

Curriculum für

Certified Professional for
Software Architecture (CPSA)[®]
Advanced Level

**Modul
DDD**

Domain-Driven Design

2023.1-RC1-DE-20230301



Inhaltsverzeichnis

Verzeichnis der Lernziele	2
Einführung: Allgemeines zum iSAQB Advanced Level	3
Was vermittelt ein Advanced Level Modul?	3
Was können Absolventen des Advanced Level (CPSA-A)?	3
Voraussetzungen zur CPSA-A-Zertifizierung	3
Grundlegendes	4
Was vermittelt das Modul „DDD“?	4
Struktur des Lehrplans und empfohlene zeitliche Aufteilung	4
Dauer, Didaktik und weitere Details	4
Voraussetzungen	5
Gliederung des Lehrplans	5
Ergänzende Informationen, Begriffe, Übersetzungen	5
1. Domäne, Modell und Ubiquitous Language	6
1.1. Begriffe und Konzepte	6
1.2. Lernziele	6
1.3. Referenzen	7
2. Knowledge Crunching: Der Weg zum Modell	8
2.1. Begriffe und Konzepte	8
2.2. Lernziele	8
2.3. Referenzen	11
3. Vom Modell zur Implementierung	12
3.1. Begriffe und Konzepte	12
3.2. Lernziele	12
3.3. Referenzen	13
4. Das Modell in der Anwendungsarchitektur	14
4.1. Begriffe und Konzepte	14
4.2. Lernziele	14
4.3. Referenzen	14
5. Strategic Design 1: Modelle schneiden und voneinander abgrenzen	15
5.1. Begriffe und Konzepte	15
5.2. Lernziele	15
5.3. Referenzen	16
6. Strategic Design 2: Context Mapping	17
6.1. Begriffe und Konzepte	17
6.2. Lernziele	17
6.3. Referenzen	18
7. Optional: Verwandte Themen	19



7.1. Empfehlungen 19

7.2. Begriffe, Konzepte, mögliche Ideen 19

7.3. Referenzen 19

Referenzen 20

© (Copyright), International Software Architecture Qualification Board e. V. (iSAQB® e. V.) 2023

Die Nutzung des Lehrplans ist nur unter den nachfolgenden Voraussetzungen erlaubt:

1. Sie möchten das Zertifikat zum CPSA Certified Professional for Software Architecture Foundation Level® oder CPSA Certified Professional for Software Architecture Advanced Level® erwerben. Für den Erwerb des Zertifikats ist es gestattet, die Text-Dokumente und/oder Lehrpläne zu nutzen, indem eine Arbeitskopie für den eigenen Rechner erstellt wird. Soll eine darüber hinausgehende Nutzung der Dokumente und/oder Lehrpläne erfolgen, zum Beispiel zur Weiterverbreitung an Dritte, Werbung etc., bitte unter info@isaqb.org nachfragen. Es müsste dann ein eigener Lizenzvertrag geschlossen werden.
2. Sind Sie Trainer oder Trainingsprovider, ist die Nutzung der Dokumente und/oder Lehrpläne nach Erwerb einer Nutzungslizenz möglich. Hierzu bitte unter info@isaqb.org nachfragen. Lizenzverträge, die alles umfassend regeln, sind vorhanden.
3. Falls Sie weder unter die Kategorie 1. noch unter die Kategorie 2. fallen, aber dennoch die Dokumente und/oder Lehrpläne nutzen möchten, nehmen Sie bitte ebenfalls Kontakt unter info@isaqb.org zum iSAQB e. V. auf. Sie werden dort über die Möglichkeit des Erwerbs entsprechender Lizenzen im Rahmen der vorhandenen Lizenzverträge informiert und können die gewünschten Nutzungsgenehmigungen erhalten.

Wichtiger Hinweis

Grundsätzlich weisen wir darauf hin, dass dieser Lehrplan urheberrechtlich geschützt ist. Alle Rechte an diesen Copyrights stehen ausschließlich dem International Software Architecture Qualification Board e. V. (iSAQB® e. V.) zu.

Die Abkürzung "e. V." ist Teil des offiziellen Namens des iSAQB und steht für "eingetragener Verein", der seinen Status als juristische Person nach deutschem Recht beschreibt. Der Einfachheit halber wird iSAQB e. V. im Folgenden ohne die Verwendung dieser Abkürzung als iSAQB bezeichnet.

