

Software Engineering

Definition:

Software Engineering zielt auf die ingenieurmäßige Entwicklung, Anpassung und Weiterentwicklung großer Softwaresysteme unter Verwendung bewährter systematischer Vorgehensweisen, Prinzipien, Methoden und Werkzeuge.

1. Grundlagen des Software Engineering

Fehlerhaftes SW-Projekte:

- Mars Climate Orbiter
 - Verlorengegangen aufgrund eines Einheitenfehlers
- RADAR System auf Grönland
 - Wird durch aufgehenden Mond irritiert -> beinahe Atomschlag gegen Russland

LOC = Lines of Codes (bsp. Handy bis zu 600 Fehler -> 3 Fehler pro 100 loc)

P = Produktivität (LOC / MannJahr)

M = Personalkapazität (Anzahl Mitarbeiter)

Arten von Software

- Produkt oder Systemsoftware
- Meistens datenintensiv oder berechnungsintensiv

Eigenschaften von Software

- Immateriell
- Veraltet
- Leicht änderbar
- Ständig anpassen (=Update)

Zielorientiertes Software Engineering

- Qualität
- Kostenaufwand (Entwicklungskosten, Wartungskosten) Optimum -> das Median von beiden
- Zeit (Entwicklungsdauer = $LOC / (P * M)$)
- Ertrag

2. Überblick

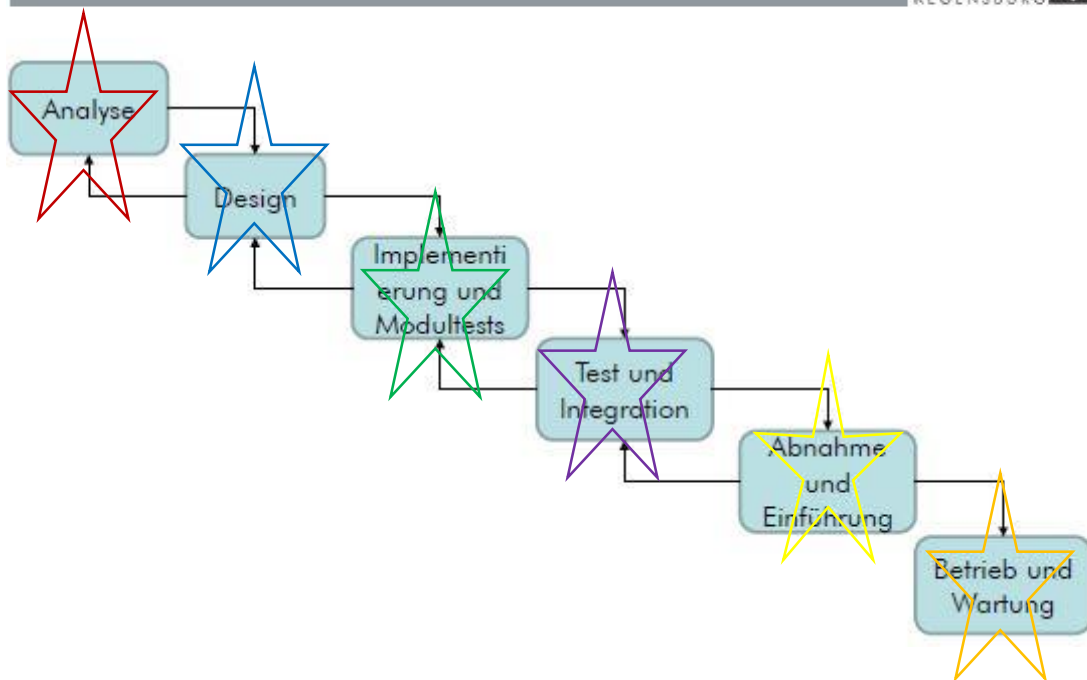
- Projektantrag
 - Darlegung der Projektidee (Bsp.. Als Präsentation)
 - Diskussion/Abwägung zur Folge
- Projektauftrag
 - Startschuss des Projekts

Jedes Unternehmen handhabt es sehr unterschiedlich. (Kultur, Struktur)

Projekte können auch extern vergeben werden:

- Kunde
- Angebote
- Auftragsvergabe

Phasen eines Software Projekts





ANALYSE

- Ziele, Einschränkungen und Leistungen werden ermittelt
- Detailliert definiert

1. Requirements Engineering

Requirements Engineering ist der Prozess des Herausfindens, Analysierens, Dokumentierens und Überprüfens der Dienste und Beschränkungen eines zu erstellenden Systems.

Ziele:

- Pflichtenheft
- Konzept des User Interface
- Analysemodell

Hauptaufgaben des Requirements Engineering

- Ermitteln
- Dokumentieren
- Prüfen und abstimmen
- Verwalten

Aktivitäten und Artefakte



Visionen und Ziele

- Sind oftmals im Projektauftrag und Projektantrag formuliert
- Vision
 - o Leitgedanke
 - o Geringe Detailtiefe
 - o Gewünschte Zukunft
- Ziele
 - o Vision verfeinern und operationalisieren
 - o Ausgehend einer Vision

Rahmenbedingung

Eine Rahmenbedingung legt **organisatorische** und/oder technische Restriktionen für das Softwaresystem und/oder den Entwicklungsprozess.

