Anforderungsbestimmung:

- Aufgaben: Problembeschreibung, Vertrag zw. Benutzer und Entwickler, Kommunikation, Veränderungen am System und Validierung des Entwurfs verwalten
- Verfikation zw. Problem und Implementierung (System richtig gebaut gemäß Spezifikation) ↔ Validierung zw. System und Realität (richtiges System? - alle Domäneneigenschaften erkannt?)
- Anwendungsdomäne: Domäneneigenschaften (unabhängig vom System), Anforderungen (mit System)
- Spezifikation: Verhalten des Programms, um Anforderungen gerecht zu werden
- Maschinendomäne: Computer und Programm
- System: Teil der beobachtbaren Realität in ihrer Interaktion mit der Umwelt, Systemgrenzen, emergente Eigenschaften, nicht nur Software, sondern auch Benutzer und andere Umweltfaktoren
- Schwierigkeiten bei der Erhebung: Unklare Ausdrucksweise der Experten, keine Fantasie, verschiedene Gruppen, Verständnisproblem zw. Technik und Fach, Anforderungformulierung unverständlich, Änderungen, stillschweigendes Wissen, limitierte Beobachtkeit, Voreingenommenheit, Beeinflussbarkeit
- Erhebungstechniken:
  tradionelle Ansätze: Introspektion (hinsetzen und Anforderungen überlegen)
  - bestehende Dokumente und Daten: **-** veraltet, vereingenommen, **+** schnell, billig, oft detaillierte Infos
  - Interviews: **+** viele Infos **–** große Datenmengen, gewisse Fähigkeit, stillschweigendes Wissen
  - Fragebögen: **+** viele Daten **-** zu simple Kategorien, Missveständliche Fragen, Testen der Fragebögen nötig
  - Erhebung in Gruppe: **+** natürliche Interaktion, Diskussion **–** Groupthink, Dominaz, oberflächlich, Leiter!

  Repräsentationsbasierte Ansätze:
  - zielbasiert: **+**gesunde Basis für Konflikteinigung **–** Detailmangel
  - Szenariobasiert: **+** sehr natürlich **–** keine Struktur, beser Use Cases oder Aufgabenmodelle
  - Use Cases: **+** einfach verständlich, Systemgrenzen findbar **–** nichtfunktionale Anf., kein Domänenwissen

  Kontextbaiserte soziale Ansätze: ethnografisch: Beobachter in Domäne: **–** aufwendig, keine Verbeserung bestender Verhältnisse
  Wissensbasierte kognitive Ansäzte: Protokollanalyse, Proximity Scaling, Card Sorting
- Anforderungsvalidierung: Überprüfung auf Korrektheit, Komplettheit, Konsistenz, Realitätsnähe, Zurückverfolgung
- Methodische Ansätze: Erhebung bei möglichst vielen Beteiligten, Beschreibung allgemein verständlicher Form, Validation gegen Missverständnisse, Spezifikation (Überführung in eine Form, die für weitere Analyse und technische Realisierung besser geeignet ist), Trennung von Belangen (möglichst wenig Koppelung → Verständnis, Unabhängigkeit, Wiederverwendung), Analyse auf Vollständigkeit und Konsistenz, Mediation (Interessensgesätze auflösen) Verwaltung (aktuell gehaltenes Anforderungsdokument, Änderungen prüfen)
- Anforderungsmanagement: Anforderungen können sich ändern → Tools

Anwendungsfälle (Use Cases):

- Beschreibung der funktionalen Anforderungen einer Software
- bei Inkonsistenzen beim Kunden nachfragen, vorher über Alternativen nachdenken → richtige Fragen stellen
- Kunde kauft, Akteur interagiert mit System, Stakeholder hat irg. Interesse oder Beziehung zum System
- Ablauf: Akteure identifizieren (auch außenstehende technische Geräte können Akteure sein), [Systemgrenzen festlegen,] Anwendungsfälle (siehe funktionale Anforderungen), Anwendungsfalldiagramm, [Glossar,] Ereignissfolge (Voraussetzung, 1. 2. 3., Zusicherung), Erweiterungen (Alternative, Fehlerfälle: 2a, 3a, 3b)
- Syntax: Anwendungsfall: Hauptakteur: Voraussetzung:, 1, 2, 2a, 3, 3a, 3b, Zusicherung:
- Fehler: Systemkompente ist kein Akteur, denn Akteure sind immer außerhalb des Systems
- für Benutzer und Anforderungserhebung hilfreich, für Entwickler ungeeignet, da keine Beschreibung von Funktionen oder Daten
- nur ein Teil der Anforderungsdokumente **-** beschreiben keine globalen Festlegungen, nichtfunktionalen Anforderungen, konkret gewünschten Funktionen, Domäneneigenschaften und Annahmen, Randbedingungen (technisch, rechtlich)