Verzeichnis der Lernziele

- LZ 1-1: Die Zusammenhänge zwischen Domäne, Software und Modell kennen und erläutern können
- LZ 1-2: Die Rolle der Fachsprache bei der Konstruktion einer Ubiquitous Language verstehen
- LZ 1-3: Die Bausteine von Domain Driven Design kennen und erklären können
- LZ 1-4: Die Zusammenhänge zwischen den Bausteinen kennen und erklären können
- LZ 2-1: Den hohen Stellenwert von Domänenexpert:innen in DDD kennen und erläutern können
- LZ 2-2: Bei der Auswahl geeigneter Ansprechpartner:innen unterstützen können
- LZ 2-3: Mit Domänenexpert:innen kommunizieren können
- LZ 2-4: Modellierungstechniken bei der Zusammenarbeit mit Domänenexpert:innen einsetzen können
- LZ 2-5: Interviews als Mittel zur Modellierung führen können
- LZ 2-6: Beobachtung als Mittel zur Modellierung beherrschen
- LZ 2-7: Einen Überblick über Collaborative Modeling, seine Methoden und seinen Zusammenhang mit DDD bekommen
- LZ 2-8: Ein geeignetes Vorgehen zur Modellierung auswählen und mit Domänenexpert:innen diskutieren können
- LZ 2-9: Einen Collaborative Modeling-Workshop durchführen können
- LZ 2-10: Agilität als Fundament von DDD verstehen
- LZ 3-1: Ein Domänenmodell um technisch motivierte Bausteine von DDD erweitern können
- LZ 3-2: Schnittstellen für fachliche Klassen modellieren können
- LZ 3-3: Wechselwirkungen zwischen einer Implementation und ihrem Modell kennen und berücksichtigen können
- LZ 3-4: Argumentieren können, wieso sich DDD bei komplexer Geschäftslogik lohnt
- LZ 4-1: Eine Ports & Adapter-Architektur um ein Domänenmodell herum entwerfen
- LZ 4-2: Zusammenhänge und Abgrenzungen zwischen DDD und BDD formulieren können
- LZ 5-1: Symptome kennen, die bei zu großen Modellen auftreten können
- LZ 5-2: Vor- und Nachteile eines Team-übergreifenden Modells abwägen können
- LZ 5-3: Vom Problem- zum Lösungsraum kommen können
- LZ 5-4: Den Kern herausdestillieren
- LZ 5-5: Modellgrenzen von Bounded Contexts in einer Context Map beschreiben können
- LZ 6-1: Schnittstellen bei Customer/Supplier Teams einsetzen können
- LZ 6-2: Ein System als Open Host Service (OHS) konzipieren können
- LZ 6-3: Das eigene Modell von äußeren Einflüssen isolieren können
- LZ 6-4: Kern-Elemente mehrerer Teilmodelle in einem Shared Kernel wiederverwenden können
- LZ 6-5: Umstände verstehen, in denen eine Aufteilung des Modells sinnvoll ist (Separate Ways)
- LZ 6-6: Domain Events als Kommunikationsmittel zwischen Bounded Contexts nutzen können

Einführung: Allgemeines zum iSAQB Advanced Level

Was vermittelt ein Advanced Level Modul?

Das Modul kann unabhängig von einer CPSA-F-Zertifizierung besucht werden.

- Der iSAQB Advanced Level bietet eine modulare Ausbildung in drei Kompetenzbereichen mit flexibel gestaltbaren Ausbildungswegen. Er berücksichtigt individuelle Neigungen und Schwerpunkte.
- Die Zertifizierung erfolgt als Hausarbeit. Die Bewertung und mündliche Prüfung wird durch vom iSAQB benannte Expert:innen vorgenommen.

Was können Absolventen des Advanced Level (CPSA-A)?

CPSA-A-Absolventen können:

- eigenständig und methodisch fundiert mittlere bis große IT-Systeme entwerfen
- in IT-Systemen mittlerer bis hoher Kritikalität technische und inhaltliche Verantwortung übernehmen
- Maßnahmen zur Erreichung von Qualitätsanforderungen konzeptionieren, entwerfen und dokumentieren sowie Entwicklungsteams bei der Umsetzung dieser Maßnahmen begleiten
- architekturrelevante Kommunikation in mittleren bis großen Entwicklungsteams steuern und durchführen

Voraussetzungen zur CPSA-A-Zertifizierung

- erfolgreiche Ausbildung und Zertifizierung zum Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- mindestens drei Jahre Vollzeit-Berufserfahrung in der IT-Branche; dabei Mitarbeit an Entwurf und Entwicklung von mindestens zwei unterschiedlichen IT-Systemen
 - Ausnahmen sind auf Antrag zulässig (etwa: Mitarbeit in Open-Source-Projekten)
- Aus- und Weiterbildung im Rahmen von iSAQB-Advanced-Level-Schulungen im Umfang von mindestens 70 Credit Points aus mindestens drei unterschiedlichen Kompetenzbereichen
 - bestehende Zertifizierungen (etwa Sun/Oracle Java-Architect, Microsoft CSA) können auf Antrag angerechnet werden
- erfolgreiche Bearbeitung der CPSA-A-Zertifizierungsprüfung



Grundlegendes

Was vermittelt das Modul „DDD“?

Das Modul präsentiert den Teilnehmer:innen Domain Driven Design (DDD) als Mittel, Software als präzise, transparente und transformierbare Repräsentation einer fachlichen Domäne zu gestalten.

Am Ende des Moduls kennen die Teilnehmer:innen die wesentlichen Prinzipien des Domain Driven Designs und können diese bei Entwurf und Implementierung von Softwaresystemen anwenden. Sie sind mithilfe der vermittelten kommunikativen Fähigkeiten in der Lage, eine einheitliche Sprache zwischen Fachexpert:innen und Entwickler:innen zu etablieren. Mit Hilfe der vermittelten Modellierungstechniken und Architekturwerkzeuge können sie die Bestandteile dieser gemeinsamen Fachsprache in ihre Softwaresysteme übernehmen.

Bei einem großen Softwareprojekt ist oft der Einsatz von mehreren Entwicklungsteams erforderlich. Dieses Modul adressiert diese Herausforderung und vermittelt den Teilnehmer:innen Methoden des Domain Driven Designs, um mit der wachsenden Komplexität eines großen Softwareprojektes umzugehen.

Struktur des Lehrplans und empfohlene zeitliche Aufteilung

Inhalt	Empfohlene Mindestdauer (min)
1. Domäne, Modell und Ubiquitous Language	195
2. Knowledge Crunching: Der Weg zum Modell	240
3. Vom Modell zur Implementierung	120
4. Das Modell in der Anwendungsarchitektur	165
5. Strategic Design 1: Modelle schneiden und voneinander abgrenzen	210
6. Strategic Design 2: Context Mapping	90
7. Optional: Verwandte Themen	
Summe	1020 (17h)

Dauer, Didaktik und weitere Details

Die unten genannten Zeiten sind Empfehlungen. Die Dauer einer Schulung zum Modul DDD sollte mindestens 3 Tage betragen, kann aber länger sein. Anbieter können sich durch Dauer, Didaktik, Art und Aufbau der Übungen sowie der detaillierten Kursgliederung voneinander unterscheiden. Insbesondere die Art der Beispiele und Übungen lässt der Lehrplan komplett offen.

