CONTEXT MAPPER: DOMAIN-SPECIFIC LANGUAGE AND TOOLS FOR STRATEGIC DOMAIN-DRIVEN DESIGN, CONTEXT MAPPING AND BOUNDED CONTEXT MODELING

MODELSWARD 2020
February 27, 2020

Prof. Dr. Olaf Zimmermann
Stefan Kapferer
HSR FHO

ozimmerm@hsr.ch
stefan.kapferer@hsr.ch

MODELSWARD 2020
8th International Conference on Model-Driven Engineering and Software Development
VALLETTA - Malta 25 - 27 FEBRUARY, 2020

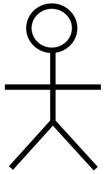
HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Session Outline

- **Context and motivation**
  - User stories for Context Mapper
  - Application integration example
  - Domain-Driven Design (DDD) in a nutshell
- **Proposed Modeling Language and Tools**
  - Domain-Driven Design (DDD) meta model
  - Domain-Specific Language (DSL) core
  - Generation tools
- **Future Work**
  - Context Mapper as an Architecture Recoverer
  - Context Mapper as a (Micro-)Service Decomposer
  - Context Mapper as an Enterprise Portfolio Planning Tool

User Story 1: Business Analyst Modeling Concepts



Business Analyst

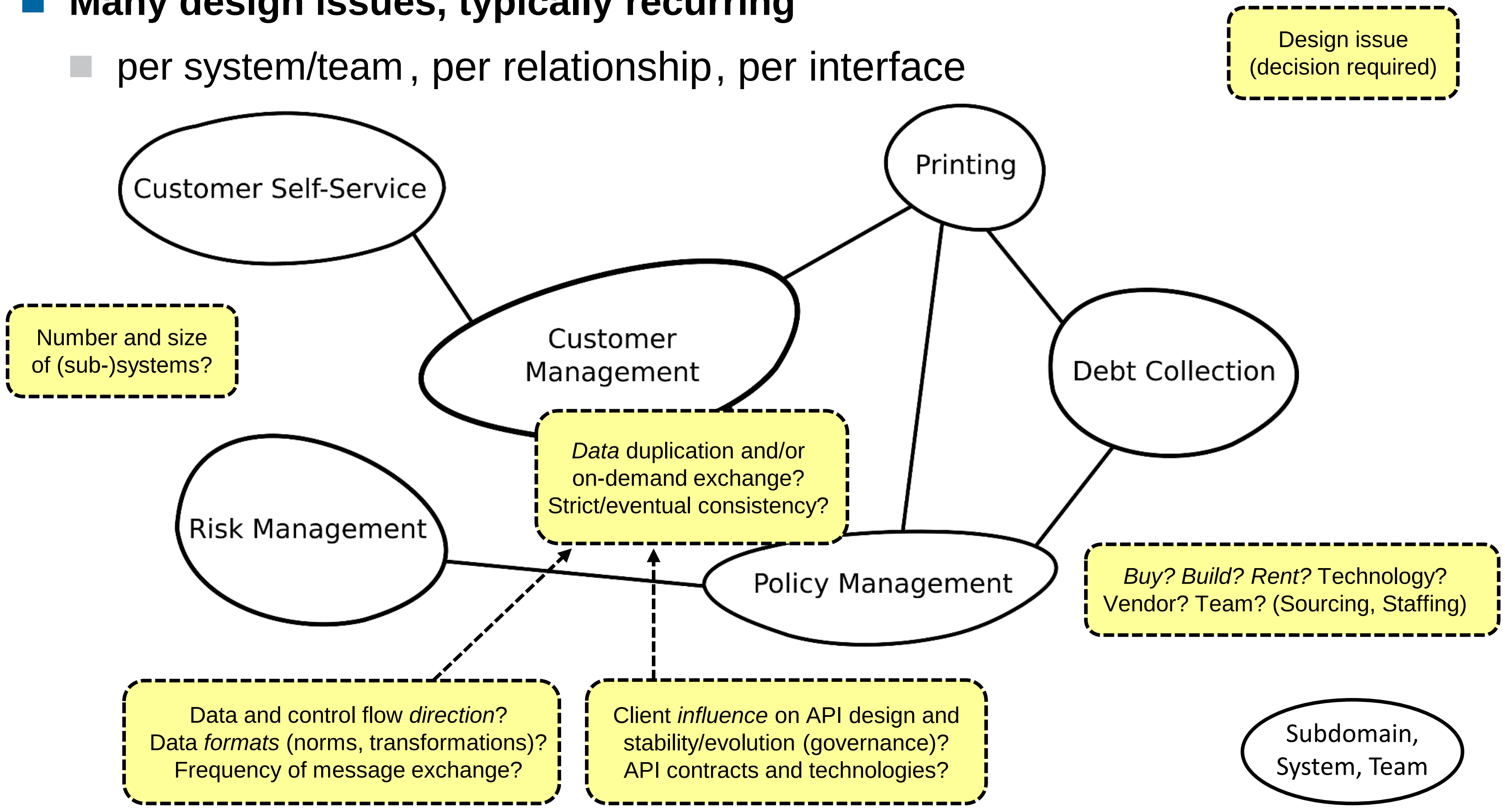
As a **business analyst (specializing on a particular business or technical domain),**

