Software Entwicklung 1

Asha Schwegler

1. April 2022

Inhaltsverzeichnis

1 Software Engineering

- Herstellung / Entwicklung von Software
- Organisation und Modellierung (Zugehörigen Datenstrukturen, Bettrieb von Softwaresystemen)
- Strukturiertes Projektplan f. Entwicklung
- Unterteilung Entwicklungsprozess
  - Schritte (überschaubar, zeitlich und inhaltlich begrenzt)
  - Phasen
  - Meilensteine
- Disziplinen während Entwicklungsprozess sind verzahnt.

Disziplinen

- **Kerndisziplinen**
  - Anforderungsanalyse
  - Softwarearchitekur und Design
  - Implementierung / Test
  - Softwareverteilung
  - Softwareeinführung
  - Wartung / Pflege
- **Unterstützungsdisziplinen**
  - Projektmanagement
  - Konfigurationsmanagemdnt
  - Risikomanagement

2 Prozess und Prozess-Modell

- Prozess
  - Beschreibung Aktivitäten, Rollen und Artefakte(Informationen)
  - Software-Entwicklung und Wartung
- Prozessmodell
  - Präskriptives Modell (Vorgehensmodell und Organisationsstrukturen)
  - Planung und Lenkung
  - Unified Process, V-Modell, Scrum,...

2.1 Vorgehensmodelle

- Code and Fix
- Wasserfallmodell
- Iterative und inkrementelle Modelle

Code and Fix

- Definition
  - Codierung / Korrektur im Wechsel mit Ad-hoc Tests
- Vorteile
  - Schnell vorankommen
  - Schnelle Ergebnisse
  - Einfache Tätigkeiten (Codieren, Testen, Fixen)
- Nachteile
  - Schlecht planbar und keine Unterstützung im Team
  - Aufwand hoch für Korrekturen
  - Sclecht wartbare Software

Wassefallmodell

- Definition
  - Folge von Aktivitäten/Phasen, gekoppelt durch Teilergebnisse (Dokumente). Reihenfolge ist fest definiert.
- Vorteile
  - hohe Planbarkeit
  - Klare Aufteilung der SWE (Analyse, Design, Test,...)
- Nachteile
  - Schlechtes Risikomanagement (Lösungskonzept nur auf Papier validiert)
  - Anforderungen zu Beginn nie alle bekannt

Iterativ-inkrementelle Modelle

- Definition
  - Geplante und kontrollierte Iterationen inkrementell entwickelt
- Vorteile
  - Flexibles Modell bei unklaren Anforderungen
  - Gutes Risikomanagement (Mitarbeiter und Technologie)
  - Frühe Einsetzbarkeit der Software und Feedback
- Nachteile
  - Upfront Planbarkeit hat Grenzen (Funktionalität, Zeit und Kosten)
  - Braucht Involvierung und Steuerung durch den Kunden über ganze Projektdauer

2.2 Agile SWE

- Basiert auf iterativ-inkrementellen Prozessmodell
- Fokussiert auf gut dokumentierten und getesteten Code statt auf ausführlicher Dokumentation
- Sammlung von Ideen SWE Prozess flexibler und schlanker zu machern
- Adressiert bekannten Probleme bei klassischen Software-Prozessmodellen

Strategie

- Definierte Prozesskontrolle
  - Planung am Anfang, Prozess gesteuert und überwacht
  - Geeignet für gut planbare Problemstellungen
  - Strategie: Steuerung
- Empirische Prozesskontrolle (Agil)
  - Nur Grobplanung am Anfang
  - Prozess fortlaufend überwacht
  - Rollende Planung
  - Geeignet für komplexe Problemstellungen
  - Strategie: Regelung, Deming-Cycle (Plan-Do-Check-Act)

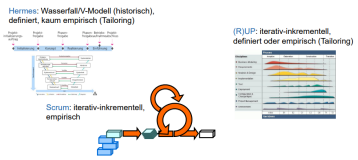

Abbildung 1: CharaktersierungProzessmodellen

3 Modellierung

Modell: Abbild oder Vorbild für ein zu schaffendes Gebilde.
Original: Abgebildete oder zu schaffende Gebilde

- **Modellierung**
  - Software selbst ein Modell
  - Anforderungen = Modelle der Lösung
  - Tesfälle = Modelle Korrektes Funktionieren des Codes

3.1 UML

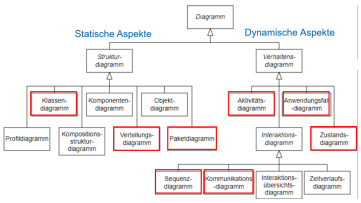

Abbildung 2: Guetereinteilung.

3.1.1 Gebrauch der UML

- UML as a sketch
  - Informell, unvollständig
  - Bevorzugt von agile Community
- UML as blueprint

  - Detallierte Analyse und Design-Diagramme für Code
  - Forward - und Reverse-Engineering
- UML as a Programming Language
  - Komplette, ausführbare Spezifikation eines Software-Systems in UML
  - MDA-Tools zur Modellierung und Generierung

4 Wesentliche Artefakte

- Anfoderungsanalyse
  - Funktionale Anforderungen mit Use Cases
  - Qualitätsanforderungen und Randbedingungen
- Design
  - Softwarearchitektur
  - Use Case Realisierung (Statische und dynamische Modelle)
- Implementation
  - Quellcode (inkl.Javadoc)
- Testing
  - Unit-Tests
  - Integrations- und Systemtests

