

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

0.0.1 Übersicht Business 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

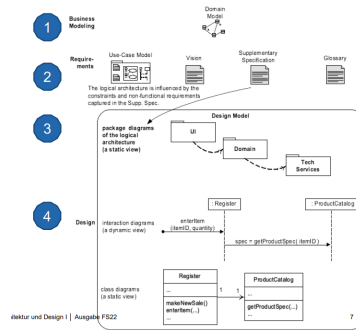


Abbildung 1: Übersicht

0.0.2 Wie entstehen Architekturen

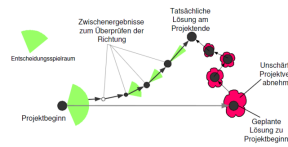


Abbildung 2: ArchitekturEntstehung

Architektur aus Anforderungen ableiten

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

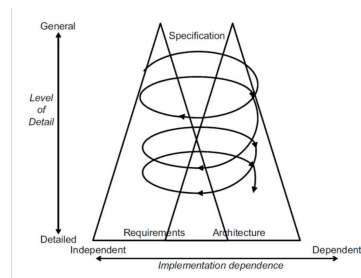


Abbildung 3: TwinPeak

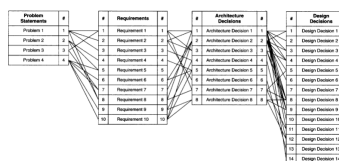


Abbildung 4: EntwurfsEntscheidungen

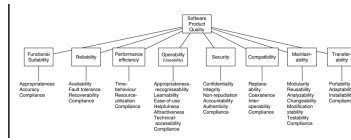


Abbildung 5: Nichtfunktionale Anforderungen

0.1 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 angewendet

orange Schnittstellen Kapselung und Austauschbarkeit

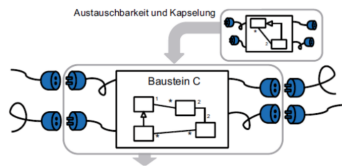


Abbildung 6: Schnittstellen

orange Prinzip modularen Struktur

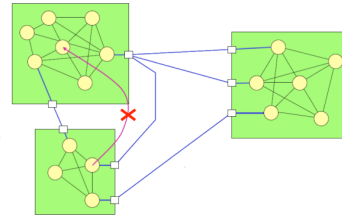


Abbildung 7: Modularen Struktur

0.1.1 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

lean Architecture

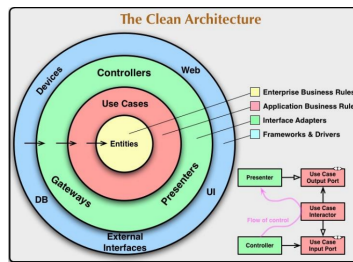


Abbildung 8: CleanArchitecture

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

0.2 Architektur Beschreiben

Aufgeteilt in Views:

- Logical View
 - Funktionalität gegen aussen
 - Aspekte:
 - * Schichten
 - * Subsysteme
 - * Pakete
 - * Frameworks
 - * Klassen
 - * Interfaces
 - UML:
 - * Systemsequenzdiagramme
 - * Interaktionsdiagramme
 - * Klassendiagramm
 - * Zustandsdiagramme
- Process View
 - Wo und wie im System
 - Aspekte:
 - * Prozesse
 - * Threads
 - * Wie Anforderungen erreicht
 - UML:
 - * Klassendiagramme
 - * Interaktionsdiagramme
 - * Aktivitätsdiagramme
- 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