I would like to **describe the problem domain and its subdomains in a natural, yet precise and ubiquitous language (i.e., domain concepts, their properties and relations)**

so that **project sponsor, team and other stakeholders can develop and share a common understanding about these concepts and their intricacies in the given domain – in line with Agile values and principles.**

Motivating Example: “Fictitious” Insurance Application Landscape

- **Many design issues, typically recurring**
  - per system/team, per relationship, per interface


Design issue (decision required)
Customer Self-Service
Printing
Number and size of (sub-)systems?
Customer Management
Debt Collection
Data duplication and/or on-demand exchange? Strict/eventual consistency?
Risk Management
Policy Management
Buy? Build? Rent? Technology? Vendor? Team? (Sourcing, Staffing)
Data and control flow *direction*? Data *formats* (norms, transformations)? Frequency of message exchange?
Client *influence* on API design and stability/evolution (governance)? API contracts and technologies?
Subdomain, System, Team

User Story 2: Software Architect Designing & Deocmposing Services

Software Architect (Service Designer)

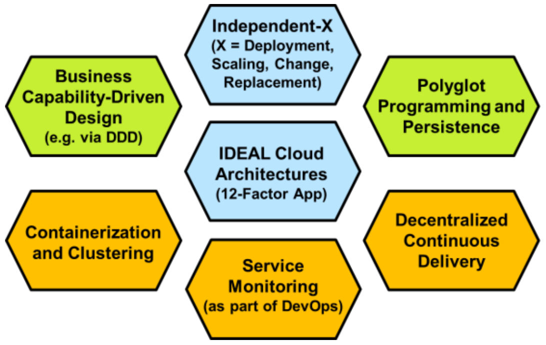
As a **software architect responsible for the design and implementation and integration of an system supporting and partially automating the results of a domain-driven business analysis,**

I would like to **model the subsystems (i.e., Bounded Contexts) and components (Aggregates) of my architecture and how they interact (Interfaces, collaborations)**

so that **I can evolve the architecture semi-automatically (i.e, supported by model refactorings and service decomposition heuristics), communicate the architecture, and generate other representations of the models such as Unified Modeling Language (UML) diagrams and service API contracts (or even code).**

Decomposition Heuristics that do not suffice

- **Two-pizza rule (team size)**
- **Lines of code (in service implementation)**
- **Size of service implementation in IDE editor**


Independent-X (X = Deployment, Scaling, Change, Replacement)
Business Capability-Driven Design (e.g. via DDD)
Polyglot Programming and Persistence
IDEAL Cloud Architectures (12-Factor App)
Containerization and Clustering
Decentralized Continuous Delivery
Service Monitoring (as part of DevOps)

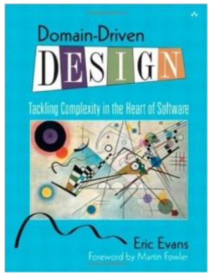
- **Simple if-then-else rules of thumb**
  - E.g. “If your application needs coarse-grained services, implement a SOA; if you require fine ones, go the microservices way” (I did not make this up!)
- **Non-technical traits, including “products not projects”**

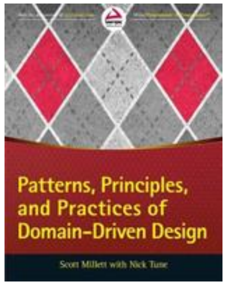
What is wrong with these “metrics” and “best practice” recommendations?