Analyse:

- Gewinnung von Klassen aus Use-Cases (meist Beteiligung mehrerer Use Cases zu einer Klasse)
- Statische Objektmodellierung: identifzieren von Klassen, Attributen, Methoden und Assoziationen
- Objekttypen:
  - Geschäftsobjekte: Daten-beinhaltende Klassen, meist Abbildungen realer Entitäten (z.B. eRezept)
  - Zugangsklasen: mit ihnen bekommen Akteure Zugang zum System, nicht nur Klassen aus der UI, sondern auch Schnittstellenklassen, die den Zugriff auf die interessanten Geschäftsobjekte ermöglichen.
  - Steuerklassen: durch dynamischen Anteil wichtig: sorgen für den zentralen Ablauf einer Operation. Im Sequenzdiagramm gehen die meisten Pfeilen von Ihnen aus, sie sind meist im System verborgen
  - flexibleres Modell und elastisch für Veränderungen eines Objekttyps
- Bereichsklassen: Repräsentation der Konzepte des Anwendungsbereichs, Geschäftsobjekte und Beziehungen, keine exakten Methodensignature (Analysemodell)
- Lösungsklassen: Repräsentation der Umsetzung auf technischer Ebene (weglassen, neue hinzu, verschmelzen)
- Vorgehensweise: Anforderungserhebung zu Analysemodelle (Erstellen von Szenarien und Ableiten von Klassen daraus)
  - stark Objekt- und Klassenbasiert → starke Repräsentation von Datenrepräsentationen und der Interaktionen zw. Beinhaltenden Objekten und Daten, führt meist zu datenflussorientierten, Objektnetz-ähnlichen oder ablagebasierten Architektur. Dies sind typischerweise Informationsysteme.
  - Weniger typisch für Realtimesystem (hoher Kontrollflussanteil) oder embedded Systeme

Dynamisches Modell:

- Ziel: Verständnis von Verhaltensanforderung und Identifzierung von Methoden
- Sequenzdiagramme: modellieren Interaktionen über Objekte als Ereignisablauf, guter Ablauf von links nach rechts: Akteur, Zugangsklassen, Steuerklassen, Geschäftsobjekte, Herleitung aus Use Cases
- Sequenzdiagramme sind gut geeignet, um den Haupterfolgsfall darzustellen (Erweiterungen und Alternativen dagegen schwierig). Dann wäre Aktivitätsdiagramm sinnvoller. Sequenzdiagramm sind ein gutes Mittel systematisch von der Erhebung zur Analyse zu gelangen, weil man sich die Klassen der Objekte überlegen muss und aufschlüsseln muss was das System eigentlich ist.

- Zustandsdiagram (Statechart): stellt Ereignisse und Zustände einer Klasse in Beziehung, Modellierung von Parallelität
- Schritte der Anforderungsanalyse: Problem identifizieren → funktionales Modell (Use Cases) → dynamisches Modell (Sequenz und Statechart) → Objektmodell (Klassendiagramme für Struktur des Systems)

Software-Architektur: Wie findet man eine Gesamtstruktur mit den gewünschten globalen Eigenschaften?

