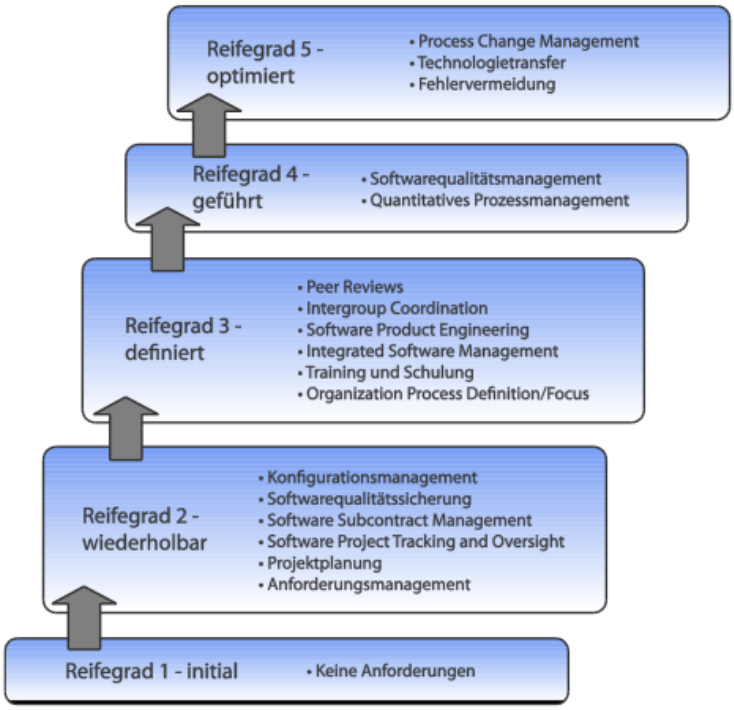
        - Versucht, das Prinzip eines herkömmlichen Reviews aus den Kopf zu stellen
        - Gutachter erhalten abwechselnd die Möglichkeit, die Führungsrolle zu übernehmen (verschiedene Testobjekte)
    - **Peer-Review**:
        - Der Unterschied zur Inspektion ist, dass der Prozess nicht definiert ist und keine speziellen Techniken zur Fehlersuche eingesetzt werden.
        - Das Team, ad-hoc zusammengestellt oder als permanente Einrichtung, bestimmt selbst die Aufgabenaufteilung und Vorgehensweise.

- **Dynamische Qualitätssicherung**:
    - Prüfen und Überwachen des Systemverhaltens gegenüber dem spezifizierten Verhalten **während der Laufzeit**
    - Das System wird mit Testdaten ausgeführt und beobachtet.
    - Dynamische Analyse:
        - Untersuchung der Interaktion verschiedener Komponenten
        - Das zu analysierende Programm wird mit speziellen Werkzeugen ausgeführt (Debugger).
        - Vorgang des Nachverfolgens von Information (Programm-Trace)
        - Information darüber, an welcher Stelle im Programm Unregelmäßigkeiten aufgetreten sind
- **Organisatorische Qualitätssicherung**:
    - **Ziel**: Bereitstellung von Infrastrukturkomponenten zur Vermeidung von Softwarefehlern oder zumindest zur Verringerung der Fehlerrate
    - **Wissensmanagement**:
        - Kann als Prozess der Weiterentwicklung von Ideen des organisationalen Lernens verstanden werden
        - Der Fokus liegt auf der Verbesserung einer Organisation auf allen Ebenen durch den gezielten Umgang mit der Ressource Wissen, insbesondere im Bereich des Qualitätsmanagements.
        - Dieses Wissen umfasst sämtliche Bestandteile, über die eine Organisation zur Lösung ihrer Aufgaben verfügt, z. B. Fähigkeiten, Erfahrungen, Fertigkeiten, Normen, Routinen.
    - **Konfigurationsmanagement**:
        - Erstreckt sich über alle Softwareentwicklungsaktivitäten im Laufe des Softwarelebenszyklus
        - Verwaltet sämtliche Objekte z. B. Spezifikationen, Dokumentationen, Änderung von Anforderungen oder Quellcode
        - Ein Konfigurationsmanagementsystem muss auch in der Lage sein, Versionslinien, Änderungsstände und Freigaben der „Configuration-Items“ zu steuern.
        - Ablauf eines typischen Konfigurationsmanagementprozesses:
            1. Objekt wird eingerichtet.
            2. Objekt wird identifiziert, versioniert und archiviert.
            3. Objekt wird zur Prüfung freigegeben.
            4. Änderung wird eingereicht.
            5. Änderung wird veranlasst.
    - **Templates**:
        - Templates helfen, das Format, den Aufbau und den Inhalt von Dokumenten in Bezug auf Vollständigkeit und Qualität zu sichern.