Lizenzierte Schulungen zu DDD tragen zur Zulassung zur abschließenden Advanced-Level-Zertifizierungsprüfung folgende Credit Points) bei:

Methodische Kompetenz:	20 Punkte
Technische Kompetenz:	0 Punkte
Kommunikative Kompetenz:	10 Punkte

Voraussetzungen

Teilnehmerinnen und Teilnehmer **sollten** folgende Kenntnisse und/oder Erfahrung mitbringen:

- Grundlagen und weiterführende Konzepte der objektorientierten Softwareentwicklung
- Erfahrungen bei der Modellierung von objektorientierten Architekturen

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Wissen über agile Methoden der Softwareentwicklung, wie z. B. Scrum, Kanban, XP, etc.
- Erfahrung bei der Zusammenarbeit von Fachbereich und Softwareentwickler:innen.

Gliederung des Lehrplans

Die einzelnen Abschnitte des Lehrplans sind gemäß folgender Gliederung beschrieben:

- **Begriffe/Konzepte:** Wesentliche Kernbegriffe dieses Themas.
- **Unterrichts-/Übungszeit:** Legt die Unterrichts- und Übungszeit fest, die für dieses Thema bzw. dessen Übung in einer akkreditierten Schulung mindestens aufgewendet werden muss.
- **Lernziele:** Beschreibt die zu vermittelnden Inhalte inklusive ihrer Kernbegriffe und -konzepte.

Dieser Abschnitt skizziert damit auch die zu erwerbenden Kenntnisse in entsprechenden Schulungen.

Ergänzende Informationen, Begriffe, Übersetzungen

Soweit für das Verständnis des Lehrplans erforderlich, haben wir Fachbegriffe ins [iSAQB-Glossar](#) aufgenommen, definiert und bei Bedarf durch die Übersetzungen der Originalliteratur ergänzt.

1. Domäne, Modell und Ubiquitous Language

Dauer: 120 Min.	Übungszeit: 75 Min.
-----------------	---------------------

1.1. Begriffe und Konzepte

- Domäne (engl.: domain), Domänenmodell (engl.: domain model)
- Ubiquitous Language
- Module
- Entity, Value Object, Aggregate, Service
- Factory, Repository
- Domain Events (Vernon, 2013)

1.2. Lernziele

Dieser Block dient der Motivation und Einführung in Domain Driven Design. Die grundlegenden Konzepte Domäne, Domänenmodell und Ubiquitous Language werden hier vermittelt. Zudem erhalten die Teilnehmer:innen einen detaillierten Einblick in die verschiedenen Bausteine von Domänenmodellen und die dazwischenliegenden Beziehungen.

LZ 1-1: Die Zusammenhänge zwischen Domäne, Software und Modell kennen und erläutern können

- Die Teilnehmer:innen (TN) können die Abhängigkeit von Software zu einer Domäne beschreiben. Sie verstehen, dass Software nicht zum Selbstzweck existiert.
- Die TN verstehen Domänenmodelle als Mittel zur Abstraktion von Fachwissen.
- Die TN verstehen, dass Domänenmodelle die Ideen und Zusammenhänge einer Domäne repräsentieren.
- Die TN können das Domänenmodell als Hilfsmittel zur Annäherung von Software an die Domäne erklären.

LZ 1-2: Die Rolle der Fachsprache bei der Konstruktion einer Ubiquitous Language verstehen

- Die TN verstehen, dass eine gemeinsame Sprache für Domänenexpert:innen und Entwickler:innen dem wechselseitigen Verständnis dient.
- Die TN verstehen den Begriff der Ubiquity (Allgegenwärtigkeit): "Allgegenwärtigkeit" gilt in einer definierten Grenze (Bounded Context) innerhalb derer die Sprache in Gesprächen, Diagrammen, Dokumentation und Code konsistent verwendet wird.
- Die TN verstehen, dass die grundlegenden Begriffe einer Ubiquitous Language der Fachsprache der Domänenexpert:innen entspringen.
- Die TN verstehen, dass die Ubiquitous Language eine formalisierte Version der Fachsprache und eine bewusste Entwurfsentscheidung ist.
- Die TN verstehen, dass sich die Ubiquitous Language im Dialog von Entwicklungsteam und Fachbereich weiterentwickelt.

LZ 1-3: Die Bausteine von Domain Driven Design kennen und erklären können

- Die TN verstehen die grundlegenden Bausteine von Domain Driven Design.
 - Value Objects stellen elementare Wert-Typen aus der fachlichen Welt dar. Sie können nur andere Value Objects enthalten. Value Objects haben keine Identität, sind jedoch vergleichbar.
 - Entities repräsentieren Dinge der fachlichen Welt. Sie können Value Objects und andere Entities enthalten. Entities haben eine Identität.
- Die TN verstehen die weiterführenden Bausteine von Domain Driven Design.
 - **Modules** als statische Gruppierungsmechanismen für Code-Artefakte.
 - **Services** kapseln eigenständige Funktionen, die sich nicht einzelnen Entities zuordnen lassen. Sie entkoppeln Funktionalität von Zuständen, und sollten deswegen selbst zustandslos sein. Sie ermöglichen unter anderem das Erfassen von fachlichen Prozessen in einem Domänenmodell.
 - **Aggregates** dienen der Gruppierung von Entities. Der äußere Zugriff geschieht ausschließlich über ein global identifizierbares Aggregate Root. Das Aggregate Root ist zudem für die Einhaltung von Invarianten innerhalb des Aggregates verantwortlich. Dies erzielt eine lose Kopplung in der Benutzung von Aggregates, versteckt die enthaltenen Entities vor äußerem Zugriff und erleichtert so das Einhalten von Invarianten. Der Lebenszyklus aller enthaltenen Entities wird durch den Lebenszyklus des Aggregate Roots bestimmt.
 - **Factories** stellen Erzeugungsmechanismen für Entities dar. Mit ihnen lässt sich potenziell komplexe Logik aus den Konstruktoren von Entities auslagern.
 - **Repositories** dienen der Bestandsverwaltung von Entities zur Laufzeit. Das Ermitteln und Herausgeben einer Referenz auf ein Entity oder Aggregate zu einem eindeutigen Identifikator fällt in ihren Aufgabenbereich. Hinter ihnen lassen sich Schnittstellen zu Drittsystemen wie Datenbanken oder entfernten Services verbergen.
 - **Domain Events** unterstützen die Verbreitung von Informationen über das Auftreten von fachlichen Ereignissen. Fachliche Ereignisse werden von Aggregates oder Entities ausgelöst und über ein Publisher/Subscriber-Pattern (vgl. Observer [\[Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. \(1994\)\]](#)) an angemeldete Klienten propagiert.

