Curriculum for

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Module
DDD

Domain-Driven Design

2023.1-rev0-EN-20230510

Table of Contents

List of Learning Goals

- LG 1-1: Know and be able to explain the connections between domains, software, and models
- LG 1-2: Understand the role of domain-specific terminology in the construction of a ubiquitous language
- LG 1-3: Know and be able to explain the building blocks of domain-driven design
- LG 1-4: Know and be able to explain the connections between the building blocks
- LG 2-1: Know and be able to explain the high importance of domain experts in DDD
- LG 2-2: Be able to provide support in selecting suitable contact persons
- LG 2-3: Be able to communicate with domain experts
- LG 2-4: Be able to use modeling techniques when working with domain experts
- LG 2-5: Be able to conduct interviews to model a domain
- LG 2-6: Be proficient in observation to understand a domain
- LG 2-7: Get an overview of Collaborative Modeling, its methods, and how it relates to DDD.
- LG 2-8: Be able to select a suitable modeling approach and discuss it with domain experts
- LG 2-9: Be able to conduct a collaborative modeling workshop
- LG 2-10: Understand agility as a foundation of DDD
- LG 3-1: Be able to extend a domain model with technical building blocks from DDD
- LG 3-2: Be able to model interfaces for domain classes
- LG 3-3: Know and be able to take into account interactions between an implementation and its model
- LG 3-4: Be able to argue why DDD is worthwhile for complex business logic
- LG 4-1: Be able to design a ports & adapter architecture for the domain model
- LG 4-2: Be able to formulate correlations and distinctions between DDD and BDD
- LG 5-1: Know symptoms that can occur with excessively large models
- LG 5-2: Be able to assess advantages and disadvantages of a cross-team model
- LG 5-3: Be able to move on from problem to solution space
- LG 5-4: Distill the core
- LG 5-5: Be able to describe model boundaries of Bounded Contexts in a Context Map
- LG 6-1: Be able to use interfaces for customer/supplier teams
- LG 6-2: Be able to design a system as an open host service (OHS)
- LG 6-3: Be able to isolate your own model from external influences
- LG 6-4: Be able to reuse core elements of several partial models in a shared kernel
- LG 6-5: Understand the circumstances in which it is appropriate to divide the model (Separate Ways)
- LG 6-6: Be able to use Domain Events as a means of communication between Bounded Contexts

Introduction: General information about the iSAQB Advanced Level

What is taught in an Advanced Level module?

- The iSAQB Advanced Level offers modular training in three areas of competence with flexibly designable training paths. It takes individual inclinations and priorities into account.
- The certification is done as an assignment. The assessment and oral exam is conducted by experts appointed by the iSAQB.

What can Advanced Level (CPSA-A) graduates do?

CPSA-A graduates can:

- Independently and methodically design medium to large IT systems
- In IT systems of medium to high criticality, assume technical and content-related responsibility
- Conceptualize, design, and document actions to achieve quality requirements and support development teams in the implementation of these actions
- Control and execute architecture-relevant communication in medium to large development teams

Requirements for CPSA-A certification

- Successful training and certification as a Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- At least three years of full-time professional experience in the IT sector; collaboration on the design and development of at least two different IT systems
  - Exceptions are allowed on application (e.g., collaboration on open source projects)
- Training and further education within the scope of iSAQB Advanced Level training courses with a minimum of 70 credit points from at least three different areas of competence
- Successful completion of the CPSA-A certification exam

Essentials

What does the module “DDD” convey?

This module presents domain-driven design (DDD) to course participants as a tool to design software as a precise, transparent, and transformable representation of a domain.

At the end of the module, course participants will know the essential principles of domain-driven design and will be able to apply them when designing and implementing software systems. They are able to use the newly learned communication skills to establish a uniform language between experts and developers. With the help of the modeling techniques and architecture tools that have been taught, they can incorporate the components of this common, domain-specific language into their software systems.

A large software project often requires the involvement of several development teams. This module addresses this challenge and teaches the course participants methods of domain-driven design to handle the growing complexity of a large software project.

Curriculum Structure and Recommended Durations

Content	Recommended minimum duration (minutes)
1. Domain, model, and ubiquitous language	195
2. Knowledge Crunching: The path to the model	240
3. From the model to implementation	120
4. The model in the application architecture	165
5. Strategic Design 1: Cutting and distinguishing models from one another	210
6. Strategic Design 2: Context Mapping	90
7. Optional: Related Topics	
Total	1020 (17h)

Duration, Teaching Method and Further Details

The times stated below are recommendations. The duration of a training course on the DDD module should be at least 3 days, but may be longer. Providers may differ in terms of duration, teaching method, type and structure of the exercises, and the detailed course structure. In particular, the curriculum provides no specifications on the nature of the examples and exercises.