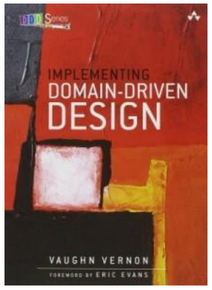
➡ Context matters, as M. Fowler pointed out at Agile Australia 2018 (or: one size does not fit all)

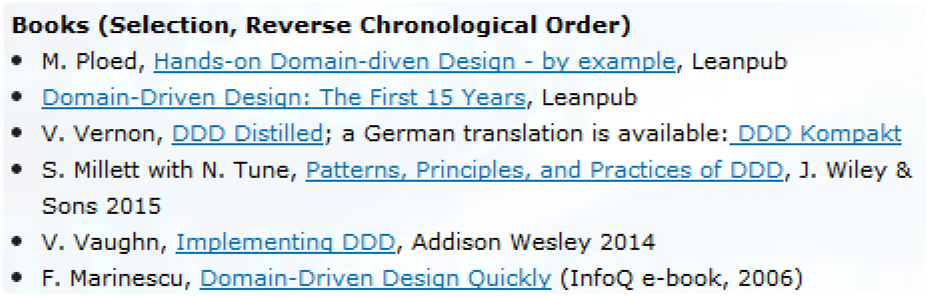
Domain-Driven Design (DDD) to the Remedy

- **Emphasizes need for modeling and communication**
  - Ubiquitous language (vocabulary) – the *domain model*
- ***Tactic DDD* – "Object-Oriented Analysis and Design (OOAD) done right"**
  - Emphasis on business logic in layered architecture
  - Decomposes Domain Model pattern from M. Fowler
  - Patterns for common roles, e.g. Entity, Value Object, Repository, Factory, Service; grouped into *Aggregates*
- ***Strategic DDD* – "agile Enterprise Architecture and/or Portfolio Management"**
  - Models have boundaries
  - Teams, systems and their relations shown in *Context Maps* of *Bounded Contexts*


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler


Patterns, Principles, and Practices of Domain-Driven Design
Scott Millett with Nick Tune