ogische Architektur vs Physikalische Architeckur

- Logische Architektur
 - Zeigt die *logische* Strukturierung

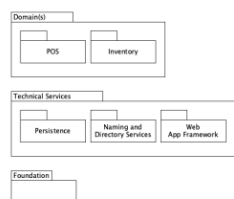


Abbildung 9: LogischeArchitektur

Mischung mit Deployment View vermeiden

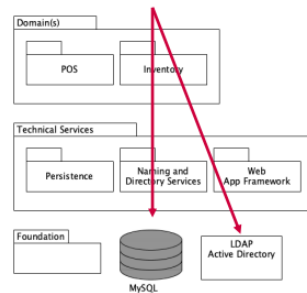


Abbildung 10: VermeidungArch

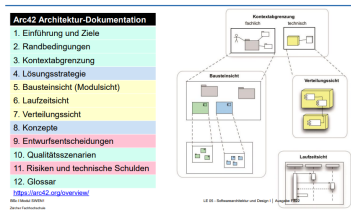


Abbildung 11: Arc42

0.3 UML-Paketdiagramme

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

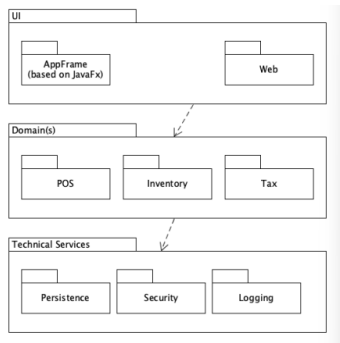


Abbildung 12: PaketDiagramm1

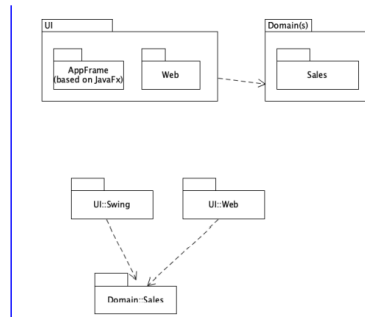


Abbildung 13: PaketDiagramm2

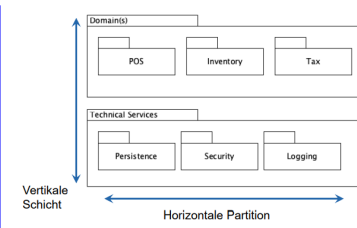


Abbildung 14: PaketDiagramm3

0.4 Verteilungsdiagramm

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

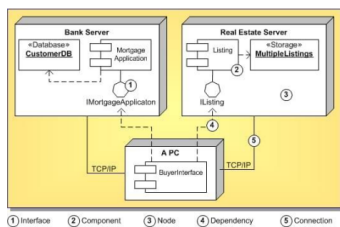


Abbildung 15: VerteilungsDiagramm

0.5 Ausgewählte Architekturpatterns

Pattern	Beschreibung
Layered Pattern	Strukturierung eines 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 eines Datenstroms (filtern, zuordnen, speichern)
Broker Pattern	Meldungsvermittler zwischen verschiedenen Endpunkten
Event-Bus Pattern	Datenquellen publizieren Meldungen an einen Kanal auf dem Event-Bus. Datensinken abonnieren einen bestimmten Kanal
MVC Pattern	Interaktive Anwendung in 3 Komponenten aufgeteilt: -Model -View - Informationsanzeige -Controller - Verarbeitung Benutzereingabe

0.5.1 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.

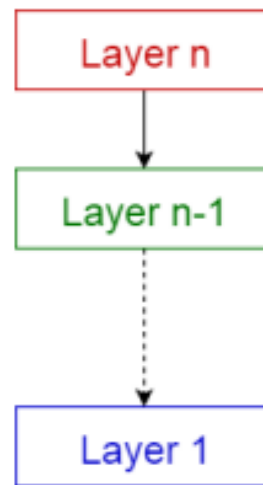


Abbildung 16: LayeredPattern1

höherer Schichten rufen Funktionalität in unteren Schichten direkt auf

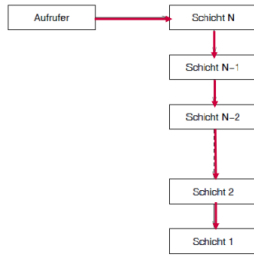


Abbildung 17: AnrufSzenarienH

untere Schicht benachrichtigt obere Schicht über Ereignis (Observer)



Abbildung 18: 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, Network IO

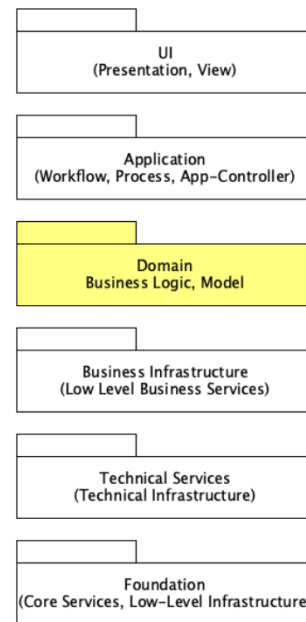


Abbildung 19: LayeredPattern2

0.5.2 Client-Server

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

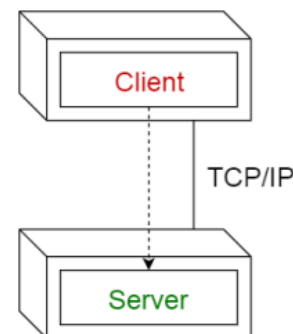


Abbildung 20: ServerClient

0.5.3 Master-Slave Pattern

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

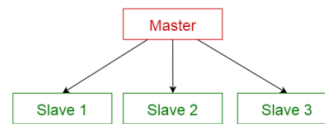


Abbildung 21: MasterSlave

0.5.4 Pipe-Filter Pattern

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



Abbildung 22: PipeFilter

0.5.5 Broker Pattern

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

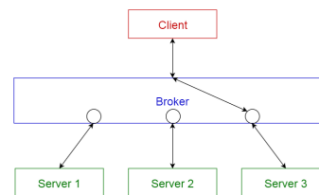


Abbildung 23: BrokerPattern

0.5.6 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

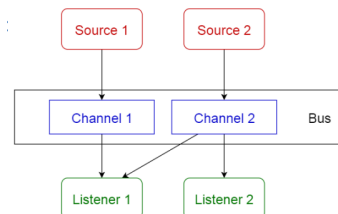


Abbildung 24: EventBus

0.5.7 Model View Controller Pattern

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

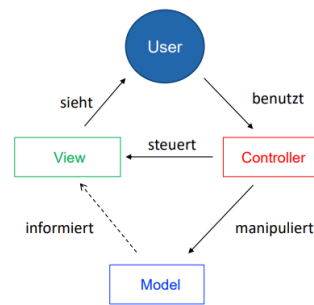


Abbildung 25: MVC