Vorgehensweise/Ideen

# Architektur

Mikroarchitektur

- Auch in unserem Projekt möglich, da man die Anwendungsteile in einzelene Funktionen mit UI unterteilen kann und dann einen Asset Server einführt. Genauer ist das die Vertikale Dekomposition. In den einzelnen Mikros dann wieder MVC als Client Architektur?

# Entwurfsprinzipien

## Information Hiding

* nur notwendige Informationen verfügbar (verhindert unbeabsichtigte Fehler)
* Achtung: konsequentes Information-Hiding kann zu erhöhter Komplexität führen  
  -> Übersichtlichkeit leidet -> richtiges Maß finden

## Entwurf für Veränderung

* für unser Projekt wahrscheinlich relativ wichtig (einzelne Parameter leicht anpassen etc.)

## Abstraktion

* Trennung von Schnittstelle und Implementierung (Interfaces)
* „Design-by-Contract“ (Vor- und Nachbedingungen festlegen und einhalten + Comment)

## Interne Wiederverwendung

* reduziert Redundanz
* allgemeine Schnittstellen
* falls es sich irgendwo anbietet

Haben wir Subsysteme die getrennt werden können?

# Komponenten

* Klassen, Objekte, etc. können in Komponenten gebündelt werden
* White-Box/Black-Box
* viele UML-Diagrammmöglichkeiten verfügbar (siehe 07/Folie 61 ff.)

# Entwurfsmuster

### Grobentwurf

* Aufteilen des Projekts in handelbare Teile
* Welche einzelnen Bausteine?
* Welche Schnittstellen zwischen Bausteinen? (Interface, etc.)

### Feinentwurf

* Datenstrukturen
* Algorithmen

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

# Architekturprinzipien

* verschiedene Diagramme für verschiedenen Einsatzzweck

### Integrität

* gleiche Aufgabenstellung -> gleiche Entwurfsmuster + Umsetzung
* keine speziellen Lösungen

### Selbstorganisation

* oft eine zentrale Struktur die alles steuert -> besser mehrere einzelne Steuerungen

### Sichten/Diagramme eines Architekturmodells („wichtig für Projektarbeit!“)

**Logische Sicht:**

* Sicht der Architektur, z.B. Klassendiagramme

**Prozess/Ablaufsicht:**

* Kommunikationsmechanismen, z.B. Komponentendiagramm

Struktursicht und Physische Sicht nicht relevant

# Architekturmuster

* bestimmte Lösungsstrukturen bewährt -> dokumentieren -> erneut verwenden  
  (evtl. praktisch in Arbeit zeigen?)

### Schichten

* jeweils logisch zusammenhängende Komponenten auf einer Schicht
* Schnittstellen nur zu Schichten darüber/von darunter
* vgl. Abstraktionsebenen
* Schicht darf nur von darüber liegenden Schichten abhängig sein und kennt nur Schicht darüber (nicht die darüber)
* bietet Dienste nach oben, nutzt Dienste von unten
* Unterscheidung zwischen Offener und Geschlossener Architektur (08/Folie 22)
* z.B. ISO-OSI-Schichtmodell, Client-Server

### Peer-to-Peer

* gleichberechtigte Partner
* Achtung: Durcheinander und Dead-Locks möglich

### Pipes und Filter

* Filter bearbeiten Daten, Pipes leiten weiter
* Flexibel bei neuen Anforderungen, gut für automatisierte Veränderungen der Daten
* nicht für hochinteraktive Aufgaben oder bei wenigen Änderungen

### SOA (Service orientated Architecture)

* gute Anpassbarkeit und Wartbarkeit
* Service: kapselt Zugriff auf Daten und Funktionen, erbringt bestimmte Leistung
* Randbedingungen festhalten (Vor- Nachbedingung und **Invariante**)
* Aufteilung in User, Provider und Directory (08/Folie 48 ff.)
* auch **Mikroservices** möglich (modularer Aufbau ermöglicht) (08/Folie 55)

### Vertikale Dekomposition

* aufteilen der Anwendung in mehrere Unteranwendungen (zu lösende Probleme werden kleiner, z.B. Eingabe und Selbstoptimierung -> 2 Anwendungen)

### MVC

**Model** -> Datenspeicher, Persistenz

* implementiert zentrale Datenstruktur

**View** -> alles sichtbare

**Controller** -> steuert IO und Funktionsaufrufe

# Allgemeines/Sonstiges

* Top-Down oder Bottom-Up?
* Kriterien für guten Entwurf: “Gute Dokumentation”!
* Implementierung austauschbar
* Datenkapselung

### Style

* mit

//region Test-Region

xxx

//endregion

lassen sich sehr schöne **Untergruppen im Code** bilden

### Test

* mit **Groovy** lassen sich schöne, nachvollziehbare Tests schreiben (siehe FunctionPointsEstimationTest.groovy)