LZ 1-4: Die Zusammenhänge zwischen den Bausteinen kennen und erklären können

- Die TN sind in der Lage die Bausteine in Beziehung zu setzen und sinnvoll miteinander zu kombinieren (vgl. Übersicht auf Seite 65 [\[Evans, E. \(2003\)\]](#)).

1.3. Referenzen

[\[Avram, A., & Marinescu, F. \(2007\)\]](#), [\[Evans, E. \(2003\)\]](#), [\[Vernon, V. \(2013\)\]](#)

2. Knowledge Crunching: Der Weg zum Modell

Dauer: 180 Min.	Übungszeit: 60 Min.
-----------------	---------------------

2.1. Begriffe und Konzepte

- Agile und evolutionäre Modellbildung
- Empowerment der Domänenexpert:innen
- Kollaborationsformen zur Modellerhebung
- Werkzeuge zur Modellerhebung

2.2. Lernziele

In diesem Block lernen die Teilnehmer:innen, wie sie gemeinsam mit Domänenexpert:innen ein Domänenmodell erstellen und verfeinern können.

LZ 2-1: Den hohen Stellenwert von Domänenexpert:innen in DDD kennen und erläutern können

- Die TN können Domänenexpert:innen vermitteln, dass DDD ihnen Verantwortung und Gestaltungsmöglichkeiten bietet (Empowerment).
- Die TN sind in der Lage, einem Entwicklungsteam zu vermitteln, dass Software die vorhandene Domäne unterstützen muss.
- Die TN verstehen, dass das Wissen über die Domäne überwiegend in den Köpfen der Domänenexpert:innen vorhanden ist.
- Die TN verstehen, dass implizites Domänenwissen offengelegt und festgehalten werden muss, damit es modelliert werden kann.
- Die TN sind in der Lage, Vertrauen bei Domänenexpert:innen in das Modell und in die gleichberechtigte Zusammenarbeit zu schaffen.
- Die TN können Projektsponsoren und Product Ownern die Notwendigkeit für die Mitgestaltung durch die Domänenexpert:innen vermitteln.

LZ 2-2: Bei der Auswahl geeigneter Ansprechpartner:innen unterstützen können

- Die TN kennen Einflussfaktoren auf die Eignung von Domänenexpert:innen und können diese einschätzen:
 - Umfassende Erfahrung und Wissen in der eigenen Domäne
 - Motivation
 - Zeitliche Verfügbarkeit während und nach dem Projekt
 - Abstraktionsvermögen
 - Flexibilität

LZ 2-3: Mit Domänenexpert:innen kommunizieren können

- Die TN beherrschen Modelle um mit Domänenexpert:innen in einen gleichberechtigten Dialog zu treten:

- Das Kommunikationsquadrat
- Vier-Ohren-Modell
- Das innere Team
- Die TN verstehen, dass der Kommunikation zwischen Fachbereichen und Entwicklungsteams in DDD eine zentrale Bedeutung zukommt.
- Die TN verstehen, dass Domänenexpert:innen bewusst und unbewusst über Domänenwissen verfügen.

LZ 2-4: Modellierungstechniken bei der Zusammenarbeit mit Domänenexpert:innen einsetzen können

- Die TN beherrschen die Verwendung von Klassen- und Objektdiagrammen zur Darstellung von Domänenmodellen.
- Die TN beherrschen szenariobasierte Modellierungstechniken:
 - Use-Cases
 - User-Stories zur späteren Modellierung von Domain Events
 - Domain Events
- Die TN können Glossare für die Begriffe einer Ubiquitous Language anlegen.

LZ 2-5: Interviews als Mittel zur Modellierung führen können

- Die TN verstehen, dass Interviews geeignet sind, um bewusstes Domänenwissen offenzulegen.
- Die TN verstehen, dass der Interviewer oder die Interviewerin das Gespräch auch unbewusst beeinflusst.
 - die Auswahl der Fragen bestimmt, was zur Sprache kommt
 - Interviewer:innen haben eigene Annahmen, die Interviewpartner:innen nicht kennen
 - durch den Bestätigungsfehler können Missverständnisse entstehen
- Die TN beherrschen es, ein Interview entlang eines konkreten Szenarios aus der Domäne zu strukturieren und durchzuführen.
- Die TN verstehen, dass es hilfreich ist, während eines Interviews ein Modell aufzubauen, welches Interviewpartner:innen nachvollziehen können.

