Curriculum für

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Modul
DDD

Domain-Driven Design

2017.1-DE-20220520

Inhaltsverzeichnis

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

[1] Vgl.: Continuous Integration [evans]
[1] Vgl.: Continuous Integration [Evans, E. (2003)]

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 Experten 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 TeilnehmerInnen Domain Driven Design (DDD) als Mittel, Software als präzise, transparente und transformierbare Repräsentation einer fachlichen Domäne zu gestalten.

Am Ende des Moduls kennen die TeilnehmerInnen 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 Fachexperten und Entwicklern 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 TeilnehmerInnen Methoden des Domain Driven Designs, um mit der wachsende Komplexität eines großen Softwareprojektes umzugehen.

Struktur des Lehrplans und empfohlene zeitliche Aufteilung

Inhalt	Empfohlene Mindestdauer (min)
1. Domäne, Modell und Ubiquitous Language	195
2. Der Weg zum Modell	240
3. Vom Modell zur Implementierung	120
4. Das Modell in der Anwendungsarchitektur	165
5. Modelle schneiden und voneinander abgrenzen	210
6. Lokale Modellkonsistenz wahren	90
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 Software-Entwicklung
- Erfahrungen bei der Modellierung von objektorientierten Architekturen

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Wissen über agile Methoden der Software-Entwicklung, wie z. B. Scrum, Kanban, XP, etc.
- Erfahrung bei der Zusammenarbeit von Fachbereich und Software-Entwicklern.

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 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 (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 Domänen-Experten und Entwickler dem wechselseitigen Verständnis dient.
- Die TN verstehen den Begriff der Ubiquity (Allgegenwärtigkeit): Alle Stakeholder verstehen und nutzen eine einzige, von ihnen entwickelte Fachsprache zur Kommunikation im Rahmen des Projekts.

LZ 1-3: Rolle der Fachsprache bei der Konstruktion einer Ubiquitous Language verstehen

- Die TN verstehen, dass die grundlegenden Begriffe einer Ubiquitous Language direkt der Fachsprache der Domänen-Experten entspringen.

LZ 1-4: 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 [2] an angemeldete Klienten propagiert.

LZ 1-5: 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.[3]

1.3. Referenzen

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

[2] Vgl.: Observer [Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1994)]
[3] Vgl.: Übersicht auf Seite 65 [Evans, E. (2003)]

2. Der Weg zum Modell

Dauer: 180 Min.	Übungszeit: 60 Min.

2.1. Begriffe und Konzepte

- Empowerment der Domänen-Experten
- Kollaborationsformen zur Modell-Erhebung
- Werkzeuge zur Modell-Erhebung

2.2. Lernziele

In diesem Block lernen die Teilnehmer, wie sie gemeinsam mit Domänen-Experten ein Domänenmodell erstellen und verfeinern können.

LZ 2-1: Den hohen Stellenwert von Domänen-Experten in DDD kennen und erläutern können

- Die TN können Domänen-Experten 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 Domänen-Experten vorhanden ist.
- Die TN verstehen, dass implizites Domänen-Wissen offengelegt und festgehalten werden muss, damit es modelliert werden kann.
- Die TN sind in der Lage, Vertrauen bei Domänen-Experten in das Modell und in die gleichberechtigte Zusammenarbeit zu schaffen.
- Die TN können Projektsponsoren und Product Ownern die Notwendigkeit für die Mitgestaltung durch die Domänen-Experten vermitteln.

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

- Die TN kennen Einflussfaktoren auf die Eignung von Domänen-Experten 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 Domänen-Experten kommunizieren können

- Die TN beherrschen Modelle um mit Domänen-Experten 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 Domänen-Experten bewusst und unbewusst über Domänenwissen verfügen.