4.1 Überblick Anforderungsanalyse

- **User Research**
  - Personas
  - Szenarien
  - Contextual Inquiry
- Sketching und Prototyping
- **Use Cases**
  - Ableiten und Modellieren
  - Detaillierung (UML-Use-Case-Diagramm, Use-Case-Spezifikation, UI-Sketching)
- **Qualitätsanforderungen, Randbedingungen** erheben
- **Domänenmodell**
  - Konzeptuelles UML-Klassendiagramm
- **objektorientierten Analyse(OOA)**
  - Objekte/Konzepte in dem Problembereich zu finden und zu beschreiben

4.2 Überblick Design

- **Softwarearfchitektur**
  - UML-Paketdiagramm
  - UML-Deploymentdiagramm
- **Use-Case-Realisierung und Klassendesign**
  - UML-Klassendiagramm
  - UML-Sequenzdiagramm
  - UML-Kommunikationsdiagramm
  - UML-Zustandsdiagramm
  - UML-Aktivitätsdiagramm
- Entwurf **Design Patterns**
- **Objektorientierten Design (OOD)**
  - Geeignete Softwareobjekte und ihr Zusammenwirken definieren

4.3 Überblick Implementation

- **Code**
  - Umsezung Design in entspr. OOP-Sprache
- **Refactoring**
  - Code smells aufdecken und verbessern
- **laufende Dokumentierung**
  - Quellcode

4.4 Überblick Testing

- **Unit-Tests**
  - Laufendes Design und Implementierung
- **Teststufen Integration und System**
  - Planung, Design und Durchführung
- **Dokumentation**
  - Testkonzept und Test

5 Usability und User Experience (UX)

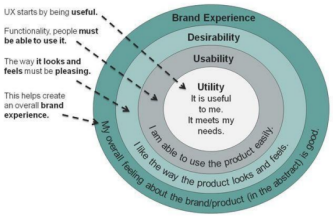

Abbildung 3: Usability.

5.1 Usability

Definition: Effektivität, Effizienz, Zufriedenheit -¿ Ziele erreichen im spezifischen Kontext

- **4 wichtige Aspekte**
  - Benutzer
  - Seine Ziele/Aufgaben
  - Sein Kontext
  - Softwaresystem (inkl.UI)

5.2 Usability Engineering

Ziel: Software entwickeln, die drei Anfroderungen erfüllt

- **Drei Anforderungen:**
  - Effektivität
    - Aufgaben vollständig erfüllen
    - Genauigkeit
  - Effizienz
    - Mit minimalem Aufwand (Mental, Physisch, Zeit)
  - Zufriedenheit
    - **Minimum:** nicht verärgert
    - **Normal:** Zufrieden
    - **Optimal:** Erfreut

5.3 Usability Anforderungen

- **7 Anforderungen:**
  - Aufgabengemessenheit
  - Lernförderlichkeit
  - Individualisierbarkeit
  - Erwartungskonformität
  - Selbstbeschreibungsfähigkeit
  - Steuerbarkeit
  - Fehlertoleranz

5.3.1 Aufgabenangemessenheit

- Minimale Anz. Schritte f. Aufgabe
- Nur wichtige Informationen
- Kontextabhängige Hilfe
- Minimale Anz. Benutzereingaben
  - Jede Eingabe nur 1x
  - Standardwerte
  - Liste vordefinierter Werte (z.B Länder)
  - Ableitbare Eingaben vorschlagen

5.3.2 Selbstbeschreibungsfähigkeit

- Benutzer ausreichend informieren
  - Wo er ist
  - Was er tun soll / kann
  - Wie er es tun soll (Formate, Werte)
- Begriffe des Benutzers verwenden (Labels, Fehlermeldungen)
- Affordanzen

5.3.3 Kontrolle

- Mit Interaktion Benutzer steuern
  - Initiative, Tempo
  - Dialogfluss
  - Darstellungsformate
  - Inputmodalität (Maus, Tastatur, Touch, Sprache)
- Benutzeraktionen rückgängig machen können
- Benutzeraktionen jeder Zeit abbrechen können

5.3.4 Erwartungskonformität

- Bezüglich
  - Design
  - Interaktion
  - Struktur
  - Komplexität
  - Funktionalität
- Konsistenz
  - Terminologie
  - Verhalten (Reihenfolge Aktionen, Änderungen)
  - Informationsdarstellung (Platzierung, Wortwahl)

5.3.5 Fehlertoleranz

- Benutzerfehler vermeiden
  - Klar kommunizieren (Erwarteter Input, Funktionen aktiv resp. sinnvoll)
- Benutzereingaben vor Aktion überprüfen
- Nicht unbedingt beim Tippen
- Benutzer helfen
  - Fehler zu erkennen
  - Ursache zu verstehen
  - Aus Fehlerzustand zu kommen
- Einfache Korrektur
- Kein Datenverlust

5.3.6 Individualisierbarkeit

- System anpassbar sein:
  - Know-How
  - Sprache
  - Kultur
  - Benutzer mit Einschränkungen

5.3.7 Lernförderlichkeit

- Informationen über unterliegende Konzepte und Regeln anbieten
  - Um mentales Modell anzugleichen
  - Nur auf Verlangen des Users
  - eifache Tasks ohne Vorkentnisse
  - komplexere Konzepte bei der Verwendung zu erlernen

