- Entstehung von SW Engineering
  - Systematisches Vorgehen
  - Vorgehensmodelle
  - Bietet mittel zur Qualitätssicherung
- Software Engineering: Vorgehensmodelle, Methoden und Modelle, Architekturen, Werkzeuge
-Vorgehensmodelle: Traditionell (Wasserfallmodell, V-Model, RUP)
Agil (Scrum, eXtrem Programming)
Hybride Modelle (Traditionell vs. Agil)
- SW Architektur: Gute Architektur, weil immer komplexere Anforderungen die schnell Und kostengünstig hergestellt werden soll.
- Schlechte SW Architektur führt zu: Verzögerungen, Kosten, unzufriedenen Kunden…
Indikatoren dafür → Gesamtüberblick fehlt,
Planbarkeit erschwert,
Wiederverwendbarkeit erschwert,
Performance schlecht,
redundanter Code,
- SW-Architektur Definition: Die grundlegende Organisation eines Systems in seine Komponenten abgebildet, ihre Verbindungen zueinander und zu zur deren Umgebung
- Architekturtreiber: Anforderungen welche die Architektur eines Systems beeinflusst (Bsp.: Funktionale und Nichtfunktionale Anforderungen, Wirtschaftliche Rahmenbedingungen, Rahmenbedingungen der Organisation)
- Systemanalyse: Erfassen der Kundenanforderungen →Mit Problembereich des Kunden befassen
Welt wird als ideal betrachtet, d. h.
noch keine technischen Einschränkungen
Sprache des Kunden/ Domäne
- Systementwurf: Erarbeitung des Lösungskonzeptes → Arbeiten im Lösungsbereich
Welt wird als ideal betrachtet, d. h.
noch keine technischen Einschränkungen
Sprache des Kunden/ Domäne
- Phasen der SW Entwicklung: Analyse, Entwurf, Implementierung, Test, Inbetriebnahme, Wartung
- Paradigmen der SW Entwicklung: Funktions-, Daten-, Objekt-, Aspektorientiert
- Vorteile OO Methodik: Schutz der Daten, Verständlich gegenüber dem Kunden,
Durchgängige Methodik, Übersicht bei großen Systemen,
Wiederverwendbarkeit, Stabilität eines Programms
- Definition Modell: ist eine konsistente Vereinfachung der Realität, welche es ermöglicht ein komplexes System besser zu verstehen.
- Modellierungssprache: UML → mit dieser können Funktionsweisen und das Design eines SW Systems beschrieben werden
Vorteile: Bsp.: Hilfe zur Kommunikation, Reduzierung von Komplexität,
- OO-Analyse: Anforderungen eines Kunden an ein neues SW-System ermitteln und beschreiben.
Meist textuell in Form des Pflichtenheftes.
Ergebnis dieser Analyse ist ein allgemeines Produktmodell (OOA-Modell), dieses
Befinden sich statische Teilmodelle: Klassen des Systems, Assoziationen
Dynamisches Teilmodell: Use Cases, Szenarios
Vorgehen: Systemkontext beschreiben, Use-Case Modell, Pakete bilden
- OO-Design: Spezifizierte Anwendung unter den geforderten Rahmenbedingungen realisieren
  - ➔ Ein Systementwurf wird erstellt
  - ➔ Vorgehen: Randbedingungen analysieren, Architektur definieren, Analysemodell verfeinern und präzisieren

- Gute Architektur erfüllt: Beständigkeit, Zweckmäßigkeit, Eleganz
- Architekt: legt die Anordnung von Teilen (SW+HW Bausteine) eines IT-Systems fest, verwendet unterschiedliche Sichten zur Darstellung einer Architektur und bewegt sich auf unterschiedlichen Ebenen, er muss relevante Anforderungen verstehen und kennen, er kommuniziert mit unterschiedlichen Personen/ Stakeholder
- SW-Architektur: beschreibt dessen Struktur, dessen Bausteine sowie deren Eigenschaften und deren Beziehungen zueinander und zu deren Umwelt
- SW-Architekturen: Systemarchitekturen, SW-Architekturen, Technischen Architektur, Plattform Architekturen, Sicherheits Architekturen, Daten Architektur
- Perspektive WO: Wo befindet sich der Architekt bei seiner Betrachtung bzw. Bearbeitung, welcher Level, welche Sicht, welcher Standpunkt
- Verschiedene Ebenen: Organisationsebene, Systemebene, Bausteinebene
  Vorteile der Ebenen-Strukturierung: Probleme und Aspekte werden Passenden Ebenen zugeordnet
  Probleme und Aspekte werden nicht vermischt
  Einflüsse auf eine Architektur liegen Explizit vor