LZ 2-4: Modellierungstechniken bei der Zusammenarbeit mit Domänen-Experten einsetzen können

- Die TN beherrschen die Verwendung von Klassen- und Objektdiagrammen zur Darstellung von Domänenmodellen.
- Die TN beherrschen szenariobasierte Modellierungstechniken:
    - Use-Cases
    - User-Stories zu 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 das Gespräch auch unbewusst beeinflusst.
    - die Auswahl der Fragen bestimmt, was zur Sprache kommt
    - der Interviewer hat eigenen Annahmen, die der Interview-Partner nicht kennt
    - 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 der Interview-Partner nachvollziehen kann.

LZ 2-6: Beobachtung als Mittel zur Modellierung beherrschen

- Die TN können die Beobachtungstechniken "Feldbeobachtung" und "Apprenticing"[4] als Mittel zur Offenlegung unbewussten Domänenwissens anwenden:
    - Feldbeobachtung
        - Analyse als stiller Beobachter
        - Arbeitsabläufe werden schriftlich erfasst
        - Verständnisfragen an den Domänen-Experten in seltenen Fällen
    - Apprenticing
        - Ist eine Erweiterung der Feldbeobachtung
        - Nach einer Reihe von Beobachtungsdurchläufen übernehmen die Beobachter exemplarisch einen kleinen Teil der Arbeit des Domänen-Experten

LZ 2-7: Einen Event-Storming-Workshop durchführen können

- Die TN können einen Event-Storming-Workshop (Brandolini, 2013) vorbereiten, moderieren, und nachbereiten.
- Die TN verstehen, dass Event-Storming-Workshops ohne strenge Koordination stattfinden.
- Die TN verstehen, dass Event Storming sich auf die Erhebung von Domain Events konzentriert.
- Die TN können die ungeordnete Information aus dem Workshop strukturieren.

LZ 2-8: Ein geeignetes Vorgehen zur Modellierung auswählen und mit Domänen-Experten diskutieren können

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

2.3. Referenzen

[Hruschka, P. (2014)], [Schulz von Thun, F. (2010)], [Schulz von Thun, F. (2013)]

[4] Vgl.: Kapitel 10 [Hruschka, P. (2014)]

3. Vom Modell zur Implementierung

Dauer: 60 Min.	Übungszeit: 60 Min.

3.1. Begriffe und Konzepte

- Kohäsion und Kopplung
- SOLID
- Zyklenfreiheit
- Law of Demeter
- CRC-Karten

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.
- Die TN beherrschen die Technik CRC-Karten als Hilfsmittel bei der Modellierung.

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 aktualisieren 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)], [Martin, R. C. (2002)], [Lilienthal, C. (December 2015)], [Liebherr, K., Holland, I., & Riel, A. (1988)], [Beck, K., & Cunningham, W. (1989)]

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)
- Werkzeug- und Materialansatz (WAM)
- Behaviour Driven Development (BDD)

4.2. Lernziele

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

LZ 4-1: Ausgewählte Architekturstile beherrschen und ein Domänenmodell integrieren können

- Die TN können um das Domänenmodell eine hexagonale Architektur entwerfen.
  - **Ports** an den Außenseiten des inneren Hexagons bieten Schnittstellen zur Kommunikation zwischen fachlichem Kern und seiner Umgebung.
  - **Adapter**[5] verbinden die Ports mit konkreten Drittsystemen über verschiedene Kommunikationskanäle wie REST-APIs, SOAP, Message Queues oder Db-Connections.
  - Die Applikation liegt im Inneren des Hexagons mit dem Domänenmodell als Kern.
  - Bei der inneren Modellierung liegt der Fokus auf der Realisierung fachlicher Anforderungen mit Hilfe des Domänenmodells.
  - In der äußeren Modellierung liegt der Fokus auf der Bereitstellung fachlicher Operationen und Daten über plattformunabhängige Schnittstellen.
  - Das Domänenmodell treibt die Modellierung externer Schnittstellen.
  - Ports verwenden Transfer-Klassen anstatt Entities aus dem Domänenmodell. Die TN verstehen, dass dies Versionskonflikte an den Schnittstellen bei änderungen des Modells entgegenwirkt.
- Die TN können das Domänenmodell mit dem Command-Query Responsibility Segregation Pattern (CQRS) einsetzen.
  - Aufteilen des Domänenmodells in Query- und Command Model
  - Repositories und Aggregrates für die jeweilige Verarbeitung von Commands oder Queries
  - Domain Events als Synchronisationsmittel zwischen Query- und Command Model
- Die TN können Software-Entwicklern die Unterschiede von CQRS, einer hexagonalen Architektur und einer Schichtenarchitektur vermitteln.