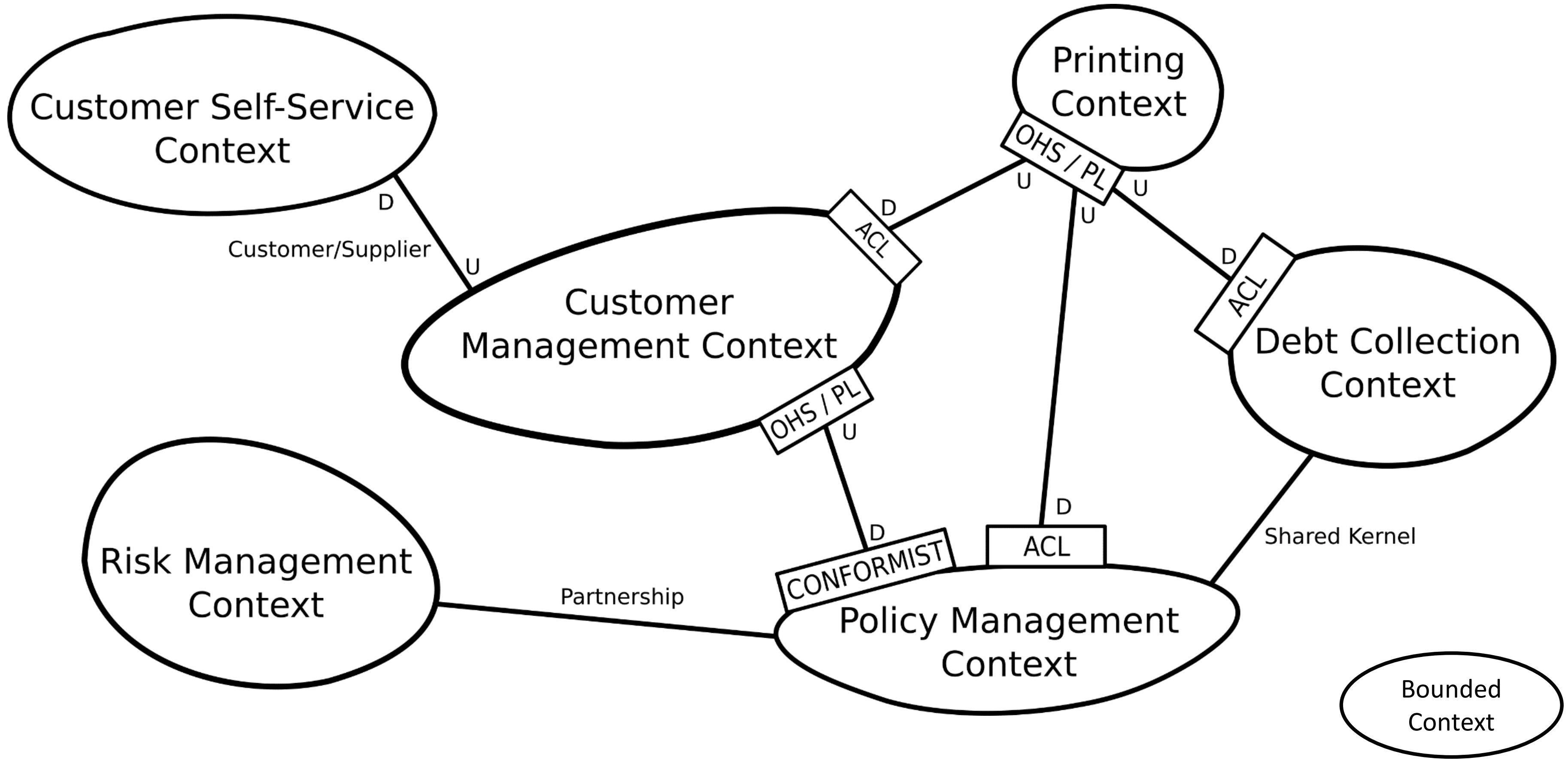



Books (Selection, Reverse Chronological Order)

- M. Ploed, Hands-on Domain-diven Design - by example, Leanpub
- Domain-Driven Design: The First 15 Years, Leanpub
- V. Vernon, DDD Distilled; a German translation is available: DDD Kompakt
- S. Millett with N. Tune, Patterns, Principles, and Practices of DDD, J. Wiley & Sons 2015
- V. Vaughn, Implementing DDD, Addison Wesley 2014
- F. Marinescu, Domain-Driven Design Quickly (InfoQ e-book, 2006)

A Strategic DDD Context Map with Relationships

- **Insurance scenario, example model from https://contextmapper.org/**

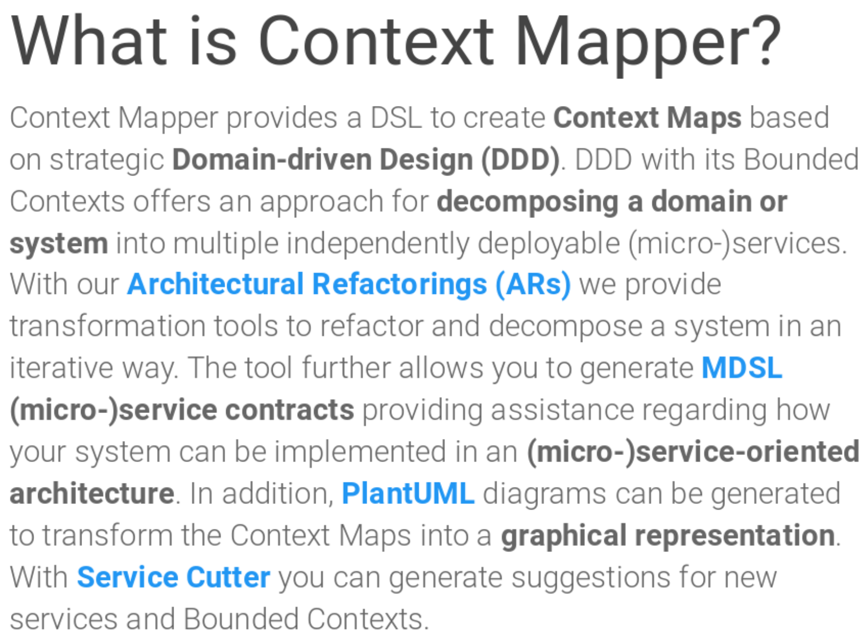


D: Downstream, U: Upstream; ACL: Anti-Corruption Layer, OHS: Open Host Service

Session Outline

- **Context and motivation**
  - User stories for Context Mapper
  - Application integration example
  - Domain-Driven Design (DDD) in a nutshell
- ***Proposed Modeling Language and Tools***
  - *Domain-Driven Design (DDD) meta model*
  - *Domain-Specific Language (DSL) core*
  - *Generation tools*
- **Future Work**
  - Context Mapper as an Architecture Recoverer
  - Context Mapper as a (Micro-)Service Decomposer
  - Context Mapper as an Enterprise Portfolio Planning Tool