- Organisatorische Rahmenbedingung
 - o Anwendungsbereiche
 - o Zielgruppen
 - o Betriebsbedingungen
- Technische Rahmenbedingung
 - o Technische Produktumgebung
 - o Anforderung an die Entwicklung

Anforderungen

Anforderungen legen fest, was man von einem Softwaresystem **als Eigenschaft erwartet**.

- Funktionale Anforderungen
 - o Beschreiben
 - o Dienste
 - o Reaktion
 - o Verhalten
- Leistungsanforderungen
 - o Zeit
 - o Geschwindigkeit
 - o Umfang
 - o Durchsatz
- Nichtfunktionale Anforderungen
 - o Beschränkungen
 - o Gesamtbetrachtung
- Spezifische Qualitätsanforderungen
 - o Zuverlässigkeit
 - o Benutzbarkeit
 - o Sicherheit
 - o Usw
 - Qualitätsmerkmale nach ISO/IEC 9126-1
 - Funktionalität
 - Benutzbarkeit
 - Effizienz
 - Zuverlässigkeit
 - usw

- Einschränkungen
 - Kulturell
 - Rechtlich
 - Umgebung
 - Schnittstellen
 - Usw

Anforderungen an Anforderungen

- Korrektheit
- Eindeutigkeit
- Vollständigkeit
- Konsistent
- Verfolgbar
- Usw

Aktivitäten des RE

Systemkontext festlegen

Definition und das Verständnis der Anforderungen des Systems.

Ziel:

- Abgrenzen des Systems von der Umgebung
- Identifikation der für die Anforderungen relevanten Umgebungsteile

Typen & Aspekte:

- Personen (Stakeholder oder Stakeholdergruppen)
 - o Eine Person oder Organisation, die Einfluss auf die Anforderungen des betrachteten Systems hat. (Er ist die Quelle!)
 - o Stakeholder Liste: Listet alle für das Projekt relevanten Stakeholder auf
 - o Stakeholder Portfolio: Zeichnet das Wohlwollen und den Einfluss zwischen den Stakeholdern
- System im Betrieb
- Prozesse
- Dokumente

Kontextdiagramm:

- Erfassung des Systemkontext erfolgt textuell und in einem Kontextdiagramm
- Beschreibt den Umfeld ihres Systems (Schnittstellen)

Anforderungen erfassen

Ziel:

- Vollständige und fehlerfreie Anforderungen aufnehmen für die Entwicklung.

Ermittlungstechniken für Anforderungen:

- Befragungstechnik
- Dokumentgetrieben
- Kreativitätstechniken
- Beobachtungstechniken

Anforderungen dokumentieren

Anforderungen, die spezifiziert sind, werden in einem Dokument definiert.

Lastenheft vs Pflcihtenheft (Sprachliche Dokumentation):

- Lastenheft
 - Wird vom Auftraggeber erstellt
 - Lässt Details bewusst offen
- Pflichtenheft
 - Wird vom Auftragnehmer erstellt
 - Detailliert
- Anforderungen ans Dokument:
 - Korrektheit
 - Vollständigkeit
 - Konsistent
 - Usw

Dokumentation durch ein Modell / UML

- Usecases
- Aktivitätsdiagramme
- Klassendiagramme
- Zustandsdiagramme

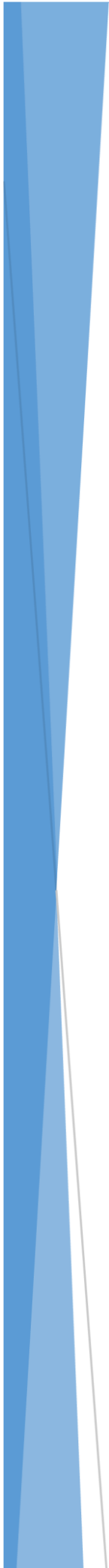
Anforderungen prüfen und abstimmen

Jedes Artefakt einer SW-Entwicklung ist einer Qualitätsüberprüfung zu unterziehen.

- Analyse der Anforderungen
 - Prüfung Qualitätsanforderungen vs Anforderungen
 - Validierung der Anforderungsspezifikation
 - Prüfung Korrektheit der Beschreibung des Produkts
- ➔ Formelle Abnahme der Anforderungsspezifikation ist empfohlen!

Kano-Modell:

- Basis-Merkmale
 - Unzufriedenheit steigt wenn FAIL
 - Zufriedenheit steigt nicht wenn WIN
- Leistungs-Merkmale
 - Abhängig vom Ausmaß der Erfüllung, Kunde bleibt zufrieden
- Begeisterungs-Merkmale
 - Nutzen enorm, da Begeisterung vorliegt.
- Unerhebliche Merkmale
 - Keine Zufriedenheit und keine Unzufriedenheit. Kunden ist es egal!
- Rückweisungs-Merkmale
 - Sind immer Unzufrieden wenn vorhanden oder nicht.



