Domain Driven Design:

- Fachlichkeit ist Hauptfokus (Fachlichkeit = Anwendungsdomäne)
- Es gibt Subdomänen (hat eigene Sprache, eigene Experten und eigenes Modell)
- Strategische Design: Beschreibung der Domäne / Subdomäne
- Taktisches Design: Strukturierung innerhalb einer Subdomäne

Layered Architecture: (Folie 55 und im Buch Seite 17/18)

Ein komplexes Programm wird in Schichten unterteilt, sodass jede Schicht nur von der darunterliegenden Schicht abhängig ist. Durch gängige Architekturmuster soll eine lose Kopplung zu den darüberliegenden Schichten erreicht werden.

Code der sich auf das Domänenmodell bezieht wird von der UI, der Applikationslogik und dem Infrastrukturcode getrennt.

Die Layered Architecture bezeichnet die technische Schichtung.

Modules: (Folie 56/57 und Buch Seite 27)

Mit Modules kann die zu entwickelnde Software in fachliche Bereiche aufgeteilt werden. Durch Modules werden eine hohe Kohäsion und eine lose Kopplung umgesetzt. Die Modules sollten sprechende Namen wie bspw. „Einkauf“ oder „Studium“ enthalten.

Definition Kohäsion: Eine hohe Kohäsion liegt vor, wenn eine Klasse/Methode/Modul genau für eine definierte Aufgabe verantwortlich ist. Bedeutet also die Kohäsion gibt an, wie gut eine Programmeinheit eine logische Aufgabe oder Einheit abbildet. (vgl.: https://de.wikipedia.org/wiki/Koh%C3%A4sion_(Informatik) Zugriff am 01.04.2022)

Anti-Corruption-Layer: (Buch Seite 55)

Eine Zwischenschicht zwischen Businesslogik und der DataAccess-Schicht, damit das Upstream-System (BL) auf die Funktionalität des Downstream-Client (DA) zugreifen kann. Diese Schicht kommuniziert mit dem anderen System über die vorhandene Schnittstelle.

Dadurch wird wenig oder gar keine Änderung am anderen System notwendig sein. Diese Schicht dient als Übersetzer zwischen den beiden Schichten.

Services (Folie 63 und Buch Seite 26)

Wenn sich eine Funktionalität nicht als eine Entität, Value-Object oder Aggregate zuordnen lässt, wird es eben ein Service. Der Service erhält Vor- und Nachbedingungen die über Assertions sichergestellt werden. Auch Services müssen über die upiquitäre Sprache ausgedrückt werden und sollten sprechende Namen haben.

Entities (Folie 58, Buch Seite 19/20)

Eine Entity ist im DDD ein individuelles Ding der Realität. Es besitzt einen Zustand und ein bestimmtes Verhalten. Eine Entity kann aber nicht durch die Attribute unterschieden werden, sodass zusätzlich ein eindeutiges Attribut erzeugt werden muss, damit jedes Objekt der Entity eindeutig wird. Die Klassendefinition ist stets einfach zu halten.

Value Objects: (Folie 59 und Buch Seite 21/22)

Im Gegensatz zu Entitys haben Value Objects keine eindeutige Identität sondern werden durch ihre Attribute und den Zustand identifiziert. Der Zustand sollte allerdings unveränderlich sein, sodass Value Objects sogenannte Immutable Objects sind.

Beispiel: Bei einem Geldschein ist die eindeutige ID oftmals irrelevant, es genügt den Zustand (Betrag und Währung) zu betrachten.

Repositories (Folie 62 und Buch Seite 30/31)

Kapselt Transaktionen auf z.B. eine Datenbank (Lesen, Schreiben, Löschen, Ändern, Suchen/Finden). Entities und Aggregates können auf verschiedenen Tabellen verteilt vorliegen. Das Repository sorgt dann für das Mapping der Tabelleneinträge auf die Objekte.

Aggregates (Folie 61 und Buch Seite 28)

Sind im Modell komplexe Beziehungen kann es vorkommen, dass die Konsistenz von Änderungen an Objekten schwierig einzuhalten sind. Es kann durchaus sein, dass sich mehrere Benutzer gegenseitig stören und das System unbrauchbar wird.

Deshalb können Entities und Value Objects zu Aggregates zusammengefasst und Grenzen definiert werden. Wichtig ist dabei, dass ein Entity als Wurzelentity definiert und für externe Objekte aufrufbar wird.

Beispiel: Root-Entity: „StudentIN“, Object Value: „Adresse“ und „Telefonnummern“

Factories (Folie 62 und Buch Seite 32)

Wenn der Client das Objekt direkt erstellen muss, hat dies Auswirkungen auf die Kapselung des erzeugten Value Objects oder Aggreates, da der Client zu stark an der Implementierung des erstellten Objekts gekoppelt ist.

Deshalb wird das Erstellen von Objekten auf ein separates Objekt ausgegliedert, welches selbst keine weiteren Aufgaben im Modell besitzt. hallo