LZ 4-2: Zusammenhänge und Abgrenzungen zwischen DDD, WAM 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)[6]
- Die TN können die Bausteine von DDD mit den Bausteinen von WAM vergleichen.

4.3. Referenzen

[Vernon, V. (2013)], [Evans, E. (2003)], [Cockburn, A. (2012)], [Dahan, U. (9. Dezember 2009)], [Lilienthal, C. (December 2015)], [Züllighoven, H. (1998)], [North, D. (3. März 2016)]

[5] Vgl.: Adapter [Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1994)]
[6] Vgl.: [Stenberg, J. (24. Februar 2015)]

5. Modelle schneiden und voneinander abgrenzen

Dauer: 150 Min.	Übungszeit: 60 Min.

5.1. Begriffe und Konzepte

- Modellkonsistenz
- Bounded Context, Context Map
- Shared Kernel, Customer/Supplier Teams, Open Host Service [7]
- Domain Event

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 verschieden 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 Entwickler-Team bewältigen.
  - Das Modell wirkt widersprüchlich.

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

- Die TN kennen Conway's Law [8] und können es auf die (Kommunikations-)Struktur von Entwicklungs-Teams und Software-Artefakten anwenden.
- 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: 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.

LZ 5-4: 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, Modell-übersetzungen 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 5-5: 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 5-6: Ein System als Open Host Service (OHS) konzipieren können

- Die TN verstehen, dass ein OHS die fachlichen übersetzungs-Schichten 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 übersetzungs-Schicht benötigen.

LZ 5-7: 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 [9] einzusetzen, um es den Subscribern zu ermöglichen, fehlerhaft verarbeitete Events erneut zu verarbeiten.

5.3. Referenzen

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

[7] Vgl.: Übersicht auf Seite 388 [Evans, E. (2003)]
[8] Cf.: [Conway, M. E. (April 1968)]
[9] Vgl.: Seite 539 [Vernon, V. (2013)]

6. Lokale Modellkonsistenz wahren

Dauer: 60 Min.	Übungszeit: 30 Min.

6.1. Begriffe und Konzepte

- Anticorruption-Layer
- Continuous Integration
- Separate Ways [7]

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: Verstehen, wie Continuous Integration (CI) [1] zu lokaler Modellkonsistenz beiträgt

- Die TN kennen die Vorteile davon, CI innerhalb eines Bounded Contexts stattfinden zu lassen, und können dies von Continuous Deployment abgrenzen.
- Die TN verstehen, dass Continuous Integration einer Fragmentierung des Modells durch verschiedene Entwickler entgegenwirkt.
- Die TN verstehen, dass CI die Kommunikation über das Modell und das Verständnis des Modells durch das Team fördert.

LZ 6-2: 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 [10] einsetzen, um ein Anticorruption Layer zu konstruieren.

LZ 6-3: Umstände verstehen, in denen eine Aufteilung des Modells sinnvoll ist (Separate Ways), unter Berücksichtigung der Aspekte aus Block 5

- 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.

6.3. Referenzen

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

[10] Vgl.: [Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1994)]

Referenzen

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

A

- [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.
- [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. (November 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

- [Hruschka, P. (2014)] Business Analysis und Requirements Engineering. Carl Hanser, München.

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. (December 2015)] Langlebige Software-Architekturen. dpunkt.verlag.

M

- [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

S

- [Schulz von Thun, F. (2010)] Miteinander Reden: 1. Rowohlt Taschenbuch Verlag.
- [Schulz von Thun, F. (2013)] Miteinander Reden: 3. Rowohlt Taschenbuch Verlag.
- [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 Longman Publishing Co., Inc.

Z

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