Anforderungen verwalten

- Toolgestützt
- Änderung
- Nachverfolgbarkeit
- Zustandsmodell

Risiken des RE

- Unzureichend repräsentiert
- Kritische Anforderungen nicht beachtet, nur funktionale
- Anforderungen werden geändert -> unkontrolliert
- Keine Qualitätsprüfung
- Usw

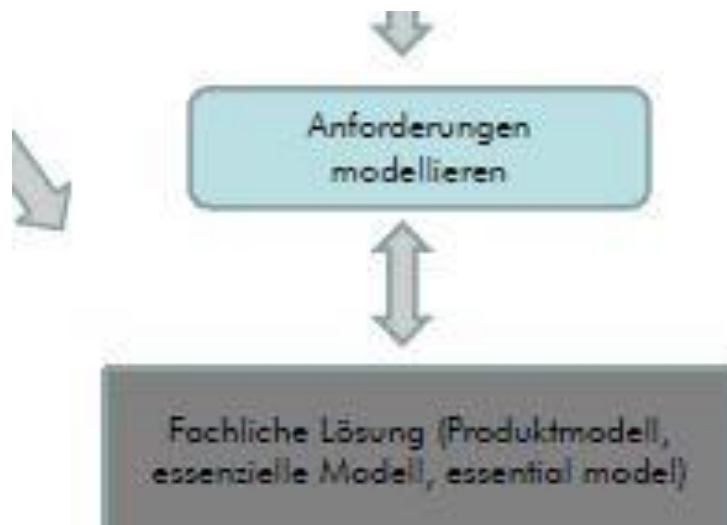
Best Practices beim RE

- Kunden und User einbinden
- Konsultieren aller RE Quelle
- Erfahrene Mitarbeiter
- Gute Atmosphäre, gutes Arbeitsklima
- Prioritäten setzen
- Usw

Aufwandschätzung

- Kalkulation
- Personalplanung
- Entscheidung vorbereiten
- Expertenschätzung
 - Fachleute nutzen ihre Erfahrung
 - Einzelschätzung (vom Einzelnen)
 - Mehrfachbefragung (mehrere Personen/Schätzungen)
 - Delphi-Methode (Unabhängige Schätzungen) **OFT ANGEWENDET!**
 - Schätz Klausur (kollektive Schätzungen in der Gruppendynamik)
- Algorithmische Schätzung
 - Kosten werden aus Größen berechnet. Frühzeitige Erkennung führt zur Schätzung.
 - COCOMO (Constructive Cost Model)
 - Function Point Verfahren

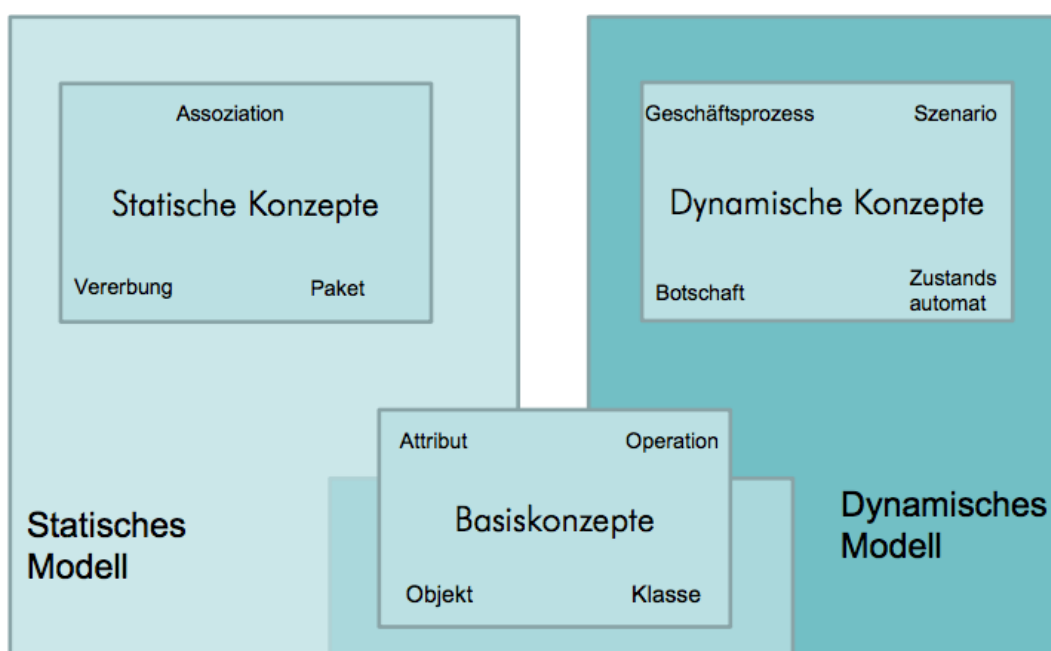
2. OOA (Objektorientierte Analyse)



Ermittlung und Beschreibung der Anforderungen an ein Software System mittels objektorientierter Konzepte und Notationen. Das Ergebnis ist ein OOA Modell.

Produktmodell:

- Korrekte Umsetzung Anforderungen
- Vollständige Umsetzung
- Referenzierung der Anforderungen
- Präzise, eindeutig, konsistente Formulierung fachlicher Lösung



OOA – Basiskonzepte:

- Objekt
 - Gegenstand des Interesses
 - Zustand \Leftrightarrow Verhalten auf die Umgebung
- Klasse
 - Objekte
 - Attribute & Operationen
- Attribut
 - Daten
- Operation
 - Ausführung eines Objekts

Makroprozess OOA:

- Relevante Geschäftsprozesse ermitteln
- Klassen ableiten
- Statisches Modell
- + parallel dynamisches Modell
- Wechselwirkung berücksichtigen

Szenariobasierter Makroprozess

- Geschäftsprozesse formulieren
- Szenarios aus den Geschäftsprozessen ableiten
- Interaktionsdiagramme aus den Szenarios ableiten
- Klassendiagramme erstellen
- Zustandsdiagramme erstellen

Datenbasierter Makroprozess

- Klassendiagramme erstellen
- Geschäftsprozesse formulieren
- Szenarios aus den Geschäftsprozessen ableiten
- Interaktionsdiagramme aus den Szenarios und dem Klassendiagramm ableiten
- Zustandsdiagramme erstellen

UML

Die Unified Modeling Language, kurz UML, ist eine grafische Darstellungsform zur Visualisierung, Spezifikation, Konstruktion und Dokumentation von (Software) Systemen. Sie bietet ein Set an standardisierten Diagrammtypen, mit denen komplexe Sachverhalte, Abläufe und Systeme einfach, übersichtlich und verständlich dargestellt werden können.

