Um nun folglich eine Microservicesarchitektur umzusetzen, muss zum einem die Anwendung so Komplex sein, dass sie sich in einzelne Services teilen lässt und zum anderen so viele Entwickler vorhanden sein, dass idealerweise jeder Dienst von einem Team betreut wird.

2.2.3 Domain Driven Design

Wie aus dem Abschnitt Abschnitt 2.2.2 hervorgeht, ist es das Ziel der Microservicesarchitektur möchglichst autonome Team zu etablieren und unnötigen Absprachen zu reduzieren. Um dies zu erreichen und gleichzeitig die hohe Kohärenz bei geringer Kopplung zu besitzen, bauen Microservices auf der Idee des Domain Driven Designs auf, welche von Eric Evans in seinem gleichnamigen Buch 2003 publiziert wurde. Es handelt sich dabei um eine Herangehensweise bzw. Denkweise zur Modellierung komplexer Systeme, bei der, dass Gesamtsystem anhand der einzelnen Geschäftsprozesse geteilt wird.

Domain Driven Design ist dabei nicht exklusiv für Microserivces gedacht, sondern beschreibt vielmehr den Ansatz Softwareentwicklung und Business zu vereinen, welche in verschiedenen Architekturen umgesetzt werden kann. Nichtsdestoweniger ist Domain Driven Design wesentlich für die Bestimmung der Grenzen eines Services und somit entscheiden für Microservices.

Ein Hauptbestandteil des Domain Driven Designs ist der Gedanke des Kontextes bzw. der Kontextgrenze. Eine Kontextgrenze ist dabei die Abgrenzung eines logischen Abschnittes (Kontext) innerhalb eines Unternehmens. Dadurch wird der Ansatz verfolgt, dass einzelne Funktionen und Inhalte nur innerhalb eines gewissen Bereiches Sinn ergeben. So ist zum Beispiel in einem E-Commerce Unternehmen bei der Bestellung von Ware die Anzahl, die Größe und das Gewicht entscheidend, während bei der Buchung Preis und Steuersatz wichtig sind. Des Weiteren kann je nach Kontext ein Begriff, wie Preis, unterschiedliche Bedeutungen haben. So bezieht sich dieser beim Bestellen von neuer Ware auf den Einkaufspreis, während er beim Verkaufen den Verkaufspreis meint. Somit hängt die Bedeutung des Wortes Preis vom dem zu lösenden Geschäftsprozess ab, in diesem Fall Bestellen von Ware oder Verkaufen von Produkten. Dies ist zwar nur ein einzelnes Beispiel, lässt sich jedoch auf Weitere übertragen. Demnach liegt Domain Driven Design nah Kontextgrenzen so zu legen, dass sie ein Geschäftsprozess umschließen und idealerweise genau ein spezifisches Problem lösen.

Wie eingangs beschrieben handelt es sich jedoch bei Domain Driven Design nur um eine Herangehensweise und nicht um klare Regeln. Daher liegt es im Ermessen jedes Unternehmens die Granularität der Kontexte zu setzen.

Der Fokus auf ein spezifische Problemstellung, ermöglicht eine einheitliche Sprache innerhalb eines Kontexts einzufügen, sowie eine enge Verbindung zwischen Geschäftslogik

und technische Entwicklung zu erzeugen. Somit vertritt Domain Driven Design den Ansatz, dass technische Modelle und Prozessbeschreibungen die gleiche Terminologie führen. Dadurch soll die Kommunikation zwischen Fachexperten und Softwareentwickler gestärkt werden.

Konkret für die Einteilung von Microservices bedeutet dies, dass ein Microservices eine Kontext abbildet, welche an ein Geschäftsprozess orientiert ist. Des Weiteren fordert Domain Driven Design, dass ein Team, welches für ein Service verantwortlich ist, auch Personen aus der Geschäftsabteilung hat und dass das Modell, durch ein fachlich diverses Team entsteht. In der strengsten Umsetzung von Domain Driven Design, besteht folglich ein Team aus Fachexperten, sowie Softwareentwicklern und besitzt einen engen Kontakt zu Usern. Nun handelt es sich bei Domain Driven Design jedoch um eine Herangehensweise und wird nicht immer absolut umgesetzt.

Die Umsetzung des Domain Driven Design hat jedoch einige Vorteile. So wird durch das abgrenzen in einzelne Kontext, die Autonomität des Teams weiter bestärkt und der Austausch zwischen Fachexperten und Softwareentwickler durch die einheitliche Sprache verstärkt. Dadurch entstehen eigenständige Services, die hinsichtliche ihrer verwendeten Daten, als auch der verwendeten Terminologie, entkoppelt sind. Eingeteilt wird anhand von Geschäftsprozessen.

2.3 Microservice

Nach Sam Newman ist ein Microservice ein *„eigenständige ausführbare Softwarekomponente, die innerhalb eines Anwendungssystems mit anderen Softwarekomponenten kollaboriert“* (Newman 2019, Kap. 2.1). Sie zeichnet sich durch das kommunizieren über definierte Netzwerkschnittstellen aus und formt in Vereinigungen eine Microservices-Architektur. Ein Microservice umfasst dabei die Datenspeicherung, Datenverarbeitung und Datendarstellung und besitzt eine gut definierte Benutzeroberfläche (vgl. Newman 2019, Kap. 2.1).

Ergänzend dazu schreibt Wolf, dass sich das Konzept der Microservices-Architektur aus der Philosophie vom Unix Betriebssystem ableitet, welches nach Peter H. Salus folgende drei Leitpunkte umfasst (Salus 1994; vgl. Wolff 2018, Kap. 1.1):

- Schreibe Programme, sodass sie nur eine Aufgabe erledigen und diese gut.
- Schreibe Programme, die zusammen arbeiten.
- Schreibe Programme, welche über definierte Schnittstellen (Textstream) kommunizieren.

Nach James Lewis sind Microservices kleine Anwendungen, die unabhängig bereitgestellt, getestet und skaliert werden. Ebenfalls wie Wolf beschreibt er diese Programme, als einfach