Licensed training courses for the DDD module contribute the following credit points towards admission to the final Advanced Level certification exam:

Methodical Competence:	20 Points
Technical Competence:	0 Points
Communicative Competence:	10 Points

Prerequisites

Participants **should** have the following prerequisite knowledge:

- Fundamentals and advanced concepts of object-oriented software development
- Experience in modeling object-oriented architectures

Knowledge in the following areas may be **helpful** for understanding some concepts:

- Knowledge of agile methods of software development, such as Scrum, Kanban, XP, etc.
- Experience in collaboration between business experts and software developers.

Structure of the Curriculum

The individual sections of the curriculum are described according to the following structure:

- **Terms/principles**: Essential core terms of this topic.
- **Teaching/practice time**: Defines the minimum amount of teaching and practice time that must be spent on this topic or its practice in an accredited training course.
- **Learning goals**: Describes the content to be conveyed including its core terms and principles.

This section therefore also outlines the skills to be acquired in corresponding training courses.

Supplementary Information, Terms, Translations

To the extent necessary for understanding the curriculum, we have added definitions of technical terms to the iSAQB glossary and complemented them by references to (translated) literature.

1. Domain, model and ubiquitous language

Duration: 120 min	Practice time: 75 min

1.1. Terms and Principles

- Domain, domain model
- Ubiquitous language
- Modules
- Entity, Value Object, Aggregate, Service
- Factory, Repository
- Domain Events (Vernon, 2013)

1.2. Learning Goals

This section serves as an introduction to domain-driven design and provides the motivation for its development. The fundamental concepts Domain, Domain Model, and Ubiquitous Language are presented here. The course participants also obtain a detailed insight into the various components of domain models and the relationships between them.

LG 1-1: Know and be able to explain the connections between domains, software, and models

- The course participants can describe the dependency between software and a domain. They understand that software does not exist as an end in itself.
- The course participants understand domain models as a tool for abstracting expert knowledge.
- The course participants understand that domain models represent the ideas and relationships of a domain.
- The course participants can explain the domain model as a tool to align the software with the domain.

LG 1-2: Understand the role of domain-specific terminology in the construction of a ubiquitous language

- The course participants understand that a common language for both domain experts and developers assists mutual understanding.
- The course participants understand the term ubiquity: "Ubiquity" applies in a defined boundary (Bounded Context) within which the language is used consistently in conversations, diagrams, documentation, and code.
- The course participants know that the fundamental terms of a ubiquitous language originate from the domain experts.
- The participants understand that the Ubiquitous Language is a formalized version of the domain language and a conscious design decision.
- The participants understand that the Ubiquitous Language evolves in a dialog between development team and domain experts.

LG 1-3: Know and be able to explain the building blocks of domain-driven design

- The course participants understand the fundamental building blocks of domain-driven design. o Value Objects represent elementary value types from the domain. They can only contain other Value Objects. Value Objects have no identity, but are comparable.
    - Entities represent things in the domain. They can contain Value Objects and other Entities. Entities have an identity.
- The course participants understand the additional building blocks of domain-driven design.
    - **Modules** as static grouping mechanisms for code artifacts.
    - **Services** encapsulate independent functions that cannot be allocated to individual Entities. They decouple functionality from states and should therefore be stateless. They enable, among other things, domain processes to be documented in a domain model.
    - **Aggregates** are used to group Entities. External access exclusively takes place via a globally identifiable Aggregate Root. The Aggregate Root is also responsible for the compliance of invariants within the Aggregate. This achieves a loose coupling when using Aggregates, hides the contained Entities from external access, and facilitates the compliance of invariants. The lifecycle of all contained Entities is thus determined by the lifecycle of the Aggregate Root.
    - **Factories** represent creation mechanisms for Entities. They allow potentially complex logic to be outsourced from the constructors of Entities.
    - **Repositories** are used for the inventory management of Entities at runtime. Determining and issuing a reference to an Entity or Aggregate for a unique identifier falls within their area of responsibility. They can be used to hide interfaces to third-party systems such as databases or remote services.
    - **Domain Events** support the sharing of information about the occurrence of domain-related incidents. Domain Events are triggered by Aggregates or Entities and propagated via a publisher/subscriber pattern (cf. Observer [Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1994)]) to registered clients.