6 User-Centered Design (UCD)

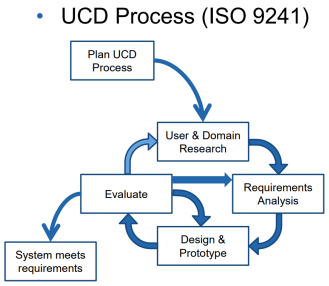

Abbildung 4: UCDProcess

6.1 User & Domain Research

- **Ziele bez. Benutzer:**
  - Wer ist Benutzer
  - Was ist die Arbeit (Aufgaben, Ziele)
  - Wie sieht Arbeitsumgebung aus
  - Was wird gebraucht um Ziele zu erreichen
- Welche Sprache, Begriffe
- Normen (organisatorisch, kulturell, sozial)
- Pain Points (Brüche, Workarounds)
- Für mobile Apps:
  - Nutzungskontext
    - Wo wird App benutzt (Umgebung)
    - Wann wird App benutzt (Tageszeit, involvierte Personen, Randbedingungen)
    - Warum wird App benutzt (Nutzen, Motivation, Trigger)
- **Ziele bez. Domäne:**
  - Buisiness der Firma verstehen
  - Domäne verstehen (Sprache, Wichtigste Konzepte, Prozesse)

6.1.1 GUI Design Process

Methoden User & Domain Research

- Contextual Inquiry
- Interviews
- Beobachtung
- Fokusgruppen
- Umfragen
- Nutzungsauswertung
- Desktop Research (Dokumentenstudium, Mitbewerber)

6.1.2 Wichtige Artekfakte

- Personas
- Usage-Szenarien
  - Kurze Geschichte
    - Usage Szenarien
      - aktuelle Situation
      - in User and domain research verwendet
    - Kontextszenarien
      - Zukünftige gewünschte Situation
      - in Anforderungsanalyse verwendet
- Mentales Modell
- Domänenmodell
- Stakeholder Map

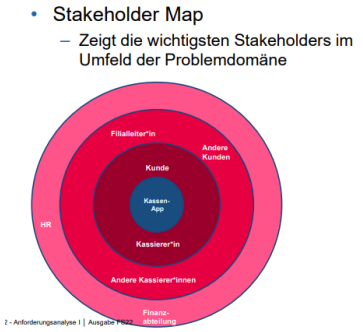

Abbildung 5: Stakeholdermap

- Service Blueprint/Geschäftsprozessmodell

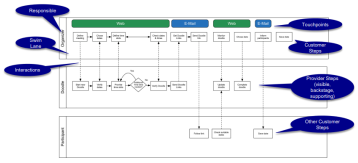

Abbildung 6: Blueprint

6.2 Anforderungsanalyse

Ziel:

- Ausgehend von den Resultaten des UCD -¿ User-Anforderungen ableiten:
  - Funktionale Abläufe, Interaktionen
    - **Kontextszenarien**
    - **Storyboards**
    - **UI-Skizzen**
    - **Use cases**
  - Konzepte, Beziehungen, Quantitäten
    - **Kontextszenarien**
    - **FURPS-Modell (Functionality, Usability, Reliability, Performance, Supportablility**

6.2.1 Use Cases

- Akteur
  - Primärakteur
  - Unterstützender Akteur
  - Offstage-Akteur
- Keine Kann-Formulierungen
- 3 Ausprägungen:
  - Kurz
    - Titel + 1 Absatz (Standardablauf)
  - Informell
    - Titel + Informelle Beschreibung (können mehrere Absätze sein, beschreibt auch Varianten)
  - Vollständig
    - Titel + alle Schritte und alle Varianten im Detail
    - UC-Name
    - Umfang
    - Ebene
    - Primärakteur
    - Stakeholders und Interessen
    - Vorbedingungen
    - Erfolgsgarantie/Nachbedingungen
    - Standardablauf
    - Erweiterungen
    - Spezielle Anforderungen
    - Liste der Technik und Datavariationen
    - Häfigkeit des Auftretens
    - Verschiedenes
  - Notation = Nomen + Verb

Use-Case-Diagramm

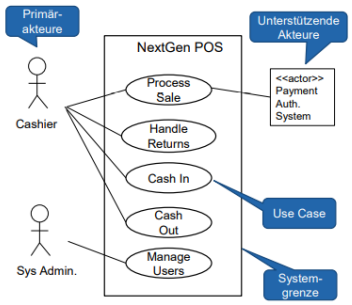

Abbildung 7: UseCaseDiagramm

Zusätzliche Beziehungen im Use Case Diagramm

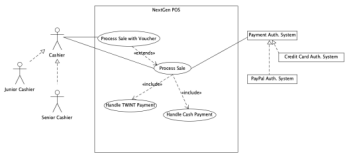

Abbildung 8: Zusätzliche Beziehungen UseCaseDiagramm

6.2.2 UML Sequenzdiagramm (SSD)

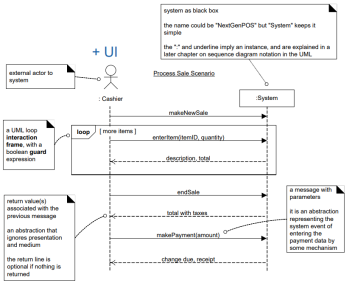

Abbildung 9: Zusätzliche Beziehungen Systemsequenzdiagramm

SSD zwischen zwei Systemen

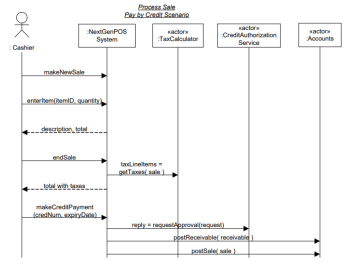

Abbildung 10: Zusätzliche Beziehungen Systemsequenzdiagramm zwischen zwei Systeme