UML – Diagrammtypen

- **Strukturdiagramm**
 - **Klassendiagramm (Analyse und Designphase)**

Klassen, Attribute und Operationen werden erstellt um einen besseren Überblick zu erschaffen und um den Übergang zur Implementierung zu erleichtern.

Grundlegende Elemente:

- Klassen
- Assoziationen
- Aggregation
- Komposition
- Generalisierung
- Spezialisierung
- Vererbung
- Attribute
- Operationen

- **Objektdiagramm**

Stellt eine Momentaufnahme des Systemzustandes. Wie Klassendiagramm nur die dazugehörige Belegung.

Grundlegende Elemente:

- Instanz/Objekt
- Link
- Wert

- **Paketdiagramm (Analyse und Designphase)**

Den Überblick zu erhalten auf ein großes SW System mit vielen Use-Cases, Activity Diagramme etc.

Grundlegende Elemente:

- Paket
- Import
- Merge

- **Architekturdiagramm**
 - **Kompositionsdiagramm**
 - **Komponentendiagramm (Designphase)**

Zeigt die Struktur des Systems und deren Erzeugung.

- **Einsatz- und Verteilungsdiagramm (Designphase)**

Zeigt die Verteilung der Artefakte des Systems zur Laufzeit.

- **Verhaltensdiagramm**
 - **Use-Case Diagramm (Analysephase)**

Ein Use-Case ist eine Beschreibung einer Interaktion zwischen Benutzer und dem zu entwickelnden System.

- Beschreibt einzelne Schritte die zur Zielerreichung notwendig sind
- Abstraktion
- Beschreibt aus einer externen, fachlichen Sichtweise
- Dienstleistung

Zwei Beschreibungsvarianten:

- Use-Case-Diagramm
 - Externe Verhalten eines Systems gegenüber seinen Benutzern
 - Strukturelle Abhängigkeiten zwischen den Anwendungsfällen und den Benutzern
- Use-Case-Definition
 - Textuelle Beschreibung der Arbeitsläufe

- **Aktivitätsdiagramm (Analyse und Designphase)**

Werden verwendet um die Abläufe/Prozesse in Zusammenhang mit einem System zu modellieren.

Grundlegende Elemente:

- Aktion
- Startknoten
- Endknoten
- Ablaufende
- Entscheidung
- Zusammenführung
- Zusammenführung-Block
- Teilung
- Synchronisation
- Signal senden
- Signal empfangen
- Partition
- Wanderung
- Objektknoten
- Kanten
- Unterbrechungsbereich
- Exceptionhandler
- Strukturierte Knoten (Schleifen, Entscheidungen)

- **Zustandsdiagramm (Designphase)**

Dient zur Modellierung von Verhalten von Klassen welches an einem internen Zustand der betrachteten Klasse hängt.

Grundlegende Elemente:

- Einfacher Zustand
- Zustand
- Transition/Trigger
- Start Zustand
- End Zustand
- Entscheidung
- Terminator
- Zusammengesetzter Zustand
- Eintritt in einen zusammengesetzter Zustand
- Verlassen eines zusammengesetzten Zustands
- Teilung
- Synchronisation

- **Interaktionsdiagramm**

- **Interaktionsdiagramm**

- **Sequenzdiagramm (Designphase)**

Liefert Informationen über den Ablauf der Kommunikation in einem System.

Grundlegende Elemente:

- Objekt
- Nachricht
- Lebenslinie
- Synchroner Methodenaufruf
- Asynchroner Methodenaufruf
- Rekursiver Methodenaufruf
- Lokaler Methodenaufruf
- Objekt-Zerstörung Methodenaufruf
- Operatoren
- Objekt-Erzeugung Methodenaufruf
- Bedingungen
- Fragmente

- **Kommunikationsdiagramm (Designphase)**

Zeigt das Wechselspiel und den Nachrichtenaustausch von Teilen einer komplexen Struktur

- **Interaktionsübersicht**

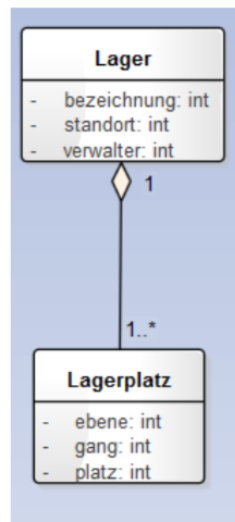
- **Zeitdiagramm**

Was sind Patterns?