Context Mapper: A DSL for Strategic DDD

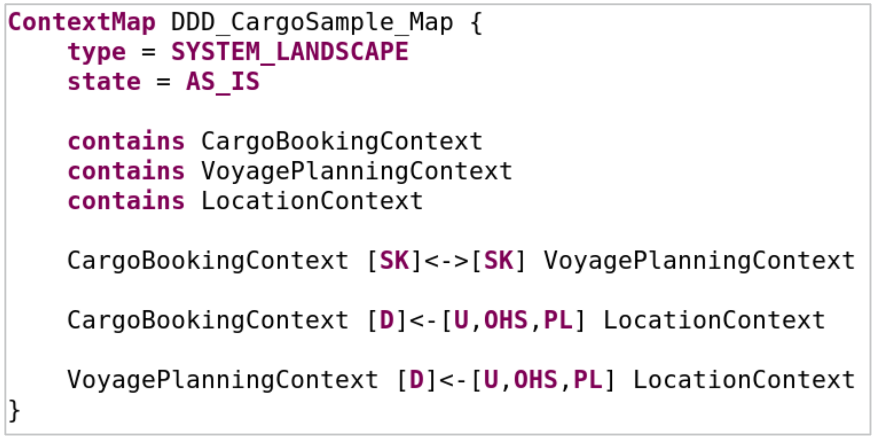


What is Context Mapper?

Context Mapper provides a DSL to create **Context Maps** based on strategic **Domain-driven Design (DDD)**. DDD with its Bounded Contexts offers an approach for **decomposing a domain or system** into multiple independently deployable (micro-)services. With our **Architectural Refactorings (ARs)** we provide transformation tools to refactor and decompose a system in an iterative way. The tool further allows you to generate **MDSL** **(micro-)service contracts** providing assistance regarding how your system can be implemented in an **(micro-)service-oriented architecture**. In addition, **PlantUML** diagrams can be generated to transform the Context Maps into a **graphical representation**. With **Service Cutter** you can generate suggestions for new services and Bounded Contexts.

- **Eclipse plugin, based on:**
  - Xtext, ANTLR
  - Sculptor (tactic DDD DSL)
- **Creator: S. Kapferer**
  - Term projects and Master thesis @ HSR FHO