System Operation

- Formal wie ein Methodenaufruf

- Treffender Name
- Evt. mit Parametern
- Durchzogener Pfeil für Methodenaufruf
- Rückgabewert (Kann fehlen falls unwichtig, Gestrichelte Linie weil Update des UI)
- Definieren API des Systems

6.2.3 Operation Contract

Definition: Spezifiziert (System)Operation

- Name plus Parameterliste
- Vorbedingung (Was muss zwingend erfüllt sein damit Systemoperation aufgerufen werden kann)
- Nachbedingung
  - Was hat sich alles geändert nach Ausführung (Erstellte / gelöschte Instanzen, Assoziationen, geänderte Attribute)
  - basierend auf Domänenmodell

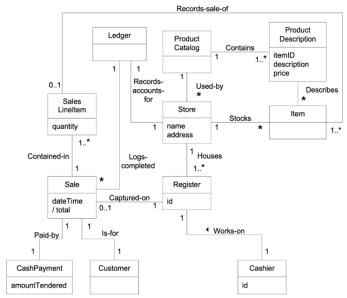

Abbildung 11: OperationContract

6.2.4 Zusätzliche Anforderungen

- Funktionale
- Nicht-Funktionale

Formulierung

- Anforderungstatements
  - Als Anforderung formuliert
  - messbar/verifizierbar
- So wenig wie nötig
  - Nur diejenige, die begründet gefordert werden
  - Keine ersten Lösungsideen als Forderungen

Checkliste FURPS+

- **F**unctionality
  - Features, Fähigkeiten, Sicherheit
- **U**sability
  - Usability Anforderungen (Kap.5.3)
  - Accessibility
- **R**eliability
  - Fehlerrate, Wiederanlauffähigkeit, Vorhersagbarkeit, Datensicherung
- **P**erformance
  - Reaktionszeiten, Durchsatz, Genauigkeit, Verfügbarkeit, Ressourceneinsatz
- **S**upportability
  - Anpassungsfähigkeit, Wartbarkeit, Internationalisierung, Konfigurierbarkeit
- +
  - Implementation (HW,BS,Sprachen, Tests...)
  - Interface
  - Operations
  - Packaging
  - Legal

Glossar

- Einfaches Glossar
  - Begriffe im Projekt und SW-Produkt
  - beliebige Elemente
- Data Dictionary
  - Zusätzliche Datenformate, Wertebereiche, Validierungsregeln

7 Domänenmodellierung

Definition: Vereinfachtes UML-Klassendiagramm.

Aufbau: Fachliche Begriffe mit ihren Attributen, setzt Begriffe in Beziehung zueinander.
Geht nur um die Problemstellung und das Fachgebiet.

7.1 Domänenmodell als vereinfachtes UML Klassendiagramm

Konzepte = Klassen
Eigenschaften = Attributen (Typenangabe entfällt)
Assoziatonen = Beziehungen zwischen Konzepte mit Multiplizitäten an beiden Enden.

Nur wenn es einen guten Grund gibt:

- **Aggregation** = Keine echte Semantik, als Abkürzung für "hat".
- **Komposition** = z.B wenn Produktkatalog gelöscht wird, dann auch die darin enthaltenen Produktbeschreibungen. Abkürzung "bietet an".

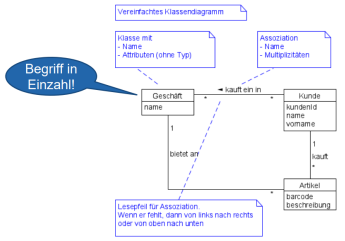

Abbildung 12: DM1

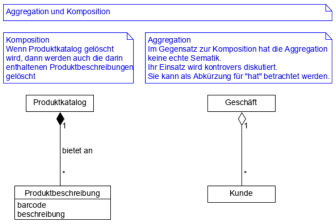

Abbildung 13: DM2

Abbildung 14: DM3

7.2 Vorgehen

1. Konzepte identifizieren
   (a) Fachwissen und Erfahrung verwenden
   (b) Substantive aus Anwendungsfällen
   (c) Kategorienliste verwenden
2. Attributen
   (a) Fachwissen verwenden
3. Konzepte in Verbindung zueinander setzen
   (a) Fachwissen verwenden
   (b) Kategorienliste verwenden
4. Auftraggeben und/oder Fachexperten beiziehen
5. Vorgehensweise eines Kartografen

7.2.1 Kategorienliste

Kategorie	Mögliche Konzepte für DM
Geschäftstransaktionen	
• Transaktionen als Ganzes	Sale
• Transaktionsposition	SalesLineItem
• Produkt, das damit verbunden ist	Item
• Wo wird Transaktion registriert?	Register
• Rollen von beteiligten Personen	Cashier
• Ort der Transaktion	Store
• Beschreibung von Dingen	ProductDescription
• Ereignisse mit Ort/Zeit	Sale

Abbildung 15: Kategorienliste1

Kategorie	Mögliche Konzepte für DM
Physische Objekte	Register
Kataloge	ProductCatalog
Container von Dingen	Store
Dinge in den Containern	Item
Andere beteiligte Systeme	CreditAuthorizationSystem
Rollen von beteiligten Personen	Cashier
Artefakte (Pläne, Finanzen, Arbeit, Verträge, ...)	Receipt
Zahlungsinstrumente	Cash, Credit Card