LG 1-4: Know and be able to explain the connections between the building blocks

- The course participants are able to create a relationship between the building blocks and combine them in a sensible manner (cf. overview on page 65 [Evans, E. (2003)]).

1.3. References

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

2. Knowledge Crunching: The path to the model

Duration: 180 min	Practice time: 60 min

2.1. Terms and Principles

- Agile and evolutionary modeling
- Empowerment of domain experts
- Forms of collaboration for model analysis
- Tools for model analysis

2.2. Learning Goals

LG 2-1: Know and be able to explain the high importance of domain experts in DDD

- The course participants can teach domain experts that DDD offers them responsibility and creative possibilities (empowerment).
- The course participants are able to teach a development team that software must support the existing domain.
- The course participants understand that knowledge about the domain primarily exists in the minds of the domain experts.
- The course participants understand that implicit domain knowledge must be revealed and recorded so that it can be modeled.
- The course participants are able to build trust with domain experts regarding the model as well as working together as equals.
- The course participants are able to communicate to project sponsors and product owners about the need for active participation from the domain experts.

LG 2-2: Be able to provide support in selecting suitable contact persons

- The course participants know influencing factors concerning the suitability of domain experts and can assess them based on their:
  - comprehensive experience and knowledge in their own domain;
  - motivation;
  - availability during and after the project;
  - ability to think abstractly; and
  - flexibility.

LG 2-3: Be able to communicate with domain experts

- The course participants are proficient in different communication models in order to enter into an equitable dialog with domain experts:
  - the “Communication Square” or “Four Sides” (German: “Kommunikationsquadrat” or “Vier-Seiten”)
  - Four-Ears (German: “Vier Ohren”)

    - the Inner Team (German: „das Innere Team“)
- The course participants understand that the communication between domain experts and development teams in DDD is of critical importance.
- The course participants understand that domain experts may consciously or unconsciously possess domain knowledge.

LG 2-4: Be able to use modeling techniques when working with domain experts

- The course participants are proficient in the use of class and object diagrams to depict domain models.
- The course participants are proficient in scenario-based modeling techniques:
    - Use cases
    - User stories for subsequent modeling of Domain Events
    - Domain Events
- The course participants can create glossaries for the terms of a ubiquitous language.

LG 2-5: Be able to conduct interviews to model a domain

- The course participants understand that interviews are suitable for revealing domain knowledge.
- The course participants understand that the interviewer also unconsciously influences the conversation.
    - The selection of questions determines what is discussed
    - The interviewer makes their own assumptions, which the interview partner is not aware of
    - Confirmation bias can lead to misunderstandings
- The course participants are proficient at structuring and conducting an interview relating to a concrete scenario from the domain.
- The course participants understand that it is helpful to create a model that the interview partner can understand during an interview.

LG 2-6: Be proficient in observation to understand a domain

- The course participants can apply the observation techniques of “field observation” and “apprenticing” (cf. [Hruschka, P. (2019)]) as tools for revealing unconscious domain knowledge:
    - Field observation
        - Analysis as a silent observer
        - Working processes are recording in writing
        - In rare cases, questions to the domain experts to verify comprehension
    - Apprenticing
        - Is an expansion of field observation
        - Following a series of observation cycles, the observers performs a small, but exemplary part of the domain expert’s work

LG 2-7: Get an overview of Collaborative Modeling, its methods, and how it relates to DDD.

- The participants know that there are different methods, e.g.:
  - EventStorming (Brandolini, 2021).
  - Domain Storytelling (Hofer, Schwentner 2022).
  - User Story Mapping (Patton, 2015).
- Participants understand the common values of the different methods:
  - Develop common understanding through close collaboration of different stakeholders: domain experts, developers, and others.
  - Lightweight methodology whose elements can be explained in five minutes to get started.
  - Focus on behavior, not only on structure/data (not only nouns, but also verbs).
- Participants understand that Collaborative Modeling can be used independently of DDD.

LG 2-8: Be able to select a suitable modeling approach and discuss it with domain experts

- The course participants can identify and address organizational constraints:
  - Technical and room resources
  - Geographically distributed domain experts
  - Legal constraints for the production of transcripts, audio/video streams, photos, etc.
- The course participants can discuss with the domain experts whether the model should be developed iteratively or in advance.
- The course participants can discuss the consequences of vagueness and misperceptions in the model with domain experts and developers.

LG 2-9: Be able to conduct a collaborative modeling workshop

- Participants are able to prepare, facilitate and follow up a workshop in at least one of the above mentioned methods.
- Participants are able to invite suitable participants to the workshop.
- Participants are able to structure the unorganized information from the workshop.
- The participants know that online and on-premise must be approached differently.