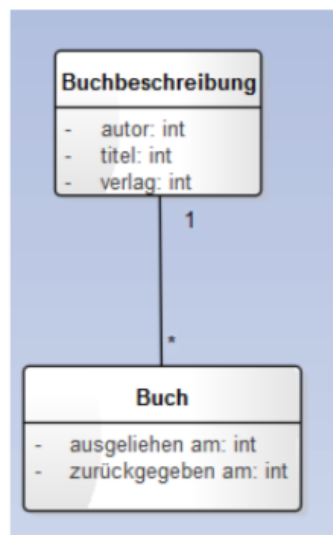
- Patterns sind wiederverwendbare Lösungen zu wiederkehrenden Problemen.
- Vereinfachen das Design
- Erleichtert Dokumentation
- Erleichtert vieles im Hinblick der Entwicklung

Muster/Analysemuster /-patterns

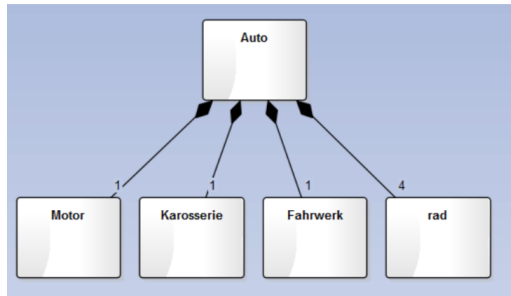
- Wiederverwendung
 - Vereinfachen Design
 - Doku + Wissenstransfer erleichtert
 - Gemeinsame Sprache
 - Ingenieurmäßiges Vorgehen bei SWE
 - Leichter Zugang zu OO Sprachen
- Liste
 - Auflisten der Objekte
 - Ein Teil ist einem Ganzen zugeordnet



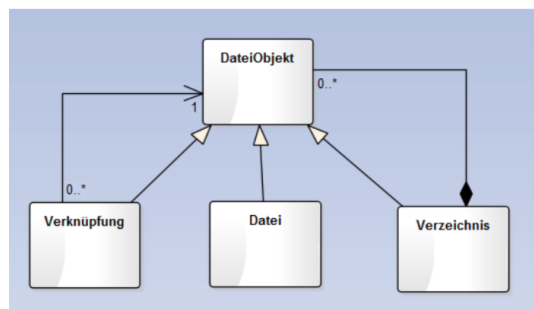
- Exemplar-typ
 - Jede Sache existieren viele Exemplare. Jedes Exemplar hat eigene Eigenschaften.



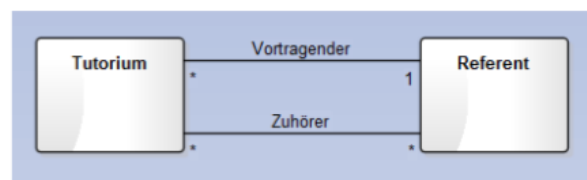
- Baugruppe
 - Ein Ganzes existiert nur mit seinen Einzelteilen
 - Ein Ganzes existiert aus Einzelteilen



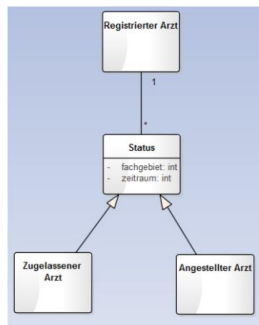
- Stückliste
 - Zwischen den Komponenten besteht eine rekursive Beziehung, dies soll als Struktur festgelegt sein und als Einheit behandelbar sein
 - Bsp.: Dateisystem, Ordner enthält Dateien usw. + evtl. wieder Ordner



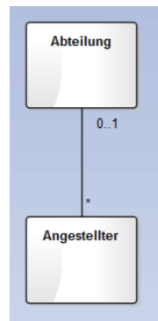
- Koordinator
 - Es herrschen Beziehungen zueinander, über die Beziehung werden Daten vorgehalten.
- Historie
 - Für ein Objekt sollen mehrere Vorgänge beziehungsweise Fakten dokumentiert werden. Die Änderungen sollen festgehalten werden.
- Rollen
 - Jedes Objekt kann verschiedene Rollen spielen. Unabhängig von der Beziehung.



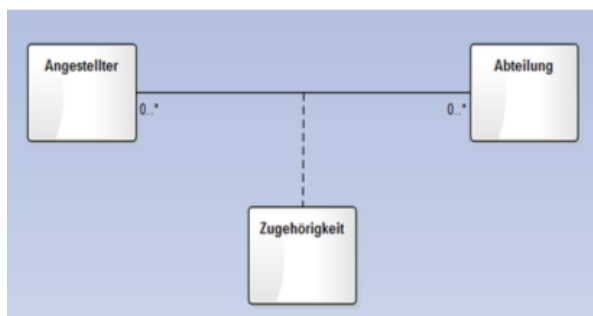
- Wechselnde Rollen
 - Nach der Zeit ändern sich die Rollen. Die müssen nachvollziehbar sein.