Abbildung 16: Kategorienliste2

Kategorie	Mögliche Assoziation für DM
Transaktion	Payment - Sale
• Position	SalesLineItem - Sale
• Produkt	Item - SalesLineItem
• Rolle	Customer - Payment
Teil zum Ganzen	Register - Store
Beschreibung zum Gegenstand	ProductDescription - Item
Protokoll zum Gegenstand	Sale - Register
Verwendung	Cashier - Register

Abbildung 17: Kategorienliste3

7.3 Datentypen von Attributen

- Wenn nötig: eigene Datentypen als **Konzepte**
- Dann definieren wenn:
  – Typ aus mehreren Abschnitten (wie Tel.Nr)
  – Operationen darauf sind möglich (Validierung Kreditkartennummer)
  – Hat selber eigene Attribute (Verkaufspreis mit Anfangs & Enddatum)
  – Verknüpft mit Einheit (Preis mit Währung)

7.3.1 Anti-Pattern

Assoziationen statt Atrribute

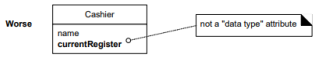

Abbildung 18: AntiPattern Bad

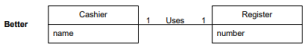

Abbildung 19: AntiPattern Good

7.4 Vorgehensweise eines Kartografen

- Vorhandene Begriffe oder Wissen einsetzen (Gebiete besuchen, Bewohner nach Begriffen befragen)
- Unwichtiges weglassen
- Nichts hinzufügen, was es (noch) nicht gibt
  - Ausnahme: System, das enwickelt wird, kann eingetragen werden
- Nur analysieren, (noch) keine Lösungen entwerfen

Keine Software Klassen im Domänenmodell :

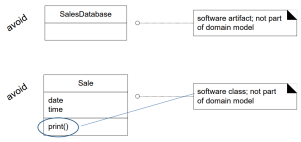

Abbildung 20: AntiPatternSoftware

7.5 Domänenmodell vollständig Beispiel

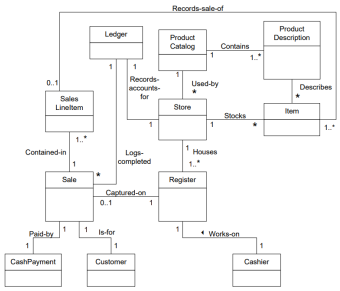

Abbildung 21: BeispielDomänenmodell

7.6 Analysemuster

- **Beschreibungsklassen**
  - Item = Physischer Gegenstand oder Dienstleistung
  - Mehrere Artikel desselben Typs
  - Attribute (description, price, serial number, itemID)

Abbildung 22: Beschreibungsklassen

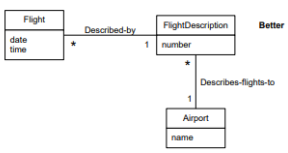

Abbildung 23: BeschreibungsklasseFlug

- **Generalisierung / Spezialisierung**
  - Spezialisierung als ïs a"Beziehung zu

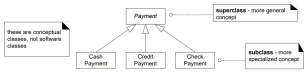

Abbildung 24: GeneralisierungSpezialisierung

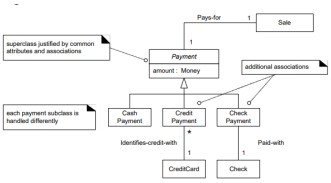

Abbildung 25: GeneralisierungSpezialisierungAttribute

- **Komposition**
- **Zustände**
  - Eigene Hierarchie für Zustände definieren:

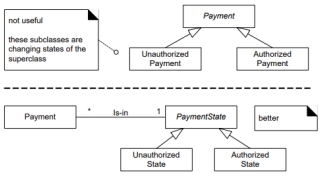

Abbildung 26: Zustände

- **Rollen**
  - Dasselbe Konzept kann unterschiedliche Rollen einnehmen:

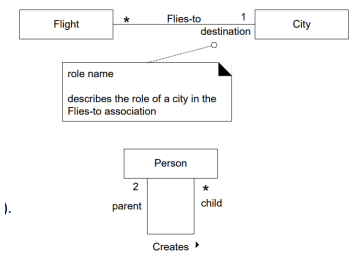

Abbildung 27: DMRolleName

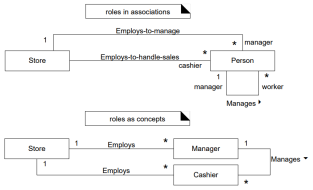

Abbildung 28: DMRolleKonzept

- **Assoziationsklasse**
  - Wenn Assoziationen eigene Attribute haben (MerchantID für Kreditkarte Geschäft¡-¿AuthorizationService):

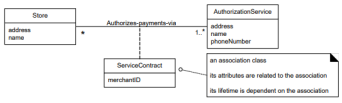

Abbildung 29: Assoziationsklasse

- **Einheiten**

– Manchmal sinnvoll explizit als Konzept zu modellieren :

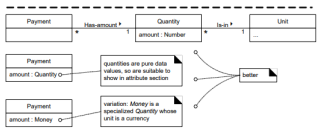

Abbildung 30: Einheiten

- **Zeitintervalle**
  - Gültigkeitsintervall für sich ändernede Attribute :

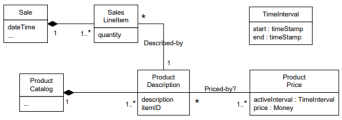

Abbildung 31: Zeitintervall

8 Softwarearchtiketur und Design I

- Gesamtheit der wichtigsten Entwurfs-Enscheidungen.
  - Programmiersprachen, Plattformen
  - Aufteilung: Teilsysteme, Bausteine, Schnittstellen
  - Veratnwortlichkeiten und Abhängigkeiten der Teilsysteme
  - Basis-Technologie oder Frameworks (z.B Java EE)
  - Besondere Massnahmen um Anforderungen zu erfüllen
- Grundlagen
  - Anforderungen (vorallem nicht-funktionale)
  - Systemkontext mit Schnittstellen
- Top Level View (das grosse Ganze)