LZ 2-6: Beobachtung als Mittel zur Modellierung beherrschen

- Die TN können die Beobachtungstechniken "Feldbeobachtung" und "Apprenticing" (vgl. [\[Hruschka, P. \(2019\)\]](#)) als Mittel zur Offenlegung unbewussten Domänenwissens anwenden:
 - Feldbeobachtung
 - Analyse als stiller Beobachter
 - Arbeitsabläufe werden schriftlich erfasst
 - Verständnisfragen an die Domänenexpert:innen in seltenen Fällen
 - Apprenticing
 - Ist eine Erweiterung der Feldbeobachtung

- Nach einer Reihe von Beobachtungsdurchläufen übernehmen die Beobachter exemplarisch einen kleinen Teil der Arbeit der Domänenexpert:innen

LZ 2-7: Einen Überblick über Collaborative Modeling, seine Methoden und seinen Zusammenhang mit DDD bekommen

- Die TN wissen, dass es verschiedene Methoden gibt, z. B.:
 - EventStorming (Brandolini, 2021)
 - Domain Storytelling (Hofer, Schwentner 2022)
 - User Story Mapping (Patton, 2015)
 - Impact Mapping (Adzic)
 - Example Mapping (Wynne)
- Die TN verstehen die gemeinsamen Werte der verschiedenen Methoden:
 - Gemeinsames Verständnis erarbeiten durch enge Zusammenarbeit unterschiedlicher Stakeholder: Fachexpert:in, Entwickler:in und weitere.
 - Leichtgewichtige Methodik, deren Elemente in fünf Minuten so weit erklärt sind, dass man starten kann.
 - Fokus auf Verhalten, nicht nur auf Struktur/Daten (nicht nur Substantive, sondern auch Verben).
- TN verstehen, dass Collaborative Modeling auch unabhängig von DDD verwendet werden kann.

LZ 2-8: Ein geeignetes Vorgehen zur Modellierung auswählen und mit Domänenexpert:innen diskutieren können

- Die TN können organisatorische Rahmenbedingungen ermitteln und berücksichtigen:
 - Technische und räumliche Ressourcen
 - Räumliche Verteilung der Domänenexpert:innen
 - Juristische Rahmenbedingungen für das Anfertigen von Mitschriften, Audio/Video-Streams, Fotos, etc.
- Die TN können mit den Domänenexpert:innen diskutieren, ob das Modell iterativ oder im Voraus entwickelt werden soll.
- Die TN können die Folgen von Unschärfe und Fehlannahmen im Modell mit Domänenexpert:innen und Entwickler:innen diskutieren.

LZ 2-9: Einen Collaborative Modeling-Workshop durchführen können

- Die TN können einen Workshop in mindestens einer der o. g. Methoden vorbereiten, moderieren, und nachbereiten.
- TN sind in der Lage, passende TN zum Workshop einzuladen.
- Die TN können die ungeordnete Information aus dem Workshop strukturieren.
- Die TN wissen, dass online und on-premise unterschiedlich vorgegangen werden muss.

LZ 2-10: Agilität als Fundament von DDD verstehen

- TN können DDD zu Werten und Prinzipien aus dem Agilen Manifest in Beziehung setzen

- TN haben verstanden, dass DDD auf einen evolutionären Entwurf setzt:
 - Ein Modell entwickelt sich immer weiter.
 - Tragfähige Modelle entstehen durch exploratives Vorgehen.

2.3. Referenzen

[Adzic, G. (2012)], [Beck, K. et al. (2001)], [Brandolini, A. (18. November 2013)], [Hofer, S. & Schwentner, H. (2022)], [Hruschka, P. (2019)], [Patton, J. (2014)], [Wynne, M. (8. Dezember 2015)]

3. Vom Modell zur Implementierung

Dauer: 60 Min.	Übungszeit: 60 Min.
----------------	---------------------

3.1. Begriffe und Konzepte

- Kohäsion und Kopplung
- SOLID
- Zyklentreiheit
- Law of Demeter

3.2. Lernziele

Dieser Block vermittelt Methoden und Vorgehensweisen, um aus einem Domänenmodell die entsprechenden fachlichen Klassen abzuleiten.

LZ 3-1: Ein Domänenmodell um technisch motivierte Bausteine von DDD erweitern können

- Die TN können für ein vorliegendes Domänenmodell bestehend aus Entities, Value Objects und Services weitere Bausteine bestimmen:
 - Factories zur Erzeugung von Entities
 - Repositories für die Verwaltung von Entities
 - Aggregates, um Entities und Value Types zu kapseln

LZ 3-2: Schnittstellen für fachliche Klassen modellieren können

- Die TN können die Entwurfsprinzipien und Heuristiken aus dem Foundation Level für den DDD-Entwurf einschätzen und anwenden.

LZ 3-3: Wechselwirkungen zwischen einer Implementation und ihrem Modell kennen und berücksichtigen können

- Die TN verstehen, dass Veränderungen in der Fachsprache oder dem Modell entsprechende Veränderungen in der Software folgen müssen.
- Die TN verstehen, dass Änderungen an der Implementierung des Modells, beispielsweise durch Refactorings von Services oder Repositories, ein Hinweis darauf sind, dass das Domänenmodell aktualisiert werden sollte.

LZ 3-4: Argumentieren können, wieso sich DDD bei komplexer Geschäftslogik lohnt

- Die TN kennen alternative Ansätze und können die Vorteile von DDD bei komplexer Geschäftslogik vermitteln.
 - Table Module, Transaction Script und Domain Model (Fowler, Patterns of Enterprise Application Architecture, 2002)
 - Smart UI (Evans, 2003)

3.3. Referenzen

[Avram, A., & Marinescu, F. (2007)], [Fowler, M. (2002)], [Martin, R. C. (2002)], [Lilienthal, C. (2019)],
[Liebherr, K., Holland, I., & Riel, A. (1988)]

4. Das Modell in der Anwendungsarchitektur

Dauer: 105 Min.	Übungszeit: 60 Min.
-----------------	---------------------

4.1. Begriffe und Konzepte

- Hexagonal Architecture (Cockburn, 2012)
- Command-Query Responsibility Segregation (Dahan, 2009), (Vernon, 2013)
- Layered Architecture (Evans, 2003)
- Dependency Injection (Vernon, 2013)
- Behaviour Driven Development (BDD)

4.2. Lernziele

Dieser Block vermittelt anhand ausgewählter Beispiele, wie ein Domänenmodell in Softwarearchitekturen integriert werden kann. Er vermittelt exemplarisch Gemeinsamkeiten und Unterschiede von Domain Driven Design und zwei verwandten Methoden.