- Gruppe
 - Objekte sollen einer Gruppe zugeordnet werden können



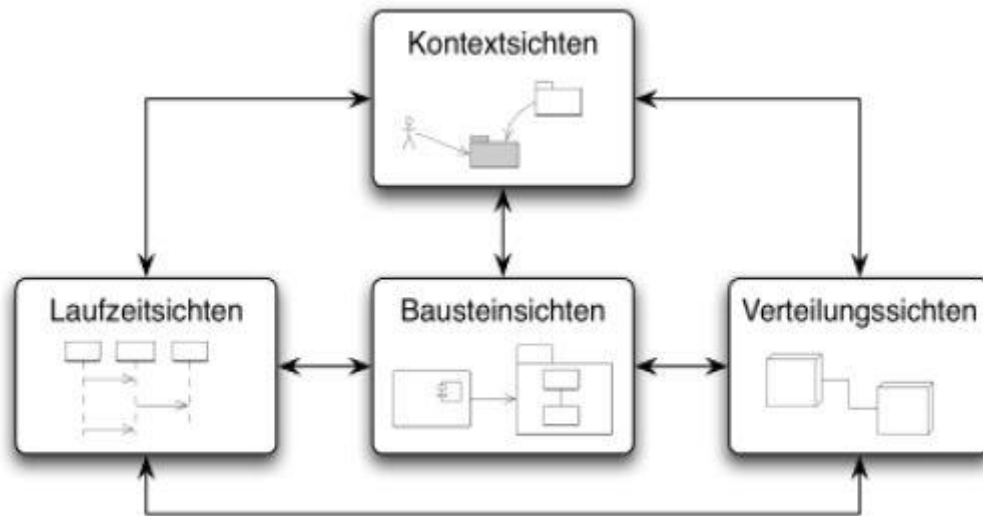
- Gruppenhistorie
 - Objekte sollen einer Gruppe zugeordnet werden können, mit Abhängigkeit deren Zugehörigkeit





DESIGN

- Ziel
 - Realisierung?
- Aktionen
 - Ermitteln von Umgebungs- und Randbedingungen
 - Entscheidungen treffen
 - Spezifikationen
 - Programmierung
- Ergebnis
 - SW-Architekturmodell
 - Systemkomponenten
 - Schnittstellen zwischen Komponenten und zur Umgebung des Produkts



4 Sichten nach Starke

- Kontextsichten
 - Wie ist das System in seine Umgebung eingebettet?
 - Blackbox-System aus einer Vogelperspektive
 - Schnittstellen zu Nachbarsystemen
 - Interaktion mit wichtigen Stakeholdern
 - Use-Case Diagramm für das gesamte System
 - Klassendiagramm für Assoziationen zu anderen Systemen sowie auch für die Darstellung der großen Systemstruktur
 - Sequenz-, Kommunikation- oder Aktivitätsdiagramme für Abläufe
 - Verteilungsdiagramm für das Zeigen der technischen Systemumgebung
- Bausteinsichten
 - Wie ist das System intern aufgebaut?
 - Diese zeigen die Strukturen der Architekturbausteine des Systems, Subsysteme, Komponenten und deren Schnittstellen.
 - Diese unterstützen Projektleiter und Auftraggeber bei der Projektüberwachung. Referenz für Softwareentwickler.
 - Paketdiagramme
 - Klassendiagramme
 - Komponentendiagramme
- Laufzeitsichten
 - Wie läuft das System ab?
 - Zur Laufzeit wird ermittelt welche Bausteine existieren und wie sie zusammenspielen.
 - Dynamische Strukturen
 - Sequenzdiagramme
 - Aktivitätsdiagramme
 - Kommunikationsdiagramme
- Verteilungssichten (Infrastruktursichten)

- In welcher Umgebung läuft das System ab?
 - Diese beschreiben auf Hardwareebene wie das System abläuft.
 - Dokumentieren alle Bestandteile des physischen Systems.
 - Betreibersicht
- Deployment Diagramme ???

Merkmale schlechter SW:

Steifheit (Rigidity):

- Kleine Änderungen sind schwierig, aufwendig, riskant
- Symptom: Nicht kritische Fehler werden nicht behoben

Zerbrechlichkeit (Fragility):

- Kleine Fehler lösen Fehler / Anomalien aus
- Symptom: Viel Zeit für Bugfixing

Unbeweglichkeit:

- Viele Abhängigkeiten
- SW kann nicht wiederverwendet werden
- Symptom: Parallele Produktlinien mit ähnlicher Funktion

Zähigkeit (Viscosity):

- Änderung gegen Entwurf leichter als mit
- Symptom: Ungenutzter/Redundanter Code

Guter OO Entwurf:

Prinzipien:

SOLID: Single Responsibility, Open/Closed Principle, LSP, Interface Segregation

GRASP: General Responsibility Assignment Patterns

Qualitätskriterien:

- Korrektheit
- Anpassbarkeit
- Verständlichkeit
- Ressourcenschonung

Architekturprinzipien

- Lose Kopplung
 - Hohe Abhängigkeit zwischen Modulen (nur bei Hohe Kopplung)
 - Bei niedrige Kopplung hat geringe Auswirkung auf andere Module durch Veränderungen
 - Erleichtert die Wartbarkeit
 - System ist stabiler
 - Ziel: Eine Reduktion der Kopplung zwischen Modulen erreichen
- Hohe Kohäsion
 - Sachen sollten in Struktureinheiten zusammengefasst werden, die auch inhaltlich zusammenpassen.
 - Hohe Abhängigkeit
 - Ziel: Hohe Kohäsion erreichen
- Information Hiding
 - Reduzierung des Informationsmissbrauch
 - Die Art und Weise wie ein System funktioniert darf nur intern des Moduls bekannt sein
 - Nur Information die zur Verfügung gestellt worden sind kann man außen bekannt geben
 - Kapselung (Getter / Setter – Methoden)
- Open Closed
 - Module (wie Klassen, Funktionen, Packages usw.) sollen für Erweiterungen offen und für Veränderungen geschlossen sein
 - Polymorphie (Veränderungen ohne die Basisklasse zu ändern)
- Abhängigkeit nur von Abstraktionen
 - Erlaubnis Abhängigkeiten nur von Abstraktionen (abstrakte Klassen)
 - Vermeidung zyklischer Abhängigkeit und Abhängigkeiten prozeduraler Systeme
- Keine zyklischen Abhängigkeiten
- Liskov Substitutionsprinzip
 - Klassen sollen von Unterklassen ersetzbar sein
- Dependency Injection
 - Dependency Injector (Assembler) erzeugt eine „Injektion“ um die Laufzeit eines Objektes zu ermitteln
- DRY Prinzip (Don't Repeat Yourself)
 - Wiederholungen vermeiden
 - Ziel: Vererbung
- Interne Wiederverwendung
 - Reduktion der Redundanz
 - Erhöhung der Stabilität und Ergonomie
 - Gefahr: Kopplung kann erhöht werden!!
 - Hilfsmittel
 - Ziel: Hohe Wiederverwendbarkeit erreichen