8.0.1 Übersicht Buisiness Analyse vs Architektur vs Entwicklung

Business Analyse

- Domänenmodell
- Kontext Diagramm
- Requirements
  - Liste Stakeholder
  - Vision
  - Funktionale Anforderungen:
    - Use Cases / User Stories
  - Nicht funktionale Anforderungen:
    - Supplementary Specification
  - Randbedingungen
  - Glossar

Architektur

- Logische Architektur

Entwicklung

- Use Case / User Story Realisierung
- Anwendung GRASP
- DCD - Design-Klassen-Diagramm
- Interaktionsdiagramme
- Programmierung
- Erstellen der Unit-/Integrations-Tests

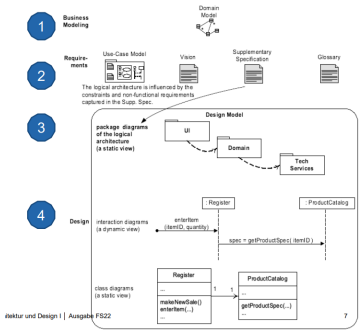

Abbildung 32: Übersicht

8.0.2 Wie entstehen Architekturen

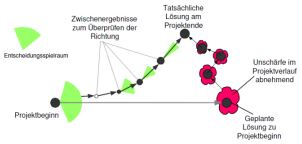

Abbildung 33: ArchitekturEntstehung

Architektur aus Anforderungen ableiten

- Muss heutige und zukünftige Anfroderungen erfüllen können
- Aufgabe Architekturanalyse
  - Analyse funktionale und nicht funktionale Anforderungen und deren Konzsequenzen
  - Berücksichtigung Randbedingung und zukünftige Veränderungen
  - Qualität, Stabilität der Anforderungen prüfen
    - Lücken in Anforderungen aufdecken

Twin peak Model

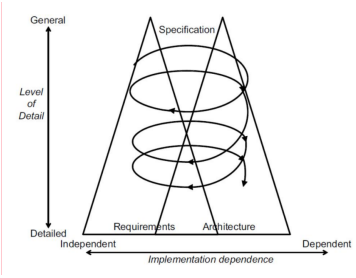

Abbildung 34: TwinPeak

Entwurfsentscheidungen

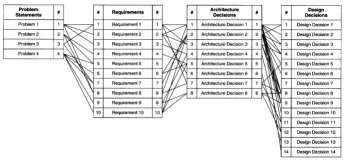

Abbildung 35: EntwurfsEntscheidungen

Nichtfunktionale Anforderungen

Abbildung 36: NichtfunktionaleAnforderungen

8.1 Modulkonzept

- Modul(Baustein, Komponente):
  - Autarkes Teilsystem
  - Klare, minimale Schnittstelle gegen Aussen
  - Software-Modul enthält alle Funktionen und Datenstrukturen
  - Modul: Paket, Programmierkonstrukt, Library, Komponente, Service
- Konzept in allen Ingenierdisziplinen angwendet

Schnitstellen Kapselung und Austauschbarkeit

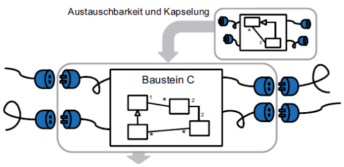

Abbildung 37: Schnittstellen

Prinzip modularen Strukur

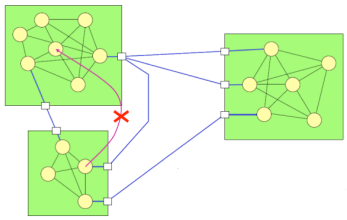

Abbildung 38: ModularenStruktur

8.1.1 Messung der Güte einer Modularisierung

- Kohäsion (Stärke inneren Zusammenhangs)
  - schlecht: zufällig, zeitlich
  - gut: funktional, objektbezogen
  - je höher Kohäsion innerhalb Modul, desto besser die Modularisierung
- Kopplung (Abhängigkeit zwischen 2 Modulen)
  - schlecht: Globale Kopplung
  - akzeptabel: Datenbereichskopplung (Referenzen auf gemeinsame Daten)
  - gut: Datenkopplung (alle Daten werden beim Aufruf der Schnittstelle übergeben)
  - Je geringer die wechselseitige Kopplung desto besser die Modularisierung

Clean Architecture

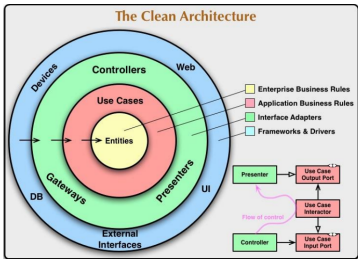

Abbildung 39: CleanArchitecture

- Unabhängikeit:
  - von Framework
  - voneinander getestet werden
  - von UI
  - von DB

8.2 Architektur Beschreiben

Aufgeteilt in Views:

- Logical View
  - Funktionalität gegen aussen
  - Aspekte:
    - Schichten
    - Subsysteme
    - Pakete
    - Frameworks
    - Klassen
    - Interfaces
  - UML:
    - Systemsequenzdiagramme
    - Interaktionsdiagramme
    - Klassendiagramm