Makro-Architektur: Findet auf allen Ebenen statt (Abstraktionsniveau hoch)
Software-Architektur
Grob-Entwurf
Architectural Design
High Level / Top Level Design
behandelt: (tragende Bausteine)
Anforderungen, Entscheidungen, Struktur
Entwurf wichtiger Systemschnittstellen

Mikro-Architektur: Findet nur auf einem Teil der Bausteinebene statt (Abstraktionsniveau hoch)
Detailentwurf
Feinentwurf
Detail Design
behandelt: (nichttragende Bausteine)
Nähe zu Source Code
Geringerer Einfluss auf große Architektur

Standpunkt: Spezifikation zu einer Sicht, Festlegung von Methoden der Beschreibung, Template/ Framework-Charakter, Klassen-Charakter, Generische Aspekte zur/ bei der Erstellung von Architekur-Sichten

Architektur-Sicht: Ausprägung einer Sicht, Dokument-Charakter, Objekt-Charakter, Sichten helfen sich auf Problemstellung zu fokussieren, Sichten können parallel erarbeitet werden

Sichten Modelle: Zusammenfassungen verschiedener Standpunkte
→Bsp: 4+1 Sichten Modell: Logische-Sicht, Implementierungs-Sicht, Prozess-Sicht Verteilungs-Sicht, Anwendungsfall-Sicht

- Perspektive Warum: Zentrale Gegenstand der Betrachtung sind die Anforderungen, Fähigkeit die das System besitzen muss essentielle Auswirkung auf Architektur muss →korrekt, eindeutig, machbar und prüfbar sein
- Anforderungen: eine vom Anwender benötigte Fähigkeit des System um ein Problem zu lösen, Eine Fähigkeit die das System besitzen muss, damit es einen Vertrag, Standard, Spezifikation oder ein anders formelles Dokument erfüllt
  Arten: Funktionale Anforderungen, Nichtfunktionale Anforderungen

- WOMIT: Werkzeugkasten (Konzepte, Techniken,…) des SW-Architekten
- 2 Gruppen von Architektur-Mittel: Grundlegende Architektur-Mittel (Prinzipien, Konzepte…)
  Weiterführende Architektur-Mittel (Basisarchitektur, Modellierungsmittel(UML,DSL))
- Architektur-Prinzipien: Sind bewährte Grundsätze zur Gestaltung der Architektur
  Ziel: Reduktion der Komplexität, Erhöhung der Flexibilität
- Grundlegende Architekturprinzipien: Lose Kopplung, Information hiding
- Abgeleitete Architekturprinzipien: Prototyping, BlackBox
- Loose Kopplung: Reduzieren von Abhängigkeiten durch Bsp. Interfaces
  Schnittstellen + Implementierung trennen → durch Abstraktion, information hiding Seperation of Concerns
- Hohe Kohäsion (Single Responsibility): Ist vorteilhaft bezüglich Änderungen und Wartbarkeit, verringert Komplexität
  Kann erreicht werden durch → durch Abstraktion, information hiding Seperation of Concerns
- Entwurf für Veränderungen: Idee ist es vorhersehbare Änderungen zu berücksichtigen und vorauszuplanen, Vorgehensweise → nicht alles hochflexibel entwerfen
  Je loser die Kopplung umso besser gewappnet für Änderungen
  Kann erreicht werden durch → durch Abstraktion, information hiding Seperation of Concerns Modularität
- Separation of Concern: Trennung von Aspekten – Behandlung als Teilproblem
  Unterstützt und erlaubt: Reduktion von Komplexität Arbeitsteilige Bearbeitung
  Vorgehensweise → Trennung von fachlichen und technischen Teilen ist Anzustreben
- Information Hiding: Klienten sehen nur notwendige Teilausschnitte von Informationen
  → Public/ private, Facade Muster, Schichtenbildung
- Abstraktion: Wichtige Aspekte identifizieren, vereinfachte Details weglassen (Wichtiges vom unwichtigem separieren)
-Modularität: Klar abgegrenzte Systembausteine mit den Vorteilen:
  Änderbarkeit, Erweiterbarkeit, Wiederverwendbarkeit
  Ansätze zur Erzielung → Objektorientiertheit, Komponentensatz, Schichten-Architektur
- Rückverfolgbarkeit: Traceability, RQ-Keys, Kommentare
  Gut dokumentierter Code, Architektur Beschreibung…
- Weitere Prinzipien: Bezug zu Anwendungsfällen, Vermeidung überflüssiger Komplexität
  Konsistenz, Konventionen statt Konfigurationen
- Architektur-Konzepte: Prozedurale Ansätze → C, Cobol (Funktionen, Globale Variablen)
  Objektorientierung → Klassen, Objekte, Assoziation, Vererbung, Schnittstellen. Framework → stellt Bausteine zur Verfügung
  Komponentenorientierung → wiederverwendbar, geschlossene Bausteine, Loose Kopplung, Modularisierung
  Meta-Architekturen → Erreichung höherer Flexibilität und Kontrolle durch Einführung zusätzlicher Abstraktionsebenen
  Modellgetriebene Archit. → Modelle für Abläufe
  Aspektorientierte Ansätze
  Skriptsprachen → weniger performant, schnelle Entwicklung, Kernsystem in Programmiersprache dynamisch / agile Sprachen, gut geeignet um DSLs zu implementieren