LG 2-10: Understand agility as a foundation of DDD

- Participants can relate DDD to the values and principles from the Agile Manifesto.
- Participants have understood that DDD relies on an evolutionary design:
  - A model is always evolving
  - Sustainable models are created through an exploratory approach.

2.3. References

[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. From the model to the implementation

Duration: 60 min	Practice time: 60 min

3.1. Terms and Principles

- Cohesion and coupling
- SOLID
- Avoidance of cyclical dependencies
- Law of Demeter

3.2. Learning Goals

This section teaches methods and approaches to derive the corresponding domain classes from a domain model.

LG 3-1: Be able to extend a domain model with technical building blocks from DDD

- The course participants specify additional components for an existing domain model consisting of Entities, Value Objects, and Services:
  - Factories to create Entities
  - Repositories to manage Entities
  - Aggregates to encapsulate Entities and Value Types

LG 3-2: Be able to model interfaces for domain classes

- The course participants can assess and apply the design principles and heuristics from the Foundation Level to the DDD design.

LG 3-3: Know and be able to take into account interactions between an implementation and its model

- The course participants understand that changes in the domain-specific terminology or in the model must be followed by corresponding changes in the software.
- The course participants understand that changes to the implementation of the model, such as due to the refactoring of Services or Repositories, are an indication that the domain model should be updated.

LG 3-4: Be able to argue why DDD is worthwhile for complex business logic

- The course participants know alternative approaches and can explain the advantages of DDD for complex business logic.
  - Table Module, Transaction Script and Domain Model (Fowler, Patterns of Enterprise Application Architecture, 2002)
  - Smart UI (Evans, 2003)

3.3. References

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

4. The model in the application architecture

Duration: 105 min	Practice time: 60 min

4.1. Terms and Principles

- 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. Learning Goals

This section uses selected examples to teach how a domain model can be integrated into software architectures. On the basis of examples, it teaches the similarities and differences between domain-driven design and two related methods.

LG 4-1: Be able to design a ports & adapter architecture for the domain model

- The course participants can explain the differences between a ports&adapter architecture (e.g., Hexagonal, Clean, or Onion) and a layered architecture.

LG 4-2: Be able to formulate correlations and distinctions between DDD and BDD

- The course participants know specifications from BDD and can relate them to the ubiquitous language. They understand that specifications can be formulated with the terms of the ubiquitous language.
- The course participants understand that BDD formulates requirements with the help of the user interface (outside-in), and DDD expresses requirements via the domain model (inside-out), cf. [Stenberg, J. (24. Februar 2015)].

4.3. References

[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: Cutting and distinguishing models from one another

Duration: 150 min	Practice time: 60 min

5.1. Terms and Principles

- Problem Space and Solution Space
- Subdomain
- Bounded Context
- Core vs supporting vs generic
- Context Map

5.2. Learning Goals

This section teaches the fundamentals of decomposing a model into several sub-models and making the model manageable using defined model boundaries. It also teaches strategies for distributing multiple models and their resulting responsibilities to various teams. Solutions are highlighted that allow interfaces to be created between models.

LG 5-1: Know symptoms that can occur with excessively large models

- The course participants can identify the following circumstances as symptoms:
  - Various user groups use the same terms in different ways.
  - The model can no longer be managed by a developer team. o The model appears inconsistent.

LG 5-2: Be able to assess advantages and disadvantages of a cross-team model

- The course participants know Conway's Law [Conway, M. E. (April 1968)] and its relevance for software architecture.
- The course participants understand that models used by several teams require cross-team communication and coordination (e.g., for deployment).
- The course participants understand that models need to be partitioned so that several development teams can work independently of one another.
- The course participants understand that specified model boundaries play an important role in keeping the communication effort low.
- The course participants know that terms in the domain-specific terminology can have different meanings in different contexts.
- The course participants know that local model consistency can be maintained independently of other models.

LG 5-3: Be able to move on from problem to solution space

- The participants know the terms "problem space" and "solution space" and the difference.
- The participants understand in the solution space "model" and "design" and how tio build them.

- The participants understand how in the problem space the domain can be understood and divided into subdomains.
- The participants understand that language is context dependent and language is an important driver for deriving bounded contexts from subdomains.
- The participants understand how bounded contexts can be derived from subdomains.

LG 5-4: Distill the core

- The participants know that large systems cannot be designed equally well in all areas and that it is therefore important to know in which areas more emphasis must be placed on the design.
- The participants know the terms core, generic, supporting subdomain.