LZ 4-1: Eine Ports & Adapter-Architektur um ein Domänenmodell herum entwerfen

- Die TN können die Unterschiede zwischen einer Ports&Adapter-Architektur (z.B. Hexagonal, Clean oder Onion) und einer Schichtenarchitektur vermitteln.

LZ 4-2: Zusammenhänge und Abgrenzungen zwischen DDD und BDD formulieren können

- Die TN kennen Spezifikationen aus BDD und können sie mit der Ubiquitous Language in Zusammenhang bringen. Sie verstehen, dass sich Spezifikationen mit den Wörtern der UL formulieren lassen.
- Die TN verstehen, dass BDD Anforderungen mithilfe der Benutzungsschnittstelle formuliert (Outside-In), und DDD Anforderungen dies über das Domänenmodell ausdrückt (Inside-Out), vgl.: [\[Stenberg, J. \(24. Februar 2015\)\]](#).

4.3. Referenzen

[\[Vernon, V. \(2013\)\]](#), [\[Evans, E. \(2003\)\]](#), [\[Cockburn, A. \(2012\)\]](#), [\[Dahan, U. \(9. Dezember 2009\)\]](#), [\[Lilienthal, C. \(2019\)\]](#), [\[North, D. \(3. März 2016\)\]](#)

5. Strategic Design 1: Modelle schneiden und voneinander abgrenzen

Dauer: 150 Min.	Übungszeit: 60 Min.
-----------------	---------------------

5.1. Begriffe und Konzepte

- Problem- vs. Lösungsraum
- Subdomain
- Bounded Context
- Core vs supporting vs generic
- Context Map

5.2. Lernziele

Dieser Block vermittelt die Grundlagen dafür, ein Modell in mehrere Teilmodelle aufzuspalten und durch definierte Modellgrenzen beherrschbar zu machen. Zudem vermittelt er Strategien, um mehrere Modelle, und die daraus erwachsenden Verantwortungen auf verschiedenen Teams zu verteilen. Es werden Lösungen aufgezeigt, mit denen sich Schnittstellen zwischen Modellen gestalten lassen.

LZ 5-1: Symptome kennen, die bei zu großen Modellen auftreten können

- Die TN können die folgenden Sachverhalte als Symptom identifizieren:
 - Verschiedene Anwendergruppen verwenden dieselben Begriffe auf unterschiedliche Art und Weise.
 - Das Modell lässt sich nicht mehr von einem Entwicklungsteam bewältigen.
 - Das Modell wirkt widersprüchlich.

LZ 5-2: Vor- und Nachteile eines Team-übergreifenden Modells abwägen können

- Die TN kennen Conway's Law [\[Conway, M. E. \(April 1968\)\]](#) und seine Auswirkungen auf Softwarearchitektur.
- Die TN verstehen, dass von mehreren Teams genutzte Modelle teamübergreifende Kommunikation und Koordination (z. B. beim Deployment) erfordern.
- Die TN verstehen, dass Modelle aufgeteilt werden müssen, damit mehrere Entwicklungsteams unabhängig voneinander arbeiten können.
- Die TN verstehen, dass definierte Modellgrenzen eine wichtige Rolle dabei spielen, den Kommunikationsaufwand gering zu halten.
- Die TN wissen, dass Begriffe der Fachsprache in verschiedenen Kontexten verschiedene Bedeutungen haben können.
- Die TN wissen, dass lokale Modellkonsistenz unabhängig von anderen Modellen aufrechterhalten werden kann.

LZ 5-3: Vom Problem- zum Lösungsraum kommen können

- Die TN kennen die Begriffe "Problemraum" und "Lösungsraum" und den Unterschied.

- Die TN verstehen im Lösungsraum "Modell" und "Design" und wie sie entstehen.
- Die TN verstehen, wie im Problemraum die Domäne verstanden und in Subdomänen aufgeteilt werden kann.
- Die TN verstehen, dass Sprache kontextabhängig ist und Sprache ein wichtiger Treiber für das Ableiten von Bounded Contexts aus Subdomänen ist.
- Die TN verstehen, wie Bounded Contexts aus Subdomänen abgeleitet werden können.

LZ 5-4: Den Kern herausdestillieren

- Die TN wissen, dass man große Systeme nicht in allen Bereichen gleich gut designen kann und es deshalb wichtig ist, zu wissen, in welchen Bereichen mehr Wert auf das Design gelegt werden muss.
- Die TN kennen die Begriffe core, generic, supporting subdomain.

LZ 5-5: Modellgrenzen von Bounded Contexts in einer Context Map beschreiben können

- Die TN können die Beziehungen zwischen mehreren Bounded Contexts als Context Map abbilden.
- Die TN verstehen, dass jedes Modell einen Kontext hat, auch wenn er nicht explizit ist.
- Die TN verstehen, dass die Begriffe der Ubiquitous Language nur innerhalb ihres Kontexts eine Bedeutung haben.
- Die TN verstehen, dass bei der Interaktion von verschiedenen Bounded Contexts die Bausteine der jeweiligen Kontexte ineinander übersetzt werden müssen.
- Die TN verstehen, die Vorteile der Korrelation zwischen den Modellgrenzen, der Organisationsstruktur und des Sourcecodes.