Architekturmuster

Diese helfen bei der Strukturierung von Systemen und Anwendungen

- Schichtenarchitektur
 - Das System wird in mehrere Schichten aufgeteilt
 - Fasst logisch zusammengehörende Komponente zusammen
 - Stellt Dienstleistungen über Schnittstellen zu Verfügung
 - Zugriff nur direkt vom Vorgängerschicht
 - Gekoppelt wenn benachbart
 - 3 Schichtenarchitektur
 - Presentation Layer
 - User Interface
 - Info darstellen
 - Interaktion System- Mensch
 - Application Layer
 - Anwendungen realisieren
 - Verwaltet alle Objekte und Klassen
 - Domainmodell
 - Persistence Layer
 - Abstrahiert die Art des Speichermediums von der Applikation
 - Speichert die Objekte ab
 - Objekte -> Relationen
 - Vorteil
 - Können verteilt werden
 - Unabhängig voneinander
 - Austauschbar
 - Gutes Strukturkonzept
 - Nachteil
 - Performance Probleme
 - Neuerungen werden schlecht unterstützt vom System
- Pipes & Filters
 - Daten werden von einem Subsystem zum nächsten weitergereicht
 - One Way Kommunikation
 - Vorteil
 - Leicht ersetzbar
 - Einfache Schnittstelle
 - Intuitiv
 - Nachteil
 - Folgefehler können auftreten
 - Keinen gemeinsamen Zustand
- Blackboard
 - Alle Subsysteme greifen auf einen gemeinsamen Datenbank
 - Vorteile:
 - Keine direkte Kommunikation zwischen den Subsystemen notwendig
 - Starke Kohäsion

- Lose Kopplung
- Nachteile
 - Uniformität vom Datenformat
 - Flaschenhalsproblem
- Peer to Peer
 - Netzwerk verbundene Systeme
 - Vorteil
 - Hohe Ausfallsicherheit
 - Kein Flaschenhals
 - Nachteile
 - Auffinden und Erkennen von Peers in großen Netzen schwierig
 - Fehlerbehandlung
 - Ist für kleine Netze praktisch



ENTWURF UND IMPLEMENTIERUNG

Detailaspekte der Software Entwicklung

- Wiederverwendung
 - Abstraktionsebene
 - Nutzen von Wissen erfolgreicher Abstraktionen (z.B. Architektenmuster)
 - Objektebene
 - Verwendung von Objekten einer Bibliothek
 - Komponentenebene
 - Verwendung von Frameworks
 - Systemebene
 - Wiederverwendung von gesamten Anwendungssystemen
- Kosten der Wiederverwendung
 - Suchen und Evaluierung
 - Kaufkosten
 - Customizing und Konfiguration
 - Integration verschiedene Komponenten
- Konfigurationsmanagement
 - Das ist der Prozess, ein sich änderndes Softwaresystem zu verwalten.
 - Aktivitäten:
 - Versionsmanagement
 - Verwaltung der verschiedenen Versionen
 - Systemintegration
 - Unterstützung bei der Festlegung, welche Version welcher Komponente verwendet wird.
 - Problemverfolgung
- Werkzeuge der Entwicklungsplattform
 - **Design Pattern**
 - Design Patterns sind wiederverwendbare Lösungen zu wiederkehrenden Design Problemen.
 - Nutzen:
 - Unterstützt die Wiederverwendbarkeit
 - Vereinfacht Design
 - Erleichtert Dokumentation
 - Erleichtert hinsichtlich alle Aspekte der Entwicklung
 - Einige Design Patterns
 - Singleton
 - Eine einzige Instanz der Klasse gibt es und somit einen globalen Zugangspunkt zu dieser Instanz
 - Factory Method
 - Verwendung einer statischen Methode anstelle eines Konstruktors

- Decorator
 - Erlaubt es leichtgewichtig und flexibel das Verhalten bestehender Klassen zu erweitern
- Intent
 - Provides a uniform way of traversing the elements of a collection object



TEST UND INTEGRATION

Testen soll zeigen, dass das Programm tut, was es soll und soll Fehler entdecken, bevor das Programm produktiv benutzt wird.