CONTEXT MAPPER

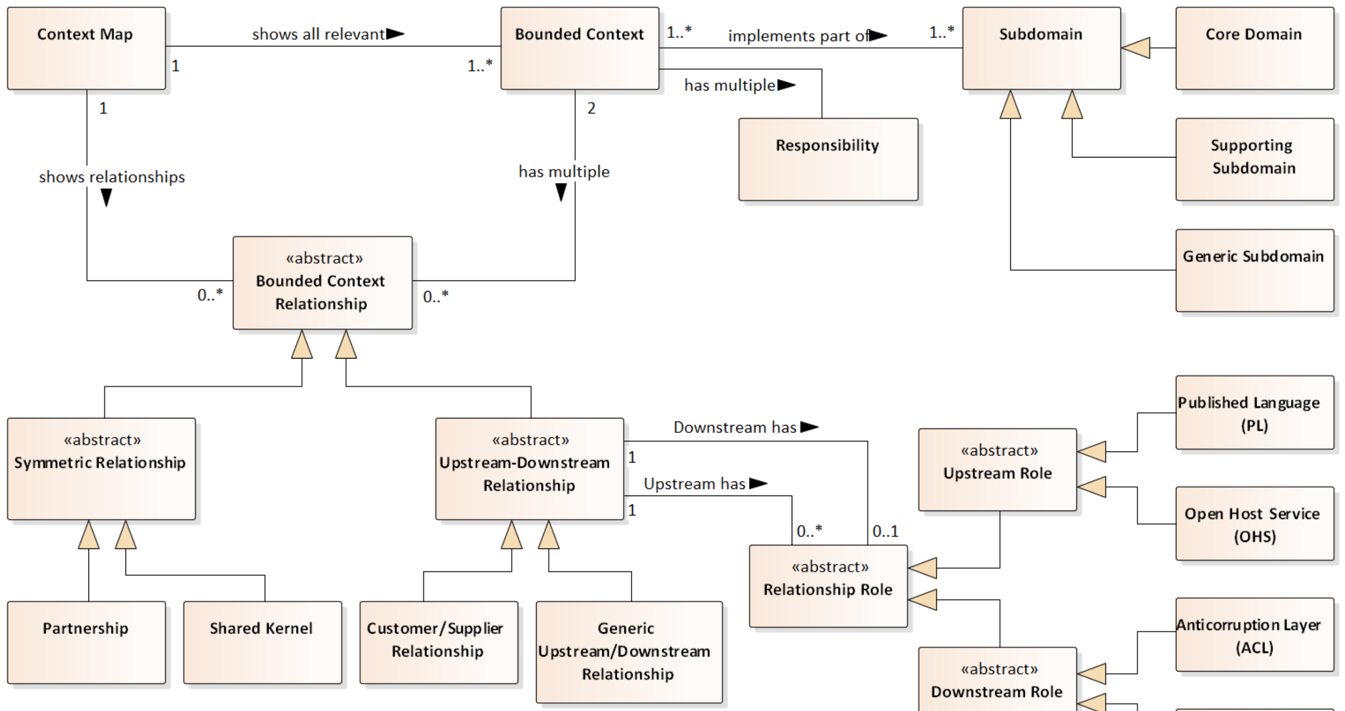


```
ContextMap DDD_CargoSample_Map {
    type = SYSTEM_LANDSCAPE
    state = AS_IS

    contains CargoBookingContext
    contains VoyagePlanningContext
    contains LocationContext

    CargoBookingContext [SK]<->[SK] VoyagePlanningContext

    CargoBookingContext [D]<-[U,OHS,PL] LocationContext

    VoyagePlanningContext [D]<-[U,OHS,PL] LocationContext
}
```

SK: Shared Kernel, **PL:** Published Language
D: Downstream, **U:** Upstream
ACL: Anti-Corruption Layer, **OHS:** Open Host Service

Context Mapper: Meta-Model and Semantic Rules

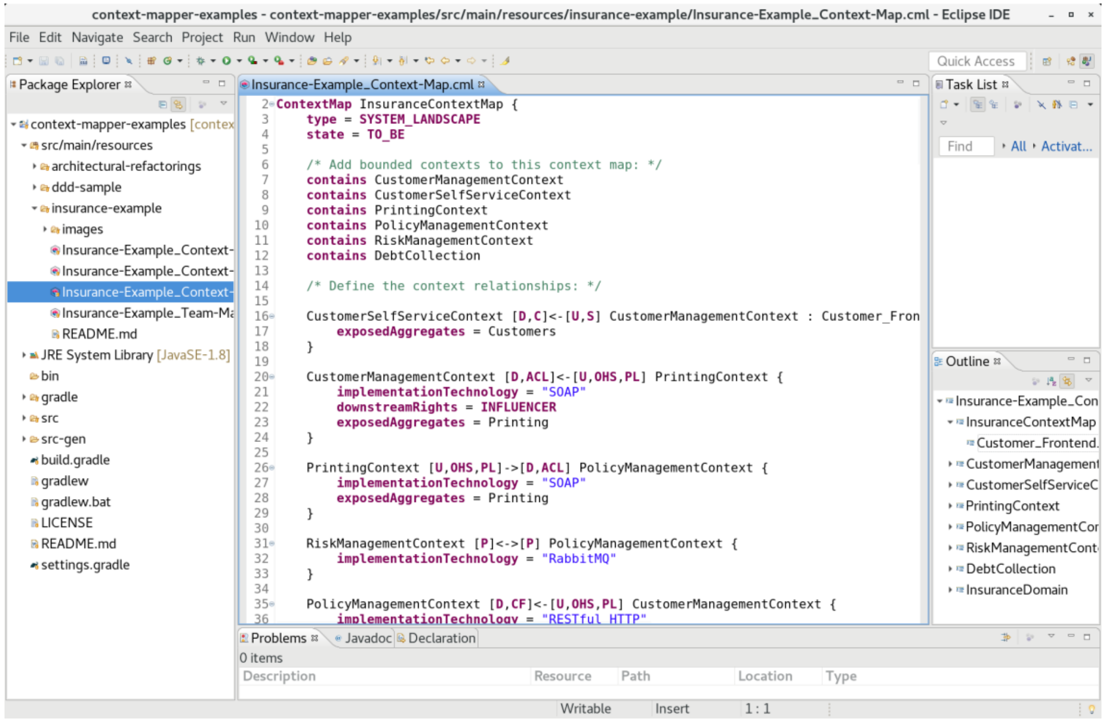
- **Goal: provide clear and concise interpretation of the strategic DDD patterns – and valid combinations of them**