        - Durch Vorgabe der Struktur und des Aufbaus von Dokumenten kann wertvolle Zeit für die Vorbereitung gespart werden.
        - Außerdem können Reviews effizienter und effektiver durchgeführt werden, wenn die Struktur der Dokumente bekannt ist.
        - Z. B. Testpläne, Testfallbeschreibungen, Testberichte, Review-Berichte
    - **Checklisten**:
        - Enthalten in Frageform gekleidete Richtlinien und Hypothesen von vermuteten Schwachstellen im zu prüfenden Produkt.
        - Diese Fragen sind so formuliert, dass sie mit ja oder nein beantwortet werden müssen.

- **Qualitätsmanagementstandards**:
    - Qualitätsmanagement beinhaltet aufeinander abgestimmte **Tätigkeiten** zum **Leiten** und **Lenken** einer **Organisation** bezüglich Qualität.
    - Sie sollen die Entwicklung eines Unternehmens hin zu einem **hohen Qualitätslevel** und **Kontinuität** unterstützen.
    - Standards verbessern zudem das **wechselseitige Verhältnis** und die **Koordination** unterschiedlicher Entwicklungsteams, speziell aber auch zwischen Entwicklung und Wartung.
    - Durch höhere **Transparenz** und **Berechenbarkeit** eines Qualitätssicherungssystems kann der Kunde von einem gewissen Qualitätsniveau ausgehen und bleibt so eher von unerwünschten Überraschungen verschont als bei intuitiver Vorgehensweise.
    - Grundsätzlich können Qualitätsmanagementstandards in **Zertifizierungs**- und **Assessmentstandards** (Bewertungsstandards) unterschieden werden.

- **CMM und CMMI**:
    - **Ziel**: Vergleichbarkeit und Qualität von Softwareprozessen im Rahmen der Vergabe an externe Lieferanten zu verbessern
    - CMMI liefert eine Analyse des Ist-Zustandes des Unternehmens und ist kein international anerkanntes Zertifikat.
    - Ein CMM-Assessment beurteilt die Fähigkeit einer Organisation, Software unter bestimmten Rahmenbedingungen erstellen zu können.



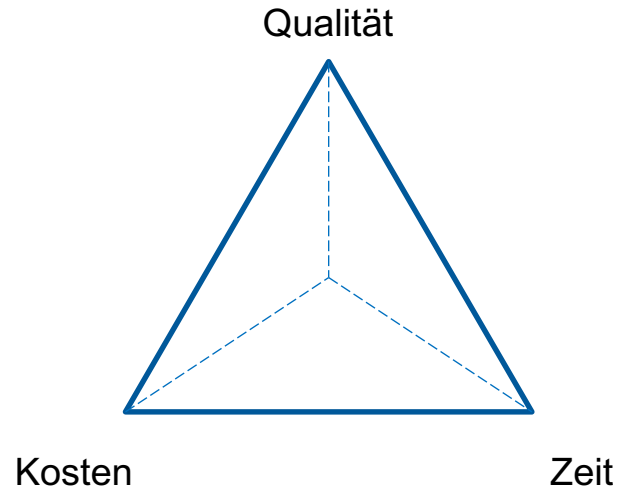
Projektmanagement

- Projektmanagement ist die Gesamtheit von Führungsaufgaben, Führungsorganisation, Führungstechniken und Führungsmitteln für die Abwicklung eines Projektes.
- **Projekt**:
  - **Klassifikation nach Projektarten**:
    - **Forschungsprojekte**:
      - **Ziel**: Entwicklungsziel nur grob definiert
      - Pioniercharakter, hoher Änderungsgrad der Projektparameter
    - **Entwicklungsprojekte**:
      - **Ziel**: Neues Produkt
      - Sehr zeitkritisch aufgrund hoher Konkurrenz
    - **Planungsprojekte**:
      - **Ziel**: Vorstudie und Konzept für anschließendes Großprojekt
      - Prüfung von Machbarkeit und Wirtschaftlichkeit
    - **Organisations-/Rationalisierungsprojekte**:
      - **Ziel**: Rationellere Abwicklung der Geschäftsprozesse
      - Neugestaltung der Aufbau- und/oder der Ablauforganisation
    - **Rollout-Projekte** (in Großunternehmen):
      - **Ziel**: Einführung einer Anwendung oder neuen Technik
      - Rollout erfolgt nach Pilotphase
    - **Investitionsprojekte**:
      - **Ziel**: Bau einer neuen Fertigungsstätte
      - ROI steht im Vordergrund
      - Qualität muss hoch sein
  - **Klassifikation nach Projektgröße**:

	Personal-aufwand	Finanz-aufwand	Dauer (in Monaten)	Anzahl Mitarbeiter	Eigenes Budget
kleines Projekt	≤ 0,5 PJ	15.000 bis 50.000 EUR	0,5 bis 3	1 bis 2	Nein
mittleres Projekt	≤ 5 PJ	≤ 250.000 EUR	≤ 6	≤ 12	Ja
großes Projekt	> 5 PJ	> 250.000 EUR	> 6	> 12	Ja

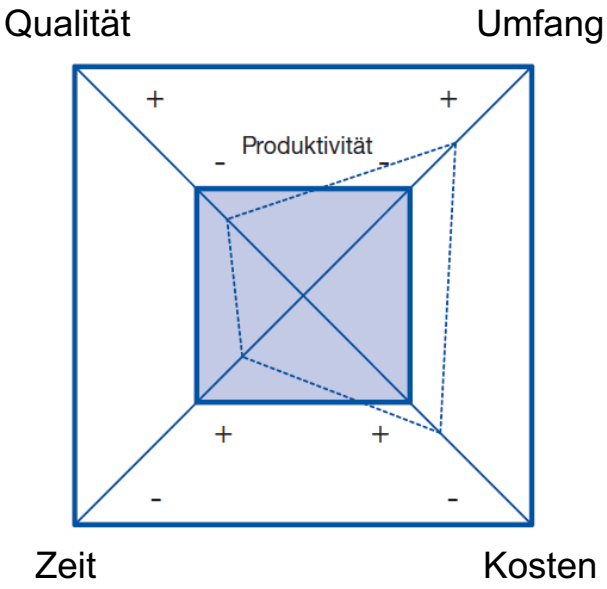
  - Klare Zuordnung, wenn mindestens drei Kategorien erfüllt sind.

- **Magisches Dreieck**:



  - Man kann nur zwei der drei Eigenschaften wählen:
    - Gut und billig geht nicht schnell.
    - Gut und schnell ist nicht billig.
    - Schnell und billig ist nicht gut.

- **Teufelsquadrat**:
  - Spannungsfeld von Qualität, Quantität/Umfang, Entwicklungsdauer/Zeit und Kosten
  - Messung und Skalierung der Werte von den Ecken zur Mitte
  - Die Fläche des Quadrates (= Produktivität) ändert sich nicht.
  - Erhöhung der Qualität und Senkung der Entwicklungsdauer funktioniert nur durch Reduktion des Umfangs und/oder Erhöhung der Entwicklungskosten.

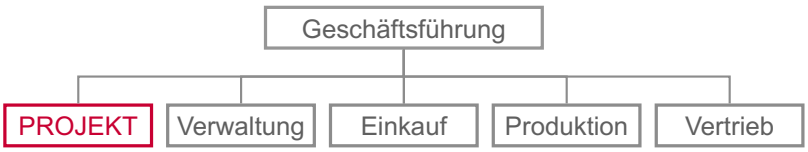


- **Hauptphasen eines Projektes**:
  1. Projektdefinition
  2. Projektplanung
  3. Projektkontrolle
  4. Projektabschluss
  - **Projektdefinition**:
    - Idee, Projektziel
    - **Vorarbeiten**:
      - **Machbarkeitsstudie**:
        - Was wird im Rahmen des Projektes getan?
        - Wie (unter welchen Umständen) wird es getan?
      - **Projektumfeldanalyse**:
        - Projekttyp
        - Funktionalitäten
        - Qualität
        - Projektunterstützende Faktoren
        - Mögliche Risiken
    - **Projektantrag**:
      - **Anforderungskatalog**:
        - Erste Planungsunterlage
        - Projektzeile erfassen (und Nichtziele)
      - **Pflichtenheft**:
        - Vereinbarungsgrundlage zwischen Auftraggeber und Auftragnehmer
        - Fachliches Grundkonzept und allgemeine Angaben
        - Erweiterung des Anforderungskataloges
      - **Leistungsbeschreibung**:
        - Fachliches Feinkonzept und technisches Grobkonzept
        - Legt Projektinhalt verbindlich fest
      - **Wirtschaftlichkeitsbetrachtung**:
        - Kosten und Nutzen als Entscheidungsgrundlage
    - **Projektauftrag**:
      - Schriftliche Vereinbarung zwischen Auftraggeber und Auftragnehmer
      - Dokument mit den wichtigsten Eckdaten als Zielvereinbarung