- Softwarearchitektur: Module deren Schnittstellen und die Art ihrer Beziehungen zueinander, Entscheidungen zur Bildung der Struktur zur Erfüllung von funktionalen und nichtfunktionalen Anforderungen
- Schichtenarchitektur: mehrere Abstraktionsschichten, universell nützlich, Austausch ganzer Schichten möglich, sehr übersichtliche und aufgeräumte Grobstruktur, keine unnötigen Kopplungen, portabel, ineffizient, unnatürlich/umständlich Bsp.: Linux-Kernel, KVV
- Datenflussnetze (Pipes-and-Filters): verknüpfe Verarbeitungsschritte mit Datenfluss, Übersetzerwerkzeugkette, Reportgenerator, einfach und klar, unflexibel
- Objektnetze mit Datenkapseln: Objekte und Beziehungen komplett problemabhängig gestalten, sehr flexibel, verworren, jedes Objektorientierte System, geringer Speicherbedarf (optimierter Datenzugriff)
- Ereignissteuerung mit impliziten Aufrufen: Verarbeitungseinheiten registrieren sich an zentraler Stelle und werden bei Ereignissen benachrichtigt, dezentralisiert und flexibel, Verhalten evtl. schwer durchschaubar, alle GUIs
- Ablagebasiert (repository, blackboard): Kommunikation über gemeinsame Datenablage, effizienter Datenaustausch, lose Kopplung, unflexible, Schema-änderung nur schwer möglich, Datenbankanwendungen, Versionsverwaltung
- Interpretierer oder Regelsysteme: Zustandsänderungen durch Regeln, erfordert Ermittlung geeigneter Regeln, Expertensysteme, Tabellenkalkulation erlaubt Formeln, die interpretiert werden
- Klient-Server-Architektur: verteilte Architektur mit Client und Server: alle Webanwendungen
- Unterbrechungsorientierte Architektur: Benachrichtigen über Unterbrechungen statt über Nachrichten oder Aufrufe, Betriebssystem, Echtzeitssysteme

Modularisierung: Systemzerlegung in passende Teile: Module / Komponenten bzw. Subsysteme

- Module: konkret und ausführbare Programmteile, Inhaltsteile eines Moduls sollten viel miteinander zu tun haben (Kohäsion), aber statisch wenig mit den übrigen Modulen (lose Koppelung), funktionales Verhalten allein durch Schnittstellen beschrieben, Verbergen von Details der Realisierung (Geheimnisprinzip)
- Schnittstellen: Beschreibung eins Moduls zum Rest des Systems, Gesamtheit aller relevanten extern sichtbaren Eigenschaften (Funktionen, Semantik, Voraussetzungen, Effekte), enthalten alle Klassen, Interfaces, die für den Nutzer wichtig sind, und ihre Verwendungsdoku, sie verbergen (sich öfter ändernde) Entwurfsentscheidungen, Verträge: Invariante, Vorbedingung, Nachbedingung, Object Constraint Language

vom Analysemodell zum Entwurfsmodell: Analysiere Klassen aus Problembereich, Grobentwurf mittels Architektur und Entwurfsentscheidungen, Modularisierung, zerlege Module in Teilmodule und schließlich in Lösungsbereichklassen

Qualitätssicherung: Gesamtheit aller Maßnahmen, die auf die Herstellung eines Produkt in guter Qualität zielen

Analytische: Prüfend, untersuche (Teil-)Produkte nach Fertigstellung, bessere Mängel nach

- Dynamische Verfahren:

  Defekttest: Versagen herbeiführen (falsches Verhalten des Programms im Sinne der Spezifikation, Anforderung oder Erwartung)

  Wie wählt man Testfälle = (Systemzustand vor dem Test, Eingaben, Systemverhalten)?

  - Funktionstest (Black Box): Testfälle durch Betrachtung der Spezifikation der Komponente, Äquivalenzklassen, Fehlerfälle, Randfälle (pro Äquivalenzklasse)
  - Strukturtest (White Box): Testfälle durch Betrachtung der Implementation, Anweisungsüberdeckung, Bedingungsüberdeckung, Schleifeüberdeckung (0-, 1-, mehrfach), mechanisch entwickelbar
  - Bekannte Versagensfälle
  - Allgemeine Erfahrung, Intuition: leere, riesige, unsinnige Eingaben

  Wer wählt Zustände und Eingaben aus?

  - Jemand Geschultes, möglichst nicht Entwickler des Codes (vllt Annahmen falsch aufgefasst), Alternative: Test-First-Entwicklung

  Wie wählt man Testgegenstände aus?

  - Mit wenigen Teilen anfangen, Bottom-Up (erst Teile, dann größere Komponenten, schließlich Hauptprogramm), Top-Down (erst Hauptprogramm, simuliere fehlende Teile durch Stubs), Opportunistisches Vorgehen: Test anhand von Prioriäten (frühe, intensive Fälle)

  Wann/wie kann und sollte man Tests automatisieren?

