Schriftliche Ausarbeitung

Domain Driven Design (DDD)

Analyse der Ubiquitous Language

Die Ubiquitous Language ist ein Konzept des Domain Driven Design, welches Verständnisprobleme zwischen Entwicklern und Domänenexperten, durch das Definieren einer gemeinsamen Projektsprache, vorbeugen soll. Die Projektsprache beinhaltet Definitionen für das gemeinsame Verständnis von Konzepten, Prozessen und Regeln aus der Domäne.

Wichtige Begriffe und Prozesse innerhalb der Problemdomäne "Aufgabenliste" sind:

- Aufgabe (Task)
- Fälligkeitsdatum (Due Date)
- Aufgabenliste (Task List)
- Arbeitsschritt (Sub Task)
- Erinnerung (Reminder)
- Liste anlegen
- Aufgabe verschieben/löschen/bearbeiten
- Liste löschen/bearbeiten

Aufgabe (Task):

Eine Aufgabe besteht aus Name/Titel, Erinnerungsdatum, Arbeitsschritte, Beschreibung/Notiz, Fälligkeitsdatum. Zudem hat eine Aufgabe mehrere mögliche Zustände:

- Erledigt und nicht erledigt
- Fälligkeitsdatum überschritten/nicht überschritten

Alle Bestandteile, außer dem Namen, sind optional. Die Anzahl Arbeitsschritte ist logisch nicht begrenzt.

Fälligkeitsdatum (Due Date):

Ein Fälligkeitsdatum kann auf einer Aufgabe gesetzt werden. Es ist darauf zu achten, dass das Fälligkeitsdatum nicht in der Vergangenheit liegen darf. Ist das Fälligkeitsdatum auf einer Aufgabe erreicht muss dies in der Aufgabe gekennzeichnet werden.

Aufgabenliste (Task List):

Eine Aufgabenliste kann mehrere (theoretisch unbegrenzt viele) Aufgaben beinhalten. Sie kann aber auch keine Aufgabe beinhalten. Aufgabenlisten können vom Benutzer erstellt werden. Dabei wird ein Name vom Benutzer festgelegt, welcher über alle Listen eindeutig sein muss. Die Bestandteile einer Aufgabenliste sind folglich ihr Name und eine Reihe von Aufgaben.

Arbeitsschritt (Sub Task):

Ein Arbeitsschritt kann selbst wieder als Aufgabe verstanden werden, mit dem Unterschied, dass ein Arbeitsschritt keine weiteren Arbeitsschritte enthalten darf.

Erinnerung (Reminder):

Eine Erinnerung kann auf einer Aufgabe gesetzt werden. Es ist erneut darauf zu achten, dass das Erinnerungsdatum nicht in der Vergangenheit liegt. Ist das Erinnerungsdatum auf einer Aufgabe erreicht, muss der Benutzer darüber in Kenntnis gesetzt werden.

Liste anlegen:

Eine neue Liste anzulegen heißt ihr einen eindeutigen Namen zu geben. Nach dem Anlegen der Liste ist diese zunächst leer. Der Benutzer hat jetzt die Möglichkeit zu dieser Liste aufgaben hinzuzufügen.

Aufgabe verschieben/löschen/bearbeiten:

Eine Aufgabe zu verschieben heißt, sie von einer Liste in eine andere zu bewegen. Dafür muss die entsprechende Aufgabe aus der Aufgabenliste, in der sie sich momentan befindet, gelöscht und anschließend einer anderen Liste hinzugefügt zu werden.
Eine Aufgabe zu löschen heißt alle mit ihr assoziierten Daten aus der Applikation zu entfernen, sodass diese anschließend nicht mehr abrufbar sind.
Eine Aufgabe zu bearbeiten, heißt eines der oben genannten Bestandteile einer Liste zu verändern oder die Aufgabe als erledigt bzw. unerledigt zu markieren. Wird eine Aufgabe als erledigt markiert wird sie aus ihrer aktuellen Liste entfernt und einer nicht vom Benutzer verwalteten Liste von erledigten Aufgaben zugeteilt, damit diese später noch abgerufen werden können. Eine Aufgabe als erledigt zu markieren ist also nicht dasselbe wie sie zu löschen! Wird eine Aufgabe wieder als unerledigt markiert wird ihr Status entsprechend verändert und sie wird der ursprünglichen Liste hinzugefügt aus der sie gekommen ist.

Liste löschen/bearbeiten:

Wird eine Liste gelöscht, dann löschen sich auch alle in ihr enthaltenen Aufgaben. Eine Liste zu bearbeiten bedeutet ihren Namen zu verändern. Dabei muss erneut geprüft werden, dass der Name eindeutig ist.

Analyse und Begründung der verwendeten Muster

Value Objects

Entities

Was sind Entities? Wo habe ich Entities in meinem Projekt verwendet und warum?

Domain Services

""

Aggregates

""
Was ist die "Root Entity" in meinem Aggregat und warum?

Repositories

""

Beschreibung und Realisierung in unterschiedlichen Packages! Keine Vermischung von essential und accidental complexity!

Factories

""

Clean Architecture

Schichtarchitektur planen und begründen

Programming Principles

Analyse und Begründung für SOLID

Analyse und Begründung für GRASP

Analyse und Begründung für DRY

Refactoring

Code Smells identifizieren

Begründung durchgeführter Refactorings

Entwurfsmuster

Einsatz begründen

UML vorher/nachher