LG 5-5: Be able to describe model boundaries of Bounded Contexts in a Context Map

- The course participants can show the relationship between several Bounded Contexts as a Context Map.
- The course participants understand that every model has a context, even if it is not explicit.
- The course participants understand that the terms of the ubiquitous language only have meaning within their context.
- The course participants understand that, when different Bounded Contexts interact, the building blocks of one context need to be translated into the other context.
- The course participants understand the benefits of the correlation between the model boundaries, the organizational structure, and the source code.

5.3. References

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

6. Strategic Design 2: Context Mapping

Duration: 60 min	Practice time: 30 min

6.1. Terms and Principles

- Shared Kernel, Customer/Supplier Teams, Open Host Service (cf. overview on page 388 [Evans, E. (2003)])
- Domain Event as means of communication
- Anticorruption-Layer
- Separate Ways [Vernon, V. (2013)]

6.2. Learning Goals

This section teaches approaches with which the consistency within a model can be ensured as well as when it is necessary to outsource parts of the model.

LG 6-1: Be able to use interfaces for customer/supplier teams

- The course participants can assess whether two teams are located in one customer/supplier constellation.
- The course participants know the circumstances for the successful collaboration of customer/supplier teams:
  - Stability and documentation of the interface are important for integration
  - Jointly developed acceptance tests help to stabilize the interface
  - The prerequisite for a functioning relationship is that the supplier has an interest in the customer actually using the interface
- The course participants know that, in a customer relationship, the customer formulates the requirements on the supplier's interface.

LG 6-2: Be able to design a system as an open host service (OHS)

- The course participants understand that an OHS replaces the domain translation layers of the clients.
- The course participants can design services for an OHS. They are able to distinguish between essential and specific requirements and take this into account in the design.
- The course participants understand that a public interface can be extended selectively for specific requirements of individual clients, without impairing the interface of other clients.
- The course participants understand that an OHS is particularly worthwhile if several customers require a translation layer.

LG 6-3: Be able to isolate your own model from external influences

- The course participants can identify the possible influences that a model of a neighboring system can have on their own model.
- The course participants can apply the Interface Patterns (cf. [Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1994)]) to construct an Anti-Corruption Layer.

LG 6-4: Be able to reuse core elements of several partial models in a shared kernel

- The course participants can design and assess the suitability of a shared kernel for a specific situation.
- The course participants understand that a shared kernel helps to avoid model translations.
- The course participants understand that a shared kernel requires a high degree of coordination among the teams involved.
- The course participants understand that the teams are equally qualified to work on the shared kernel.

LG 6-5: Understand the circumstances in which it is appropriate to divide the model (Separate Ways)

- The course participants can contrast the coordination costs of a common model with the overheads for separate models.
- The course participants understand that a model must be divided along a sensible boundary.
- The course participants understand that, in these cases, local model consistency is easier to maintain in two partial models.
- The course participants are able to duplicate the remaining functions or integrate them with each other.

LG 6-6: Be able to use Domain Events as a means of communication between Bounded Contexts

- The course participants understand that Domain Events uncouple the Subscriber from the Publisher.
  - In particular, the course participants understand that, for the Publisher, this means that it is not relevant for him who processes his events.
- The course participants are able to create Domain Events as a means of communication between Bounded Contexts.
- The course participants are able to assess opportunities and risks of Domain Events in this context:
  - If events are used despite existing dependencies (e.g., within a Bounded Context), there is a risk that the dependencies will only be hidden, which could lead to control flows that are difficult to understand at runtime.
- The course participants are able to use an event store (cf.: page 539 [Vernon, V. (2013)]) to allow Subscribers to reprocess events that were incorrectly processed.

6.3. References

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

7. Optional: Related Topics

Duration: free	Practice time: free

Around DDD there are many topics that are related and can also be taught by the trainer as needed.

7.1. Recommendations

- "Refactoring towards deeper insights" should be illustrated with the case study
- "Large Scale Structures" by extending the case study

7.2. Terms, Principles, possible Ideas

- Implementation:
  - Demonstrate a concrete implementation example
  - CRC Cards
- Architecture:
  - Tools & Materials Approach
  - Event Sourcing
  - CQRS
  - Modernizing Legacy-Software with DDD
- Organization and strategy:
  - Team Topologies
  - Wardley Mapping
  - System approaches for organizations
  - Business Model Canvas & Value Proposition
- Communication:
  - How to win colleagues and managers for DDD?

7.3. References

[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)]

References

This section contains references that are cited in the curriculum.

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.