- WOMIT: Basisarchitekturen →konkrete Architekturmittel, mit denen man Systeme ganzheitlich Strukturieren kann bzw. sind Vorlagen
- Schichtenarchitektur: Unterteilung von System in Gruppen mit ähnlichen Funktionen
  Gruppen sind in Layers organisiert, diese können nur auf unmittelbar Benachbarte Schichten zugreifen. Jede Schicht bietet Dienste für die höhere an. → Erhöht Veränderbarkeit des Systems, System wird portierbar gestaltet
  Wiederverwendbarkeit wird erhöht
- Datenflussarchitektur: Gesamtaufgabe wird in Teilschritte zerlegt, Daten werden von Baustein zu Baustein übergeben. Bsp.: Pipes and Filters
- Repositories: Verschiedenen Bausteinen den gleichzeitigen Zugriff auf Daten ermöglichen
- Zentralisierung vs Dezentralisierung: Zentralisieren: Aufgabe auf einen Baustein bündeln
  Dezentralisieren: Aufgabe auf mehrere Bausteine verteilen
- Rich Client vs Thin Client: Thin Client: "nur" Konsole
  Rich Client: viele Funktionalität
- n-Tier Architektur: 2,3,5,… n-tier Architekturen
  Klassische Client/ Server-Architektur: 2-tier
  Bsp.: Einziehen einer Schicht zur Ansteuerung redundanter Server
  Nachteile: komplexe Funktionen für triviale Operationen
  Mehr Schichten = mehr Komplexität, mehr Wartungsaufwand
  Anzahl der Schichten abhängig von der Anforderung
- P2P Architektur: Jeder Client kommuniziert explizit mit dem Server
  ➔ Es gibt keinen zentralen Server, jeder Peer kann im Netz Services anbieten
  ➔ Gesamtzustand über Peers verteilt
- Publisher/ Subscriber Architektur: Register & Notify
  Man kann sich auf ein Event subscriben über das man informiert werden will.
  Entsrpicht Implicit Invocation (ich will event x, da hast event x)
- Middleware: Ist eine Plattform, die Anwendungen Dienste für alle Aspekte der Verteilung anbietet.
  Verwendung bei verteilten Systemen z.B. auf Basis des Broker Musters
  Herausforderung → Aufrufzeit länger, Vorhersagbarkeit der Aufrufzeit, Nebenläufigkeit, Skalierbarkeit des Systems
- Komponentenplattformen: Im Enterprise Umfeld → JEE, .NET
  Trennung von Technischen und Fachlichen belangen.
  Technische Belangen: Verteilung, Sicherheit, Persistenz…
  Fachliche Anforderungen: Entitäten-persistent, Sessions, Services
- SOA: SW-Bausteine als lose gekoppelt, verteilte, wiederverwendbare und standardisierte Dienste
  Dienste im SOA zeichnen sich aus durch → stärker strukturiert als Komponenten
  Erlauben Anonyme Nutzung
  Sind gewissermaßen selbstbeschreibend
  SOAs können geschichtet Aufgebaut sein und unterstützt Wiederverwendbarkeit, SOA Architekturen sind Zusammenfassungen von Design Patterns und Architekturansätzen
-Sicherheitsarchitekturen: bezieht sich auf eine zu schützende bzw. sichernde Anwendung
  Systembaustein, die einer Sicherheitsinfrastruktur zuzurechnen sind
  Voraussetzung bzw. Vorhandensein von: Sicherheitssysteme auf allen Netzwerkebenen
  Authentifizierungs-System
  Autorisierungs –System…
  Ansätze → Eigenbau (schwer integrierbar)
  Standarddienste-basiert(verwenden von bewährten Lösungen)
  Komponentenplattform-basiert(Sicherheitsfunktionalität wird breit gefächert bereitgestellt)