5.3. Referenzen

[Evans, E. (2003)], [Avram, A., & Marinescu, F. (2007)], [Conway, M. E. (April 1968)], [Vernon, V. (2013)]

6. Strategic Design 2: Context Mapping

Dauer: 60 Min.	Übungszeit: 30 Min.
----------------	---------------------

6.1. Begriffe und Konzepte

- Shared Kernel, Customer/Supplier Teams, Open Host Service (vgl. Übersicht auf Seite 388 [\[Evans, E. \(2003\)\]](#))
- Domain Event als Kommunikationsmittel
- Anticorruption-Layer
- Separate Ways [\[Vernon, V. \(2013\)\]](#)

6.2. Lernziele

Dieser Block vermittelt Ansätze, mit denen die Konsistenz innerhalb eines Modells sichergestellt werden kann und wann es dafür notwendig ist, Teile des Modells auszugliedern.

LZ 6-1: Schnittstellen bei Customer/Supplier Teams einsetzen können

- Die TN können beurteilen, ob zwei Teams sich in einer Customer/Supplier-Konstellation befinden.
- Die TN kennen die Rahmenbedingungen für die erfolgreiche Zusammenarbeit von Customer/Supplier Teams:
 - Stabilität und Dokumentation der Schnittstelle sind wichtig für die Integration
 - Gemeinsam entwickelte Akzeptanztests helfen, die Schnittstelle zu stabilisieren
 - Voraussetzung für eine funktionierende Kunde/Anbieter-Beziehung ist ein Interesse des Anbieters an der Nutzung durch den Kunden
- Die TN wissen, dass in einer Kundenbeziehung der Kunde die Anforderungen an die Schnittstelle des Anbieters formuliert.

LZ 6-2: Ein System als Open Host Service (OHS) konzipieren können

- Die TN verstehen, dass ein OHS die fachlichen Übersetzungsschichten der Klienten ersetzt.
- Die TN können Services für einen OHS entwerfen. Sie sind in der Lage, essentielle Anforderungen von spezifischen zu unterscheiden und im Entwurf zu berücksichtigen.
- Die TN verstehen, dass eine öffentliche Schnittstelle punktuell für besondere Anforderungen einzelner Klienten erweitert werden kann, ohne die Schnittstelle der anderen Klienten zu beeinträchtigen.
- Die TN verstehen, dass sich ein OHS besonders lohnt, wenn mehrere Kunden eine Übersetzungsschicht benötigen.

LZ 6-3: Das eigene Modell von äußeren Einflüssen isolieren können

- Die TN können erkennen, welchen Einfluss das Modell eines angrenzenden Systems auf das eigene Modell hat.
- Die TN können die Schnittstellen-Patterns (vgl. [\[Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. \(1994\)\]](#)) einsetzen, um ein Anticorruption Layer zu konstruieren.

LZ 6-4: Kern-Elemente mehrerer Teilmodelle in einem Shared Kernel wiederverwenden können

- Die TN können die Angemessenheit eines Shared Kernels in einer konkreten Situation bewerten und ihn entwerfen.
- Die TN verstehen, dass ein Shared Kernel hilft, Modellübersetzungen zu vermeiden.
- Die TN verstehen, dass ein Shared Kernel ein großes Maß an Koordination der beteiligten Teams erfordert.
- Die TN verstehen, dass die Teams bei der Arbeit am Shared Kernel gleichberechtigt sind.

LZ 6-5: Umstände verstehen, in denen eine Aufteilung des Modells sinnvoll ist (Separate Ways)

- Die TN können die Koordinierungs-Kosten eines gemeinsamen Modells dem Overhead bei getrennten Modellen gegenüberstellen.
- Die TN verstehen, dass die Aufteilung eines Modells entlang einer sinnvollen Grenze geschehen muss.
- Die TN verstehen, dass in diesen Fällen die lokale Modellkonsistenz in zwei Teilmodellen leichter zu wahren ist.
- Die TN sind in der Lage, die verbleibenden Funktionen können zu duplizieren oder miteinander zu integrieren.

LZ 6-6: Domain Events als Kommunikationsmittel zwischen Bounded Contexts nutzen können

- Die TN verstehen, dass Domain Events den Subscriber vom Publisher entkoppeln.
 - Insbesondere verstehen die TN, dass dies für den Publisher bedeutet, dass es für ihn nicht relevant ist, wer seine Events verarbeitet.
- Die TN sind in der Lage Domain Events als Kommunikationsmittel zwischen Bounded Contexts zu entwerfen.
- Die TN sind in der Lage, Chancen und Risiken von Domain Events in diesem Zusammenhang abzuwägen:
 - Wenn Events trotz vorhandener Abhängigkeiten (z. B. innerhalb eines Bounded Contexts) eingesetzt werden, besteht das Risiko, dass dadurch die Abhängigkeiten nur versteckt werden und zur Laufzeit zu schwer nachvollziehbaren Kontrollflüssen führen.
- Die TN sind in der Lage einen Event Store (vgl. Seite 539 [\[Vernon, V. \(2013\)\]](#)) einzusetzen, um es den Subscribern zu ermöglichen, fehlerhaft verarbeitete Events erneut zu verarbeiten.

6.3. Referenzen

[\[Evans, E. \(2003\)\]](#), [\[Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. \(1994\)\]](#)

7. Optional: Verwandte Themen

Dauer: frei	Übungszeit: frei
-------------	------------------

Rund um DDD gibt es viele Themen, die einen Bezug dazu haben und von Trainer:innen nach Bedarf auch gelehrt werden können.