  - Testframeworks (Junit), Rückfalltesten (Regressionstesten, automatisiert alles außer Testeingaben), Prüfung von Zusicherungen, Automatisierung bei Lasttest, Leistungstest und Stresstest, Testwerkzeuge, Quick-and-Dirty für Evaluation von Fremdsoftware

  Wann sollte man mit dem Testen aufhören?

  - Kosten übersteigt Nutzen, Zeitplan erschöpft, neues Versagen seltener, alle Testfälle bestanden

  Benutzbarkeitstest: Beobachtung von echten Nutzern (iterativ)

  Lasttest: strapaziöse Eingaben zum Testen der Antwortzeiten (Leistungstest), Verhalten bei viel Last (an die Grenze des Erlaubten → Lasttest) und Verhalten bei Überlast (außerhalb des Erlaubten → Stresstest) → Werkzeuge

  Akzeptanztest(Black Box): demonstriert dem Kunden, dass das Produkt tauglich ist, aus Use Cases hergeleitet

  => Schwäche aller Testverfahren: erkennt nur Versagen, fehlt Defektlokalisierung (Debugging), sehr aufwendig, statische + konstruktive Verfahren günstiger
- Statische Verfahren:

  Manuelle Verfahren:

  - Durchsicht: Artefakte (Dokumente / Code) werden vom Menschen gelesen, um Mängel aufzudecken, Mängel frühzeitig aufgedeckt + Kommunikation und Ausbildung
  - Selbst-Begutachtung: Autor sieht sein Produkt allein durch, effektiv, einstellungsabhängig
  - Peer-Review: Durchsicht durch jemand anderen, halbformell
  - Fagan-Inspektion: Durchsichten im aufwendigen Teamprozess mit Planung
  - Lesetechniken: Perspective-based-Reading: jeder verwendet andere Perspektive zur Analyse
  - Empirische Resultate: Codedurchsichten finden mehr Defekte pro Stunde als Test und auch andere Defekte und sie haben Lerneffekt und wirken sich auf weitere Durchsichten aus
  - Vorteile von Durchsichten: auch auf nicht ausführbaren Dokumenten, Lokalisierung, zusätzliche Mängel (Kodierrichtlinien), Hinterfragen des korrekten Verständnisses der Anforderungen, Lernen von gelesenem Codec

  Automatische Verfahren: Modellprüfung (Prüfung von Sicherheitseigenschaften), Quelltextanalyse

Konstruktive: vorbeugend, gestalte Konstruktionsprozess und Umfeld so, dass Qualitätsmängel seltener werden, beseitige Mängel und auch deren Ursachen, beginne vor Entwicklungsarbeit, lerne aus Projekterfahrungen, korrigiere Prozess auch unterwegs, strukturiere den Arbeitsprozess vor und überwache den laufenden Prozess

- Projektmanagement auf ein einzelne Produkt bezogen
- Prozessmanegement zielt auf Verbesserung des Prozesse einer Organisation über alle Projekte hinweg:
    - charakterisiere Projektumgebung, setze quantifizierbare Ziele, wähle geeignete Vorgehensweise aus Erfahrungen für Projekt aus, analysiere Ergebnis, ergänze Erfahrungsdatenbank, Lernprozess
    - Prozessreifemodelle:
        - Capability Maturity Model for Software (5 Reifestufen: Initial, Repeatable, Defined, Managed, Optimized; ab Stufe 3 ist z.B. Umfang der Testphase abhängig von aktuellen Testmaßen. Richtig gesteuert wird der Prozess schon ab Stufe 4),
        - Total Quality Management (Qualität steht hat oberste Priorität, dies gilt auch für jede Tätigkeit jedes einzelnen Mitarbeiters. Ziel ist es den Kunden (Kollege) zufrieden zu stellen)
- Schutz (safety, Schutz der Umwelt vor dem System) und Sicherheit (security, Schutz des Systems vor der Umwelt ebenfalls wichtige Qualtitäsmerksmale

Prozessmodelle:

- Gesamtvorgehensweise nicht für jedes Projekt neu erfinden, sondern auf vorhandene Erfahrungen abstützen und normales Vorgehen praktizieren (Planung, Koordination, Korrekturen, Iterationen)
- Software-Prozessmodell: Schablone, die die Geymeinsamkeiten der Abläufe in vielen verschiedenen Projekten zusammenfasst
- Aktivität: Abstraktion für zielgerichtetes Handeln in einem Projekt, Steuerung der Aktivitäten ist Element des Prozessmodells (Ablaufplan, Verteilung, ...)
- Artefakt: Abstraktion für das Arbeitsergebnis einer Aktivität (Klassendiagramm, Testfall)
- Wasserfallmodell:
    - kleine Anzahl verschiedener Aktivitäten (Planung, Anforderungsbestimmung, Architekturentwurf, Feinentwurf, Implementierung, Integration, Validierung, Inbetriebnahme, Pflege), die nacheinander durchlaufen werden (Vorbild Hausbau)
    - dokument-getriebener Prozess mit gründlicher Prüfung und Weitergabe ohne Kommunikation
    - gut planbar da feste Anforderungen, Systemzerlegung und Aufwandsschätzung (→ Festpreis)
    - Schwächen bei unklaren Anforderungen, veränderlichen Anforderungen, nicht beherrschten Architekturen und kaum Kommunikation („über die Mauer werfen“)
    - gut anwendbar, wenn eine enge Koordination mit Technik (hardwarenah) nötig ist (also Planbarkeit). Bei kleinen, kritischen Systemen wie ABS können die Anforderungen klar definiert werden => kaum Nachteile des Wasserfallmodells.
- Iterationen:
    - moderne Prozessmodelle empfehlen iteratives Vorgehen
    - Vorteil: senkt Komplexität einzelner Schritte, besserer Umgang mit unklaren oder veränderlichen Anforderungen, verlangt engere Kommunikation der Beteiligten
    - Nachteil: gewisse Doppelarbeit, theoretisch höherer Aufwenad, verlangt engere Kommunikation
    - Prototypmodell: baue wegwerfbares Hilfssystem, um kritische Anforderungen besser zu verstehen
    - Inkrementelles Modell: baue Gesamtsystem schrittweise, falls nötig Änderungen existierender Teile
    - Spiralmodell (Risikomodell): Tue in jeder Iteration das, was am stärksten zur Verringerung des kritischen Projektrisikos beiträgt
    - Evolutionäres Vorgehen z.B. bei unserem Prüfungsverwaltungssystem, da Anforderungen sich mit der Zeit ändern können, insbesondere Funktionsumfang der Software steigt.
    - Inkrementelles Vorgehen z.B. bei Selbstbedienungskonsole für Bahnhöfe, da die Bedienbarkeit ein hohes Risiko ist. Also wird man erstmal die Benutzungsoberfläche bauen und sie ausführlich testen.
- Allgemeine Ziele statt konkreter Pläne:
    - Agile Prozesse: verlangen sehr viel Kommunikation und Disziplin, dafür sehr flexibel
    - bei agilen Prozessen verhindert das grundsätzlich iterative Vorgehen (Zwischenversionen) und die ständige Rückmeldung beim Kunden, dass man am Ende ohne lauffähige Software dasteht.
    - strikte, stark planende Modelle, wenn wohldefiniertes Resultat in definierter Zeit erreicht werden muss; wenn große verteilte Projektgruppen koordiniert werden müsse; bei paralleler Entwicklung von Hardware und Software
    - Agile Methoden, wenn hohe Unsicherheit über Anforderungen besteht, wenn Änderungen von außen häufig, wenn es kein Zeitplan gibt, wenn mit unausgereifter, unbeherrschter Technologie gearbeitet wird (radikales Vorgehen)
- Universale Prozessmodelle:
    - Rational Unified Process (Dynamische Aspekte und Statische Aspekte): ist er wasserfallentartet, so sind die Phasen gleich den Workflows (=Aktivitäten) ohen Iteration.
    - V-Modell XT: zu jeder Entwicklunsaktivität gehört eine Validierungsaktivität
- eXtreme Programming: grundsätzlich iterativ, geeignet für vage und ändernde Anforderungen (da agiler Prozess), Satz von Praktiken, die genau eingehalten werden müssen (hohe Disziplin), verstärken sich gegenseitig, short releases, Kunde vor Ort, Pair programming, Test First, Unit Testing (Modultests)