Reference: https://contextmapper.org/docs/language-model/

Context Mapper: DSL implements Meta-Model and Semantics

- **A Domain-Specific Language (DSL) for DDD:**
  - Formal, machine-readable DDD Context Maps via *editors and validators*
  - Model/code *generators* to convert models into other representations
  - Model transformations for *refactorings* (e.g., “Split Bounded Context”)



```
ContextMap InsuranceContextMap {
    type = SYSTEM_LANDSCAPE
    state = TO_BE

    /* Add bounded contexts to this context map: */
    contains CustomerManagementContext
    contains CustomerSelfServiceContext
    contains PrintingContext
    contains PolicyManagementContext
    contains RiskManagementContext
    contains DebtCollection

    /* Define the context relationships: */

    CustomerSelfServiceContext [D,C]<-[U,S] CustomerManagementContext : Customer_Fron
        exposedAggregates = Customers
    }

    CustomerManagementContext [D,ACL]<-[U,OHS,PL] PrintingContext {
        implementationTechnology = "SOAP"
        downstreamRights = INFLUENCER
        exposedAggregates = Printing
    }

    PrintingContext [U,OHS,PL]->[D,ACL] PolicyManagementContext {
        implementationTechnology = "SOAP"
        exposedAggregates = Printing
    }

    RiskManagementContext [P]<->[P] PolicyManagementContext {
        implementationTechnology = "RabbitMQ"
    }

    PolicyManagementContext [D,CF]<-[U,OHS,PL] CustomerManagementContext {
        implementationTechnology = "RESTful HTTP"
```

Plugin update site: https://dl.bintray.com/contextmapper/context-mapping-dsl/updates/

Context Mapper: Domain-Specific Language

```
ContextMap DDDSampleMap {
    contains CargoBookingContext
    contains VoyagePlanningContext
    contains LocationContext

    CargoBookingContext [SK]<->[SK] VoyagePlanningContext

    [U,OHS,PL] LocationContext -> [D] CargoBookingContext

    VoyagePlanningContext [D]<-[U,OHS,PL] LocationContext
}
```

Bounded Contexts (systems or teams)

DDD relationship patterns (role of endpoint)

Influence/data flow direction: ->, <-> (upstream-downstream or symmetric)

SK: Shared Kernel, **PL:** Published Language
D: Downstream, **U:** Upstream
ACL: Anti-Corruption Layer, **OHS:** Open Host Service