- Vertragscharakter
- Unterzeichnung von beiden Parteien
- „Geburtsschein“ des Projektes

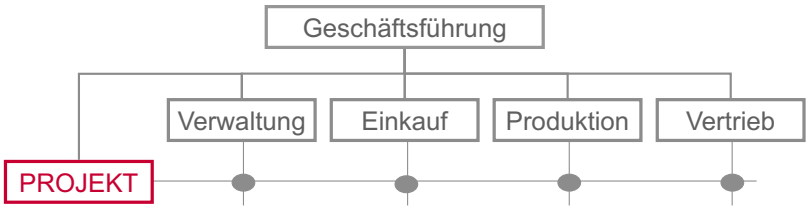
▪ **Projektorganisation**:

- **Reine Projektorganisation**: Projektleiter trägt Gesamtverantwortung und hat volle Weisungsbefugnis (Functional organization)



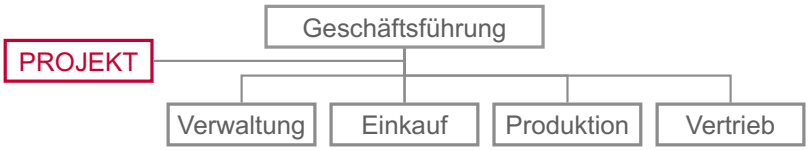
Vorteile	Nachteile
+ Verantwortung ist eindeutig geregelt + Starke Identifikation der Mitarbeiter mit dem Projekt + Hohe Motivation	- Schwierige Akquirierung der Projektmitarbeiter - Wiedereingliederung der Mitarbeiter am Ende des Projektes problematisch

- **Matrix-Projektorganisation**: Projektleiter trägt Gesamtverantwortung hat aber nicht die volle Weisungsbefugnis.



Vorteile	Nachteile
+ Projektverantwortung durch PL + Sicherheitsgefühl der Mitarbeiter + Flexibler Personaleinsatz	- Mitarbeiter ist Diener zweier Herren - Konfliktpotential „Linie/Projekt“ - Hoher Koordinationsaufwand

- **Einfluss-Projektorganisation**: Projektleiter ist nur Projektkoordinator und hat kaum Weisungsbefugnis.

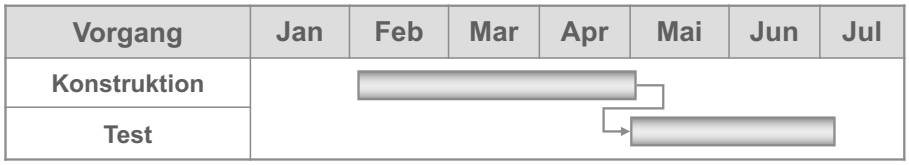


Vorteile	Nachteile
+ Geringe organisatorische Veränderungen + Hohe personelle Flexibilität + Kommunikation mit Fachabteilung bleibt erhalten	- Keine klare Verantwortlichkeit - Fehlende Autorität des PL - Hohes Konfliktpotential - Fehlender Teamgeist