7.1. Empfehlungen

- "Refactoring towards deeper insights" an einem Fallbeispiel verständlich machen
- "Large Scale Structures" über das Beispiel verständlich machen

7.2. Begriffe, Konzepte, mögliche Ideen

- Tools:
 - Bounded Context Canvas
 - User Story Mapping
- Implementierung:
 - Ein konkretes Implementationsbeispiel zeigen
 - CRC Cards
- Architektur:
 - Werkzeug- und Materialansatz (WAM)
 - Event Sourcing
 - CQRS
 - Legacy-Software mit DDD modernisieren
- Organisation und Strategie:
 - Team Topologies
 - Wardley Mapping
 - Systemische Organisationstheorie
 - Business Model Canvas & Value Proposition
- Kommunikation:
 - Wie überzeuge ich Kollegen und Chefinnen von DDD?

7.3. Referenzen

[Beck, K., & Cunningham, W. (1989)], [Dahan, U. (9. Dezember 2009)], [Evans, E. (2003)], [Fowler, M. (14. Juli 2011)], [Kühl, S. & Muster, J. (2018)], [Lilienthal, C. & Schwentner, H. (2023)], [Manns, M. L. & Rising, L. (2015)], [Osterwalder, A. & Pigneur, Y. (2010)], [Osterwalder, A. (2013)], [Skelton M. & Pais, M. (2019)], [Vernon, V. (2013)], [Wardley, S. (2020)], [Züllighoven, H. (1998)]

Referenzen

Dieser Abschnitt enthält Quellenangaben, die ganz oder teilweise im Curriculum referenziert werden.

A

- [Adzic, G. (2012)] **Impact Mapping**. Provoking Thoughts.
- [Avram, A., & Marinescu, F. (2007)] **Domain-Driven Design Quickly**. InfoQ Enterprise Software Development Series.

B

- [Beck, K., & Cunningham, W. (1989)] **A laboratory for teaching object oriented thinking**. OOPSLA '89 Conference proceedings on Object-oriented programming systems, languages and applications. ACM New York.
- [Beck, K. et al. (2001)] **Manifesto for Agile Software Development**. Von <https://agilemanifesto.org> abgerufen
- [Brandolini, A. (18. November 2013)] **Introducing Event Storming**. Von <http://ziobrando.blogspot.de/2013/11/introducing-event-storming.html> abgerufen

C

- [Cockburn, A. (2012)] **Hexagonal Architecture**. Von <http://alistair.cockburn.us/Hexagonal+architecture> abgerufen
- [Conway, M. E. (April 1968)] **How do Committees Invent?** Datamation 14.

D

- [Dahan, U. (9. Dezember 2009)] **Clarified CQRS**. Von <http://udidahan.com/2009/12/09/clarified-cqrs/> abgerufen

E

- [Evans, E. (2003)] **Domain-Driven Design: Tacking Complexity In the Heart of Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

F

- [Fowler, M. (2002)] **Patterns of Enterprise Application Architecture**. Addison Wesley.
- [Fowler, M. (14. Juli 2011)] **CQRS**. Von <http://martinfowler.com/bliki/CQRS.html> abgerufen

G

- [Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1994)] **Design Patterns. Elements of Reusable Object-Oriented Software**. Addison-Wesley Professional Computing Series.

H

- [Hofer, S. & Schwentner, H. (2022)] **Domain Storytelling**. Addison-Wesley.

- [Hruschka, P. (2019)] **Business Analysis und Requirements Engineering**. Carl Hanser, München.

K

- [Kühl, S. & Muster, J. (2018)] **Designing Organizations: A Very Brief Introduction**. Organizational Dialogue Press.

L

- [Liebherr, K., Holland, I., & Riel, A. (1988)] **Object-oriented programming: an objective sense of style**. OOPSLA '88 Conference proceedings on Object-oriented programming systems, languages and applications. ACM New York.
- [Lilienthal, C. (2019)] **Langlebige Software-Architekturen**. dpunkt.verlag.
- [Lilienthal, C. & Schwentner, H. (2023)] **Domain-Driven Transformation**. dpunkt.verlag.

M

- [Manns, M. L. & Rising, L. (2015)]. **More Fearless Change**. Boston: Addison-Wesley.
- [Martin, R. C. (2002)] **Agile Software Development**, Principles, Patterns, and Practices. Prentice Hall Computer.

N

- [North, D. (3. März 2016)] **Introducing BDD**. Von <http://dannorth.net/introducing-bdd/> abgerufen

O

- [Osterwalder, A. & Pigneur, Y. (2010)] **Business Model Generation**. Wiley.
- [Osterwalder, A. (2013)] **Value Proposition Design: How to Create Products and Services Customers Want**. Wiley.

P

- [Patton, J. (2014)] **User Story Mapping: Discover the Whole Story, Build the Right Product**. Sebastopol, CA: O'Reilly.

S

- [Skelton M. & Pais, M. (2019)] **Team Topologies: Organizing Business and Technology Teams for Fast Flow**. Portland, OR: IT Revolution.
- [Stenberg, J. (24. Februar 2015)] Behaviour-Driven Development Combined with Domain-Driven Design. Von <http://www.infoq.com/news/2015/02/bdd-ddd> abgerufen

V

- [Vernon, V. (2013)] **Implementing Domain-Driven Design**. Addison-Wesley.
- [Vernon, V. (2016)] **Domain-Driven Design Distilled**. Addison-Wesley.
- [Vernon, V. (2017)] **Domain-Driven Design kompakt**. dPunkt.

W

- [Wardley, S. (2020)] **Wardley Maps: Topographical Intelligence in Business**. Von <https://github.com/HiredThought/wardley-maps-ebook/raw/main/bin/Wardley%20Maps%20-%20Simon%20Wardley.pdf> abgerufen
- [Wynne, M. (8. Dezember 2015)] **Introducing Example Mapping**. Von <https://cucumber.io/blog/bdd/example-mapping-introduction/> aufgerufen.

Z

- [Züllighoven, H. (1998)] **Das objektorientierte Konstruktionshandbuch - nach dem Werkzeug und Material-Ansatz**. dpunkt.