        - Zustandiagramme

- Process View
    - Wo und wie im System
    - Aspekte:
        - Prozesse
        - Threads
        - Wie Anforderungen erreicht
    - UML:
        - Klassendiagramme
        - Interaktionsdiagramme
        - Aktiviitätsdiagramme

- Development View (Implementation View):
    - Wie Struktur umgesetzt
    - Aspekte:
        - Source Code
        - Executables
        - Artefakte
    - UML:
        - Paketdiagramme
        - Komponentendiagramme

- Physical View (Deployment View)
    - Auf welcher Infrastruktur wird System ausgeliefert /betrieben
    - Aspekte:
        - Prozessknoten
        - Netzwerke
        - Protokolle
    - UML:
        - Deployment Diagram

- +1 View: Scenarios (Use Cases)
    - Wichtigste Use Cases und ihre nicht funktionale Anforderungen? Wie umgesetzt?
    - Aspekte:
        - Architektonisch wichtige UCs
        - deren nichtfunktionale Anforderungen
        - deren Implementation
    - UML:
        - UC-Diagramm
        - Systemsequenzdiagramme
        - UC-Realisierungen

- Daten-Sicht

- Sicherheit

Logische Architektur vs Physikalische Architeckur

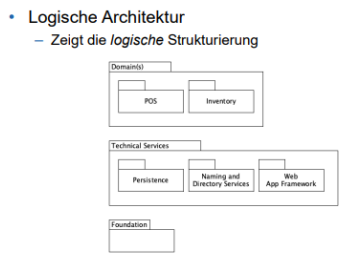

Abbildung 40: LogischeArchitektur

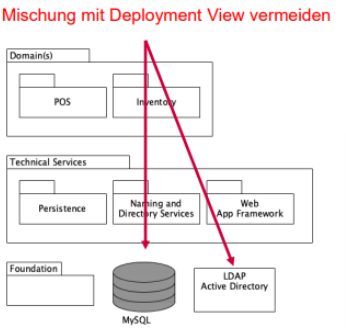

Abbildung 41: VermeidungArch

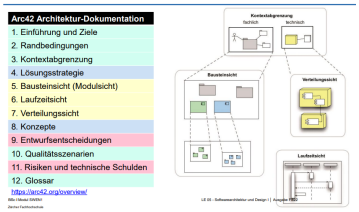

Abbildung 42: Arc42

8.3 UML-Paketdiagramme

- Mittel, zum Teilsysteme zu definieren
- Mittel zur Gruppierung von Elementen
- Paket enthält Klassen und andere Pakete
- Abhängigkeit zwischen Paketen

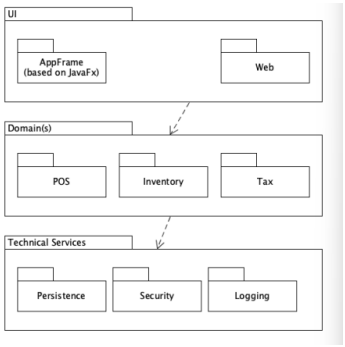

Abbildung 43: PaketDiagramm1

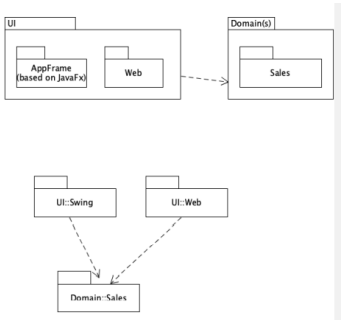

Abbildung 44: PaketDiagramm2

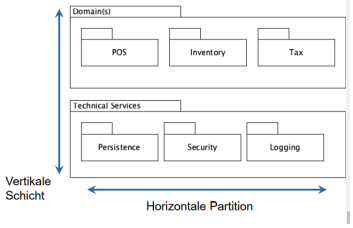

Abbildung 45: PaketDiagramm3

8.4 Verteilungsdiagramm

- Darstellung Verteilung von Komponenten auf Rechenknoten mit Abhängigkeiten, Schnittstellen und Verbindungen
- Statische Modellierung

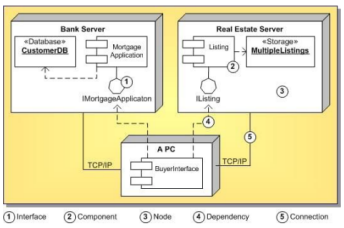

Abbildung 46: VerteilungsDiagramm

8.5 Ausgewählte Architekturpatterns

Pattern	**Beschreibung**
Layered Pattern	Strukturierung eins Programms in Schichten
Client-Server Pattern	Server stellt Services für mehrere Clients zur Verfügung
Master-Slave Pattern	Master verteilt Arbeit auf mehrere Slaves
Pipe-Filter Pattern	Verarbeitung einses Datenstroms (filtern, zuordnen, speichern)
Broker Pattern	Meldungsvermittler zwischen verschiedenen Endpunkten
Event-Bus Pattern	Datenquellen publizieren Meldungen an einen Kanal auf dem Event-Bus. Datensenken abonnieren einen bestimmten Kanal
MVC Pattern	Ineraktive Anwendung in 3 Komponenten aufgeteilt: -Model -View - Informationsanzeige -Controller - Verarbeitung Benutzereingabe

8.5.1 Layered Pattern

- Je weiter unten, desto allgemeiner
- Je höher, desto anwendungs-spezifischer
- Zuoberst ist das Benutzerinterface

Kopplung von oben nach unten NIE von unten nach oben.