Validierungstests:

- Erfüllt die gestellten Anforderung für den Entwickler und dem Kunden

Fehlertests:

- Fehler ermitteln in der Software

Verifikation:

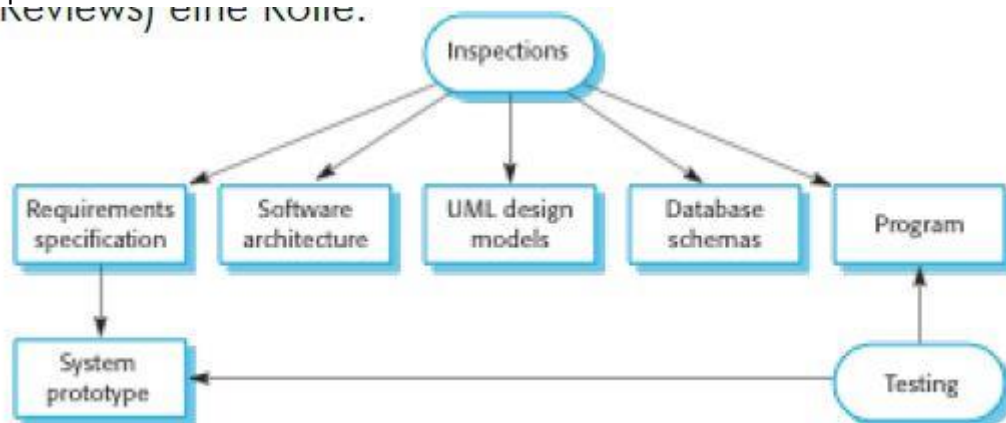
- Prüfen gegen die Spezifikation. Produkt richtig gebaut?

Validierung:

- Prüfen gegen die wirklichen Bedürfnisse des Nutzers. Richtigen Produkt gebaut?

Inspektionen

Reviews) eine Rolle.



Der Verifikation und Validierung-Prozess schafft das Vertrauen, dass das SW funktioniert. Es gibt dynamische Test und statische Tests (Inspektionen).

Vorteile:

- Fehler können andere Fehler überdecken
- Unvollständige Versionen testen mit weiteres testen
- Qualitätsmerkmale prüfen

Nachteile:

- Benutzerfreundlichkeit nicht ermitteln
- Performance, Usability nicht ermitteln

Entwicklertests:

Entwicklertests umfassen alle Testaktivitäten, die vom Entwicklerteam während der Entwicklung durchgeführt werden.

- Sind Defekt Tests

- Modultests – Unit Testing
 - Objekte und Methoden
 - Isoliert testen (einzelne Methoden, Klassen, zusammengesetzte Komponenten mit Schnittstellen)
 - Ziel: Alle Eigenschaften werden abgedeckt
 - Möglichst viele Modultests automatisieren
- Komponententests
 - Fokussieren nur auf Schnittstellenfehler
- Systemtests
 - Getestete Komponenten integriert und geprüft auf ihre Korrektheit
 - Sequenzdiagramme helfen für die Auswertung

Freigabetests:

Freigabetests haben den Zweck, ein spezielles Release eines Systems zu testen, das außerhalb der Entwicklungsteams verwendet wird.

- Ziel: Nachweis gebrauchsfähig, Performance, Zuverlässigkeit etc.
- Blackbox helfen für die Auswertung

Benutzertests:

Vom Benutzer selber getestet und der Benutzer selber entscheidet ob die Software passt

- Alphatests: Benutzer ist ein Teil des Entwicklerteams
- Betatests: Benutzer bekommt ein Vor-Release von der Software um zu testen.
- Abnahmetests: **Mehrere** Benutzer testen und entscheiden ob das System in den Markt kommt.



ABNAHME UND EINFÜHRUNG

---- nicht prüfungsrelevant ----



BETRIEB UND WARTUNG

--- nicht prüfungsrelevant ---

Vorgehensmodelle:

Wichtige Vorgehensmodelle:

- Wasserfallmodell
 - Ein nicht-iterativer Prozess der kontinuierlich (wie ein Wasserfall) über alle Phasen der Entwicklung fließt. Von Analyse Phase, Design Phase, Implementierung Phase, Test Phase zu Abnahme Phase.
 - In der Realität nicht immer so.
- V-Modell
 - Die Entwicklung ist in Phasen eingeteilt. Je zunehmender der Fortschritt desto mehr gehen wir ins Detail.
- Scrum

In kurzen Zyklen releasefähige Software auszuliefern

- Rollen:
 - Product Owner
 - Verantwortlich für die Arbeit des Entwicklungsteams und der Instandsetzung des Produkts. -> Produktmanager und Projektleiter
 - Scrum Master
 - Verantwortlich für das Verständnis und die Durchführung vom Scrum. Die Mitglieder müssen die Regeln einhalten.
 - Entwicklungsteam
 - Profis, die das Produkt erstellen und bei jeden Sprint ein potentiell fertiges Teil übergeben müssen. -> Softwareentwickler
 - Customer Manager
 - Auftraggeber
 - Finanzierung des Projekts
 - User
 - Wichtige Informationsquelle für das Scrum Team
- Meetings
 - Sprint Planning
 - Planung des kommenden Sprint
 - Daily Scrum
 - Täglich einen Feedback geben über seine Tätigkeit
 - Sprint Review
 - Zum Schluss eines Sprint Stakeholder und Scrum Team überprüfen miteinander
 - Retrospektive
 - Selbstreflektion vom Scrum Team um sich zu verbessern für den kommenden Sprint
 - Estimation Meeting
 - Schätzung des Product Backlogs
- Artefakte
 - Product Backlog
 - Geordnete Liste von allen Sachen die mit dem Produkt enthalten sein kann.
 - Sprint Backlog
 - Menge der Sprints ausgewählten Product Backlog-Einträge