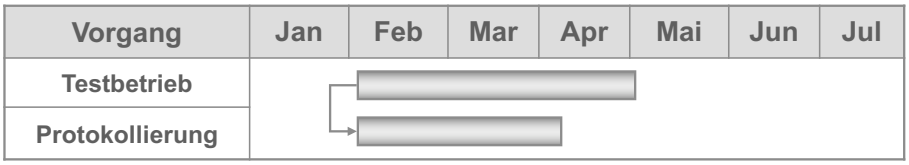
○ **Projektplanung**

- **Motivation**: WARUM wird das Projekt gemacht, was sind die generellen Zielsetzungen?
- **Leistung**: WAS muss gemacht werden, was sind die spezifischen Projektziele? WO wird am Projekt gearbeitet (Standort)? WELCHE Firmen bzw. Personen sind an der Durchführung des Projektes beteiligt?
- **Qualität**: WELCHE Qualitätsziele müssen erreicht werden? WIE soll vorgegangen werden?
- **Kosten**: WIEVIEL wird das Projekt kosten? WELCHE Mittel und Ressourcen werden für das Projekt benötigt?
- **Zeit**: WANN wird mit dem Projekt begonnen und wann muss es fertig sein?
- **Vorgehensweise**:
    1. Projektstrukturplan erarbeiten
    2. Arbeitsplanung und Arbeitspaketdefinitionen
    3. Aufwandsabschätzung
    4. Ablauf und Terminplanung
    5. Ressourcenplanung
    6. Kostenplanung
    7. Optimierung des Gesamtprojektes
- **Projektstrukturplan**:
    - **Ziele**:
        - Übersichtliche Gesamtdarstellung des Projektinhalts
        - Erkennen der Schwerpunktaufgaben
        - Systematische Erfassung aller Aktivitäten des Projektes
    - **Vorgehensweise**:
        - **Top-down**: Stufenweise Zerlegung der Gesamtaufgabe bis nur noch überschaubare Teilaufgaben vorliegen
        - **Bottom-up**:
            - Sammlung von Aufgaben (Brainstorming, Mind-Mapping)
            - Strukturierung der Aufgaben nach einem bestimmten Kriterium
            - Aufbau einer Aufgabenhierarchie (Clustern)
            - Ergänzung um fehlende Aufgaben
    - **Darstellungsform**:
        - Organigramm
        - Listenform
- **Arbeitspakete**: Kleines Einheiten des PSP
    - **Umfang** kann sehr unterschiedlich sein:
        - Bestimmte Tätigkeiten einer Einzelperson
        - Sachlich zusammengehörige Aktivitäten, die von einer Gruppe/Person durchgeführt werden sollen
        - Entwicklungsauftrag an eine Fremdfirma
    - **Kriterien für Arbeitspakete**:
        - Konkretes Ergebnis
        - Kalkulierbarkeit (Aufwand, Termin, Kosten)
        - Keine Überschneidung mit anderen Paketen
        - Klare Schnittstellen
        - Eindeutige Verantwortlichkeit
- **Aufwandsschätzung**: Expertenschätzung, Planning-Poker
- **Ablauf und Terminplanung**:
    - **Vorgangsbeziehungen**:
        - **Normalfolge: Ende-Anfangsbeziehung (EA)**

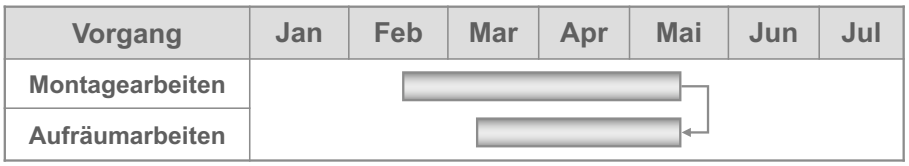
- Das Ende von Vorgang 1 ist Voraussetzung für den Beginn von Vorgang 2.
- Sequenzielle Abarbeitung



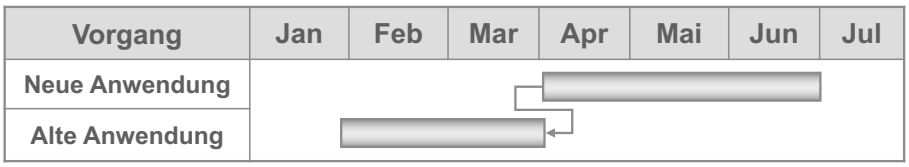
- **Anfangsfolge: Anfangs-Anfangs-Beziehung (AA)**
  - Der Anfang von Vorgang 1 ist Voraussetzung für den Anfang von Vorgang 2.
  - Parallele Abarbeitung



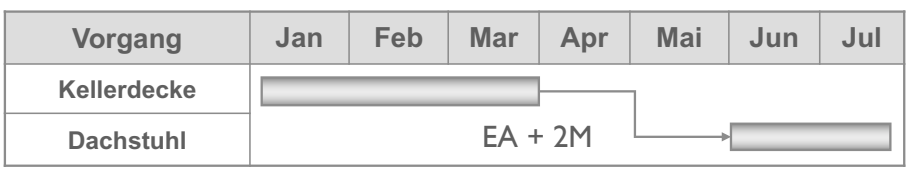
- **Endefolge: Ende-Ende-Beziehung (EE)**
  - Das Ende von Vorgang 1 ist Voraussetzung für das Ende von Vorgang 2.
  - Parallele Abarbeitung