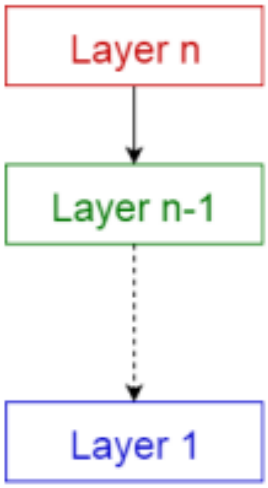

Abbildung 47: LayeredPattern1

Anrufszenarien

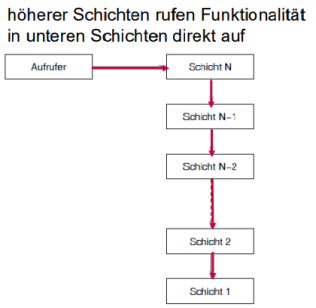

Abbildung 48: AnrufSzenarienH

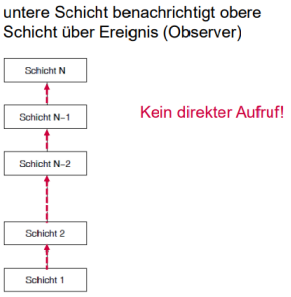

Abbildung 49: AnrufSzenarienObserver

- UI
  - Presentation, Windows, Dialoge, Reports, WEB, Mobile
- Application
  - Requests von UI Layer, Workflow, Sessions
- Domain
  - Requests von Application Layer, Domain Rules, Services
- Business Infrastructure
  - Low level business Services (z.B CurrencyConverter)
- Technical Services
  - Persistence, Security, Logging
- Foundation
  - Datenstrukturen, Threads, Dateien, Netwerk IO

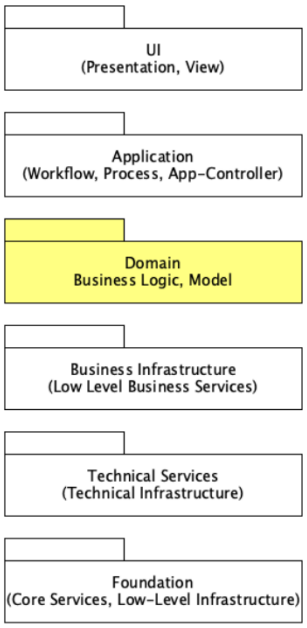

Abbildung 50: LayeredPattern2

8.5.2 Client-Server

- 1 Server und mehrere Clients
- 1 Server stellt einen oder mehrere Services zur Verügung
- Client macht Request zum Server
- Server sendet Response zurück

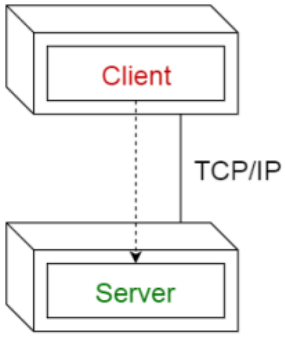

Abbildung 51: ServerClient

8.5.3 Master-Slave Pattern

- Master verteilt die Aufgaben auf mehrere Slaves
- Slaves führen Berechnungen aus und senden Ergebnis zum Master
- Maseter berechnet Endergebnis

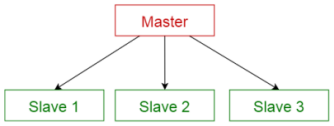

Abbildung 52: MasterSlave

8.5.4 Pipe-Filter Pattern

- Verarbeitung von Datenströmen (Linux Pipe, RxJS Observable Streams, Java Streams,...)
- Verarbeitungsschritt durch Operator wie Filter, Maper, etc. umgesetzt

Abbildung 53: PipeFilter

8.5.5 Broker Pattern

- verteilte Systeme mit entkoppelten Subsysteme zu koordinieren
- Broker(Vermittler) ermittelt Kommunikation zwischen einem Client und dem entspr. Subsystem
- Bsp.: Message Broker

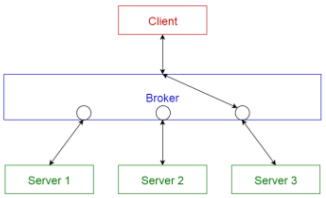

Abbildung 54: BrokerPattern

8.5.6 Event-Bus Pattern

- 4 Hauptkomponenten:
  1. EventSource
  2. Eventlistener
  3. Channel
  4. Event Bus
- Event Sources publizieren Meldungen zu einem bestimmten Kanal auf dem Event Bus
- EventListeners:
  - Melden sich für bestimmte Events an
  - werden informiert, sobald entsprechende Meldungen auf dem Kanal befinden

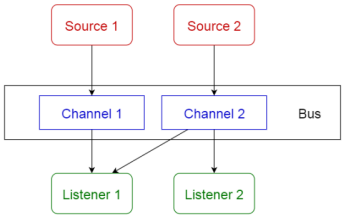

Abbildung 55: EventBus

8.5.7 Model View Controller Pattern

- Interaktive Anwendung in 3 Komponenten:
  - Model: Daten und Logik
  - View: Informationsanzeige
  - Controller: Verarbeitung der Benutzereingabe
- Entkopllung UI und Logik
- Erlaubt Austauschbarkeit des UIs
- Alternativen:
  - MVVM: Model View View Model
  - MVP: Model View Presenter

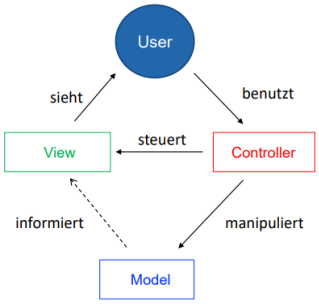

Abbildung 56: MVC