Projektmanagement: Gesamtheit von Führungsaufgaben, -organisation, -techniken und -mittel zur Projektabwicklung

- Balanzierung von (Funktions)Umfang, Qualtität, Projektdauer / Kosten
- Entwicklung eines umfassenden Projektplans: Dokumente über Vereinbarungen mit Auftraggeber, produktorientierte, prozessorientierte technische Dokumente und prozessorientierte Organisationsdokumente
- Pflichtenheft (WAS für Auftraggeber) und Projektplan (WIE für Projektteam)
- Projektleitung und -überwachung:
    - Nichtlineare Dynamik: Rückkopplung von Effekten kann sich aufschaukeln (verstärkend oder abschwächend), Brooks' Gesetz: Adding people to a late project make it later
    - Teufelskreis zw. Qualität und Zeitdruck, meist ausgelöst durch Anforderungsänderungen
    - Kommunikation: geplant um zuverlässig alle Betroffenen zu erreichen, ungeplant um Lücken zu füllen (schriftlich vs. mündlich, synchron vs. asynchron, Besprechungen)

- Umfangsmanagement: Umfangsdefinition (Anforderungsbestimmung), Produktzerlegung (Work Breakdown Structure, Entwurf, Prozessmodell), Umfangsverwaltung (Anforderungsverwaltung)
- Zeitmanagement: Aktivitätsliste, Aufwandsschätzung, Aufstellen eines Zeit- und Arbeitsplans, Zeitplanüberwachung
- Schätzung des Gesamtaufwands: nur für normales Vorgehen und selbst da schwierig (hohe Komplexität): Schätzen durch Vergleich, Zerlegung (von Anforderungen oder Entwurf) + Aufsummierung, separate Expertenschätzungen, kombiniere Schätzungen zu einer, Schätzen mit Korrekturverfahren, Schätzen mit Stellvertretergröße, Funktionspunktverfahren (zähle und klassifiziere wenige Arten von Anforderungen)
- Todesmarsch-Projekte: zentraler Ressourcen-Parameter um mind. Faktor 2 unter dem Normalen
- Aufstellen eines Zeit- und Arbeitsplans: Projektmanagement-Software
  Workbreakdown-Structure:
    - Top-Down: Identifiziere nötige Schritte und breche sie in feinere Schritter herunter
    - Brainstorming: trage Aufgaben zusammen und ordne sie anschließend in Gruppen
- Kostenmanagement: Kostenschätzung, Budgetaufteilung, Kostenüberwachung (meist Personalkosten)
- Qualitätsmanagement: Qualitätsplanung, Qualtitässicherung, Qualitätssteuerung
- Personalmangement: Personalplanung, Team einwerben und Teamformung, Teamführung
  Linear Responsibility chart: P (primary), S, A (approval), R (review), O (Output), I (Input)
- Kommunikationsmanagement: Kommunikationsplanung, Informationsverteilung, Fortschrittsberichte, Interaktion mit Beteiligten
- Risikomanagement: Risiken identifzieren, Risikoeinschätzung, Vorbeugung planen, Gegenmaßnahmen, Überwachung (Risikohöhe = Eintrittswahrscheinlich * Schadenshöhe)
- Beschaffunsmanagement: Planung und Durchführung von Beschaffung, meist Möbel, Hardware, Lizenzen
- Prinzipien:
    - Zielsetzung: Sorge dafür, dass die Ziele des Projekts + Prioritäten alle Beteiligten klar sind und sie sich damit identifizieren.
    - Stabile Anforderungen: möglichst wenig Veränderungen im Laufe des Projekts
    - Iteration: wohldefinierte Ergebnisse (Meilensteine) in kurzen Abständen, idealerweise einsetzbare Versionen des Endprodukts
    - Planung und Koordination: Alle Beteiligten sollen konkrete Aufgabe haben, die sie kennen. Es gibt ein sinnvolles zeitliches Ziel für die Erledigung und die Einhaltung der Zeitplaung wird auch projektweit überwacht und sichtbar gemacht
    - Kommunikton: alle im Projektverlauf benötigten Informationen sollen die Personen auch zu geeigneter Zeit erreichen
    - Konflikt: sorge dafür, dass entstehende Konflikte auf zufrieden stellende und zielführende Weise gelöst werden
    - Risikomanagement: identifiziere alle plausiblen unerwünschten Tendenzen und Ereignisse (Risiken), die den Projekterfolg erheblich bedrohen. Entwickele für jedes Risiko entweder einen Kontigenzplan für den Fall des Eintretens oder leite vorbeugende Maßnahmen ein, die das Eintreten unwahrscheinlicher machen. Überprüfe Risiken, Pläne und Maßnahmen regelmäßig wegen Änderungen.
    - Normales vs. radikales Vorgehen: Halte dich bei allen Aspekten des Projekts (Anforderungen, Entwurf, Vorgehensweise, Technologie) wo immer möglich an Ansätze, für die das Projektteam bereits ausreichende Erfahrungen beistzt und vermeide es, den Bereich solcher Erfahrungen zu verlassen, so lange nicht der davon erwartete Nutzen die Risiken überwiegt.