- **Sprungfolge: Anfangs-Ende-Beziehung (AE)**
  - Der Anfang von Vorgang 1 ist Voraussetzung für das Ende von Vorgang 2.
  - Parallele oder sequenzielle Abarbeitung



- **Vorgangsbeziehung mit Überlappung oder Verzögerung:**
  - Durch Zeit- oder Prozentangabe definiert, wann der Nachfolger vor/nach dem Anfang/Ende des Vorgängers beginnen kann: ... ± Zeit oder ± Prozent



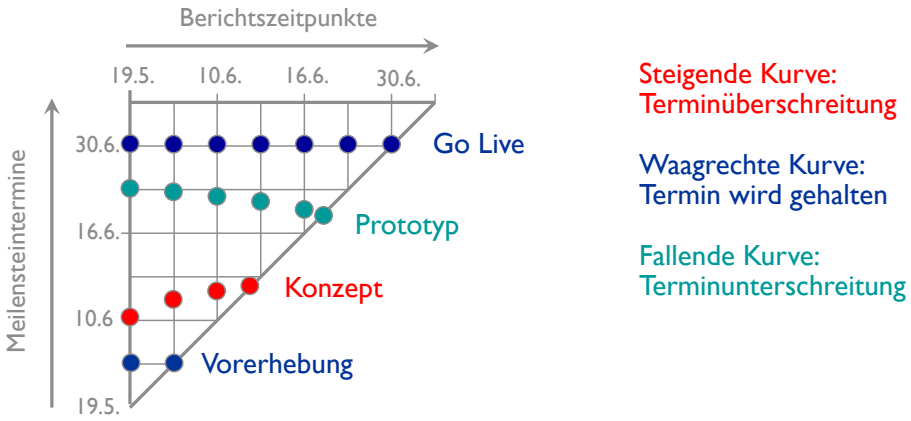
- **Terminberechnung**:

    - Vorwärtskalkulation
    - Rückwärtskalkulation
- **Meilensteine**:
    - Ohne Meilensteine kommt es zu starken Kurskorrekturen.
    - Mit Meilensteinen kommt es nur zu leichten Kurskorrekturen.
- **Balkendiagramm (Gantt-Diagramm)**:
    - Grafische Methode, Aufgaben zur Terminplanung darzustellen
    - Personen- oder Aufgabenbezogen
    - **Pufferzeit eines Vorgangs**:
        - **Gesamtpuffer**: Zeitspanne, um die sich ein Vorgang verzögern darf, ohne dass das Projektende verzögert wird
        - **Freier Puffer**: Zeitspanne, um die sich ein Vorgang verzögern darf, ohne dass ein andere Vorgänge verzögert werden
        - **Kritischer Vorgang**: Vorgang ohne Pufferzeit
        - **Kritischer Pfad**: Enthält nur Kritische Vorgänge
- **Netzplan**: Grafisch ausgelegtes Netz von Arbeitspaketen ohne Zeitachse
- **Ressourcenplanung**:
    - Bedarfsermittlung für jedes Arbeitspaket
    - Ermittlung der im Projektzeitraum verfügbaren Kapazitäten
    - Soll/Ist-Vergleich: (benötigte/vorhandene Ressourcen)
    - Kapazitätsabgleich bei Überlastung
- **Kostenplanung**:
    - **Ziel**: Transparente Darstellung der Projektkosten
    - Basis für Kostenkontrolle bei der Projektkontrolle
    - Projektkostenplanung und PSP sollten gleichen Aufbau haben
    - Kostenarten: Personal-, Material-, Betriebsmittel-, Kapital- und Finanzierungskosten, Kosten für Fremdleistungen
- **Projektplanoptimierung**:
    - **Vorgehensweise**:
        - Ersten Planentwurf wiederholt überarbeiten
        - Planungsalternativen erstellen
        - Wahl der optimalen Lösung
    - **Termine**: Konzentration auf kritischen Pfad
    - **Ressourcen**: Bessere Auslastung erzielen
    - **Kosten**: Teure Ressourcen durch billige ersetzen
    - **Qualität darf nicht außer Acht gelassen werden.**

- **Projektkontrolle**:
    - **Projektüberwachung**:
        - **Ziel**: Rechtzeitiges Erkennen von Abweichungen gegenüber den Plandaten
        - **Voraussetzungen**:
            - Vollständige, detaillierte und realistische Planung
            - Zeitnahe Kenntnis über aktuelle Ist-Daten
        - **Vorgehensweise**:
            - Berichtswesen (Ermittlung von Ist-Daten)