Referenzarchitekturen: Verbund zw. Architektur Expertise und industriespezifischer Kenntnisse
= Vorlage für Softwaresysteme

Vorteile: Aufbau auf Wissen und Erfahrung
Senkung von Risiko von nicht tragfähigen Architekturen
Senkung der Kosten (nicht alles neu)
Schnellere Entwicklung

Anforderungen: basierend: Prinzipien, Konzepten, Taktiken, Stilen, Mustern
musst bewährt / erfolgreich eingesetzt worden sein
konkret an Bedürfnisse anpassbar sein
umfassende dokumentiert sein

- WER: Software Architekt → Bearbeitet Struktur / Design von SW Systemen
  Funktionale Anforderungen, Qualitätsattribute
  Rolle wird einem Projekt zugeordnet
- Solution Architekt → Fokus auf Business Requirements und IT Capabilities / Services
  Einzelprojekte unter Einhaltung übergreifender Architekturprinzipien
  Bedeutsam bei größeren projekten mit unterschiedlichen Systemen
- Enterprise Architekt → auf Unterstützung der Geschäftsstrategie durch IT Strategie
  Strategische Ausrichtung der IT Capabilities unter wirschaftlichen Aspekten
  Etablierung von Standard und unternehmensweite Governance
- Architekt: muss Architekturmittel und seine Erfahrung zielgerichtet und bewusst einsetzen
- Software Architektur: umfasst neben der SW-Strukturen auch den Weg dorthin.
- Vorgehen: Erstellen der Systemversion: Anforderungssicht, Logische Sicht
  Architekt soll mitwirken und ist Berater
  Verstehen der Anforderungen: Anforderungssicht
  Entwerfen der Architektur: Logische Sicht, Implementierungssicht, Verteilungssicht, Prozessicht
  Details: Kontext →Schnittstellen und Systembausteine, Dienste/ Akteure
  Schlüsselabstraktion → Abstraktion (fachliche Ebene -> fachliche Architektur)
  Architekturrelevante Anforderungen → Anforderungen mit hoher Priorität früh einfließen lassen
  Architekturmittel → rein Zweckmäßig folgen, korrekte Verwendung achten, Mittel für fachliche/ technische- Architektur einsetzen
  Bausteine → es gibt fachliche und technische Bausteine, Identifikation erfolgt Durch ausgewählte Architektur-Mittel
  Schnittstellen → Bausteine kommunizieren über Schnittstellen, stellen selbst Dienste zur Verfügung
- Umsetzen der Architektur: Definition von Entwicklungsrichtlinien, manuelle Reviews, Etablierung Infrastruktur zur architekturkonforme Umsetzung
- Entscheidungen: Entscheidung vorbereiten
  Entscheidung treffen
  Entscheidungen kommunizieren
  Entscheidung realisieren
  Entscheidung beurteilen

Muster: -wiederkehrende Problemstellungen & bewährte Lösungsvorschläge
-Idee, die sich als nützlich erwiesen hat
-Kontext, Problem, Lösung

Mustertype: -Entwurfsmuster(Prototyp):bewährte Lösungsvorschläge
-Architekturmuster(MVC):Helfen beim Zerlegen des Systems
-Idiome(Singleton): ausprogrammiert

Einsetzen wenn sinnvoll und Vorteile bringen

Vorteile: Risiko von Fehlern senken, Entwürfe flexibler gestalten

Nachteile: Zusätzliche Klassen & Interfaces entstehen, Mehrwert prüfen