Wiederverwendung:

- fast alles wiederverwendbar, wäge Gewinn und Verlust ab, Normales Vorgehen
- Ziel: Aufwandsverminderung jetzt / später, Qualitätsverbesserung, Risikovermeidung, Standisierung
- Risiken: ungewisse Qualität, Eignung fraglich, eingeschränkte Flexibilität, Lieferantenabhängigkeit
- Hindernisse: „not invented here“, Ignoranz, Faulheit, Auswahl- und Zusatzaufwand, Flexibilität eingeschränkt
- Analysemuster, Benutzbarkeitsmuster (Benutzereffektivität erhöhen und Wirkung von Systemfehler reduzieren)
- Prozessmuster: Baue Prototypen (senkt Risiko), Liefertermin festlegen (klare Orientierung und Motivation), Beobachte Liefertermin-Puffer (ständig), Keine kleinen Terminverzögerungen, Gemeinsame Zustimmung zum neuen Plan, Unterbrechungen (Kindertagsstätte für Neue, Söldner für Doku), Programmieren in Episoden (höhere Qualität)
- Anti-Muster: Analyse-Paralyse (Tod durch Planen), Rad wiedererfinden, Goldener Hammer (bewährtes Konzept übermäßig intensiv benutzen), Minenfeld (bleeding edge), Dumm halten (kein Kontakt zu Benutzern), Pseudo-Experten-Herrschaft (Entscheidungen durch Presseberichte), Todesmarsch, Email-Glaube, Entwurf per Komitee (zu viele Köche...), Conway Gesetz (Architektur bildet Organisationsstruktur ab), Krake (Klasse mit Beteilung überall → zu viel Kopplung)

Dokumentation:

- sowohl des entstandenen Entwurfs selbst (Modullschnittstellenbeschr.), also auch alle Entwurfsentscheidungen + Begründung (für Veränderungen)
- Bei Entwurfsentscheidungen: verfolgte Zweck, Alternativen, und Warum diese Alternative
- Mehrwert: keine offensichtlichen Informationen
- Nützlichkeit: hilfreiche Informationen
- Lokalität: möglichst eng mit dem Dokumentierten
- Integration: möglichst zeitnah zur Entstehung des Dokumentierten
- Redundanzarmut: eine Information möglichst nur an einer Stelle
- Nachvollziehbarkeit: Überlegungen des Erbauers sollen nachvollziehbar sein
- Selbstdokumentation: Software mit so klaren Strukturen, dass sie auch ohne Doku veständlich ist
- Minimaldokumentation: schreibe nur denjenigen winzigen Teil einer vollen Dokumentation der am hilfreichsten ist (spart Aufwand, wenig und selten Änderungen, aber was ist der richtige Teil?)
- Hilfreiche Dokumentation: Konzepte erläutern, Problemlösungenbeschreiben (Beispiele)
- Dokumentation als Erfolgsfaktor: bei komplexer Software ist die tollste Funktionalität und Qualität auch nur so viel wert, wie die Dokumenation auch vermitteln kann
- Dokumentation von Architekturentwurf: WIE und WARUM nichtfunktionale Anf. erreicht werden
- Dokumentation von Modulentwurf: Voraussetzungen, Effekt und Zweck für jeden Dienst
- Rationale: Begründung einer Entscheidung des unterliegenden Systems