- Soll/Ist-Vergleich (Kontrolle der Trendanalyse)
- Analyse der Abweichungen (Ursachenanalyse und Bewertungen)

- **Überwachung durch Kontrolle**: Reaktion erst nach eingetretener Abweichung möglich; Steuerungsmaßnahmen bei eingetretenen Abweichungen einleiten
- **Überwachung durch Trendanalyse**: Reaktion schon vor tatsächlichem Eintreten der Abweichung möglich; Steuerungsmaßnahmen bei zu erwartenden Abweichungen einleiten
- **Meilenstein-Trend-Analyse**:



- **Projektsteuerung**:
  - **Korrektive Maßnahmen**: Ist an Soll
    - Motivation der Mitarbeiter verbessern
    - Neuverhandlungen führen
    - Kostenkontrolle verstärken
  - **Reaktive Maßnahmen**: Anpassung der Planung an die Projektsituation (Soll an Ist)
    - Verschiebung der Termine
    - Erhöhung des Budgets
    - Erweiterung des Projektteams
    - Reduktion des Projektumfangs
- **Projektkommunikation**:
  - Klare Kommunikation
  - **Ziel**: Einheitliches Besprechungs- und Präsentationswesen
  - **Inhalte**: Projektbesprechungen, Projektdokumentation, Prozessdokumentation, Produktdokumentation

- **Projektabschluss**:
  - **Ziel**: Erfolgreicher Abschluss des Projektes
  - **Formaler Projektabschluss**:
    - Abschluss der Dokumentation
    - Abschlusspräsentation
    - Projektabnahme durch den Auftraggeber (Abnahmeerklärung)
    - Freigrabe der Projektmitglieder und –ressourcen
  - **Nachbetrachtung der Projektarbeit (Review)**:
    - Abschlussanalyse
    - Erfahrungssicherung (Lessons Learned)
    - Abschlussbesprechung inkl. Social Event

- **Risikomanagement**: systematischer Prozess zur Identifizierung, Analyse, Behandlung und Kontrolle der Projektrisiken
    - **Risiko**: Unsicheres Ereignis, von dem nicht bekannt ist, ob es eintreten wird und in welcher Höhe es Schaden verursachen wird.

    - **Risikoidentifikation**: Liste von konkret formulierten Risiken
    - **Risikoanalyse**: Quantitative und qualitative Bewertung von Risiken
    - **Risikobehandlung und –kontrolle**: Erstellung eines Maßnahmenplanes zur Risikobehandlung

- **Faktor Mensch**:
    - **Teambildungsprozess**:
        - **Forming**: Team kommt zusammen. Aufgaben werden verteilt.
        - **Storming**: In der Startphase erfolgt die Bewährungsprobe. Normen und Ziele werden in Frage gestellt. Strukturen ändern sich, Aufgaben wechseln, Personen scheiden aus.
        - **Norming**: Team legt Regeln fest. Führung und Strukturen bilden sich heraus.
        - **Performing**: Team ist zusammengeschweißt und arbeitet effizient mit wenig Reibungsverlusten.

    - **Merkmale erfolgreicher Teams**:
        - Hoher Zusammenhalt
        - Engagement und Zielorientierung im Vordergrund
        - Ausgewogene fachliche und soziale Kompetenz
        - Unterstützung und Anerkennung von außen
        - Klare Rollen- und Aufgabenverteilung
        - Teamleiter ist nicht autoritär oder dominant
        - Jeder Beitrag wird aufgenommen und gewürdigt
        - Konflikte werden offen angesprochen und geklärt

- Erfolgsorientiertes Motivationssystem
- Personen mit Entscheidungskompetenz sind im Team

- **Rollen von Projektteammitgliedern**:
  - **Fachspezialisten**: Experten erster Wahl mit sehr guten Kenntnissen
  - **Teamarbeiter**: Kreativ, konstruktiv, kritisch, produktiv und kommunikationsfähig
  - **Botschafter**: Vertreter anderer Unternehmenseinheiten mit Doppelfunktion
  - Für ein erfolgreiches Projekt ist eine Ausgewogenheit an unterschiedlichen Mitarbeitercharakteren erforderlich.
- **Rollen des Projektleiters**:
  - **Architekt**: Schafft organisatorisches und technisches Umfeld
  - **Stratege**: Erstellt die Pläne für die zukünftige Vorgehensweise
  - **Führungskraft**: Fachliche Kompetenz und menschliche Autorität
  - **Controller**: Erkennt und reagiert auf Zielabweichungen
  - **Moderator**: Einbeziehung aller Mitglieder am Lösungsprozess
  - **Konfliktmanager**: Löst Konflikte der Teammitglieder
  - **Motivator**: Schafft motivierende Rahmenbedingungen
  - **Psychologe**: Ansprechpartner bei Unsicherheiten und Bedenken
  - **Sündenbock**: Verantwortlich für alle Fehlentwicklungen
- **Führung von Projektteams**:
  - Führungsstil ist ein zeitlich überdauerndes, für bestimmte Situationen konsistentes Führungsverhalten von Vorgesetzten gegenüber Mitarbeitern
  - Dimensionen zur Beschreibung des Führungsstils:
    - Ausmaß der Teilnahme am Entscheidungsprozess
    - Aufgabenorientierung vs. Personenorientierung
  - **Führungsstile**:
    - Autokratischer Führungsstil
    - Kooperativer Führungsstil
    - Demokratischer Führungsstil
    - Situativer Führungsstil
    - Authentischer Führungsstil
    - **Autokratischer Führungsstil**:
      - Projektleiter entscheidet über Inhalt und Prozess
      - Wenig Kritikbereitschaft
      - Genaue Einzelanweisungen und Kontrolle
      - Unternehmerisches Denken und Handeln wird nicht gefördert
      - Keine kreativen Einbringungsmöglichkeiten
      - Gefühl von Sicherheit bei unerfahrenen Mitarbeitern
      - Zielführend in Krisen durch rasche und fokussierte Entscheidungsfindung
      - Bindung von Mitarbeitern, die keine Verantwortung übernehmen möchten und Anweisungen korrekt ausführen wollen
    - **Kooperativer Führungsstil**:
      - Beteiligung an der Zielfestsetzung und Prozessgestaltung
      - Delegation von Aufgaben, Befugnissen und Verantwortung
      - Transparenz bei Entscheidungen und Maßnahmen
      - Ergebniskontrolle als Ergänzung zur Selbstkontrolle
      - Starke Betonung der Eigenverantwortlichkeit
      - „Teamorientierte Führung“

  - Verlangt vom Projektleiter die notwendigen Kenntnisse über geeignete Techniken der Intervention
  - Gruppenphänomene müssen vom Projektleiter durch Zuhören und Beobachten erkannt werden.
- **Demokratischer Führungsstil**:
  - Inhalt und Prozess werden durch Gruppendiskussion und Gruppenentscheidung beschlossen.
  - Persönliche und fachliche Präferenzen werden bei der Zusammenarbeit im Team berücksichtigt.
  - Projektleiter schlägt Entscheidungen vor, entscheidet aber nicht selbst.
  - Hohe Gruppenkohäsion und Interaktion stellen sich ein, starke Gruppenmoral ist die Folge.
  - Qualität und Originalität von Leistungen sind hoch, Geschwindigkeit und Effizienz von Entscheidungen sind oft unbefriedigend.
  - Sonderform: Kollegiale Führung
- **Situativer Führungsstil**:
  - Projektleiter setzt in unterschiedlichen Situationen verschiedene Führungsstile ein.
  - Z. B. GRID-Führungsmodell mit zwei Ausrichtungen: Orientierung an den Menschen (Mitarbeiter, Vorgesetzte, ...) und Orientierung an der Sache (Projektziel, Aufgaben, ...)
  - Grundhypothese ist, dass ein wechselnder Führungsstil sinnvoller ist als immer der gleiche Führungsstil.
  - Die Wahl des Führungsstils ist jedoch subjektiv beeinflusst und lässt sich nicht sehr leicht standardisieren.
  - Besonders gut für heterogene Projektstrukturen geeignet
- **Authentischer Führungsstil**:
  - Unterscheidet sich wesentlich von den üblichen Modellen
  - In Projekten sehr erfolgreich
  - Konzentriert sich nicht auf Eigenschaften und Verhaltensweisen einer idealen Führungskraft
  - Individuelle Persönlichkeit steht im Vordergrund
  - Grundwerde einer authentischen Persönlichkeit:
    - Akzeptanz und Respekt
    - Vertrauen und Toleranz
    - Offenheit für Neues
  - Führungskräfte sind dann erfolgreich, wenn sie in Übereinstimmung mit dem eigenen Wertesystem handeln