Pros and Cons of Context Mapper DSL

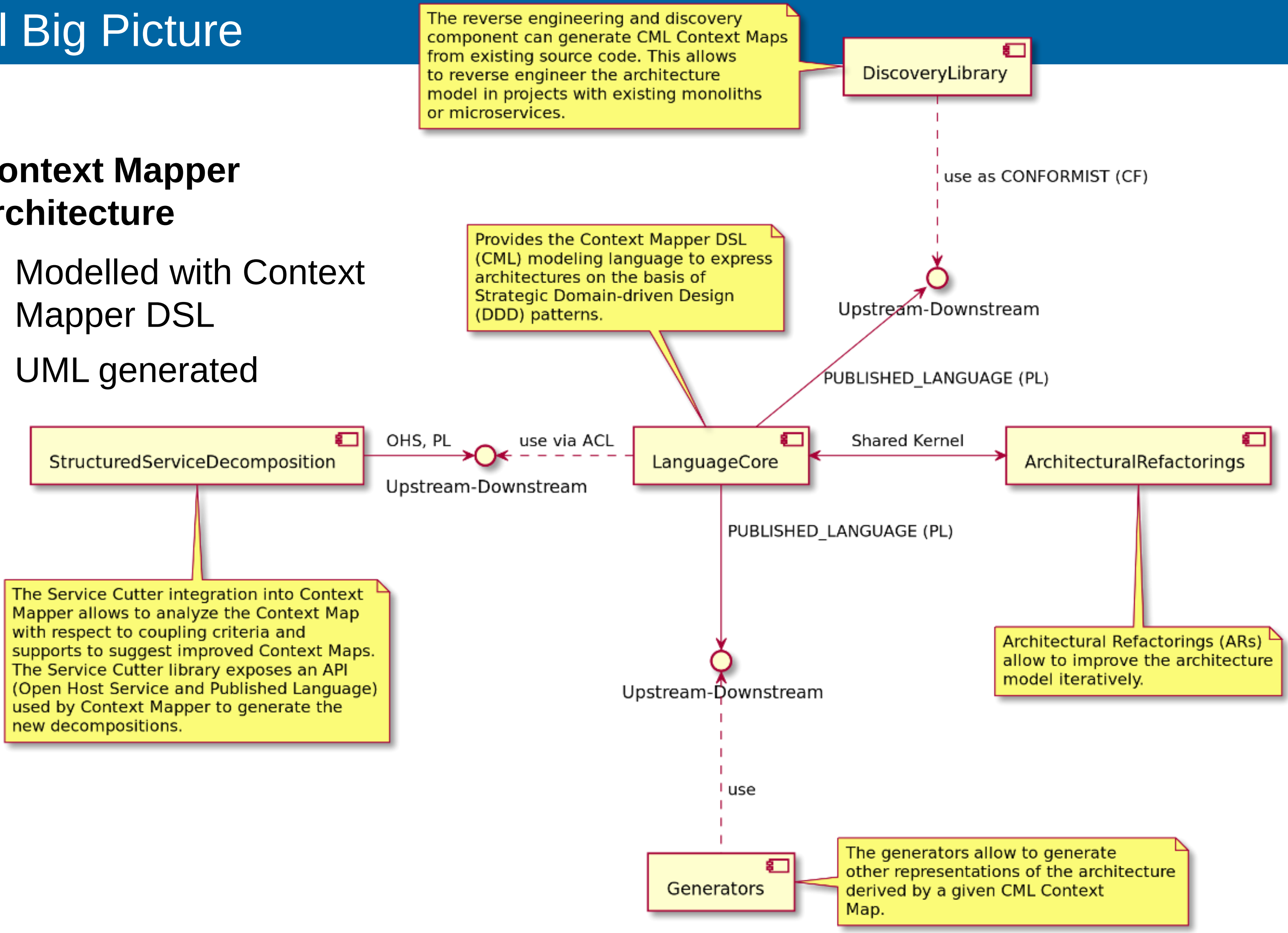
- **Pros:**
  - High *understandability* and *usability* for DDD adopters (conformance with patterns)
  - Increased *productivity* in context mapping
  - Iterative (agile) evolution
  - Diagrams on different levels of abstraction
    - Context, component and class diagrams
  - Future-proof: domain modeling is architecture and technology independent
  - Framework maturity increased iteratively
- **Cons:**
  - Steep *learning curve* for modelers not familiar with DDD
  - Model-driven approach potentially considered to be "not agile"
  - Maintenance of different levels of abstraction in one model (CML)
  - Supporting many IDEs will be expensive (currently limited to Eclipse)

Session Outline

- **Context and motivation**
  - User stories for Context Mapper
  - Application integration example
  - Domain-Driven Design (DDD) in a nutshell
- **Proposed Modeling Language and Tools**
  - Domain-Driven Design (DDD) meta model
  - Domain-Specific Language (DSL) core
  - Generation tools
- ***Future Work***
  - *Context Mapper as an Architecture Recoverer*
  - *Context Mapper as a (Micro-)Service Decomposer*
  - *Context Mapper as an Enterprise Portfolio Planning Tool*

Tool Big Picture

- **Context Mapper architecture**
  - Modelled with Context Mapper DSL
  - UML generated



The reverse engineering and discovery component can generate CML Context Maps from existing source code. This allows to reverse engineer the architecture model in projects with existing monoliths or microservices.

Provides the Context Mapper DSL (CML) modeling language to express architectures on the basis of Strategic Domain-driven Design (DDD) patterns.

The Service Cutter integration into Context Mapper allows to analyze the Context Map with respect to coupling criteria and supports to suggest improved Context Maps. The Service Cutter library exposes an API (Open Host Service and Published Language) used by Context Mapper to generate the new decompositions.

Architectural Refactorings (ARs) allow to improve the architecture model iteratively.

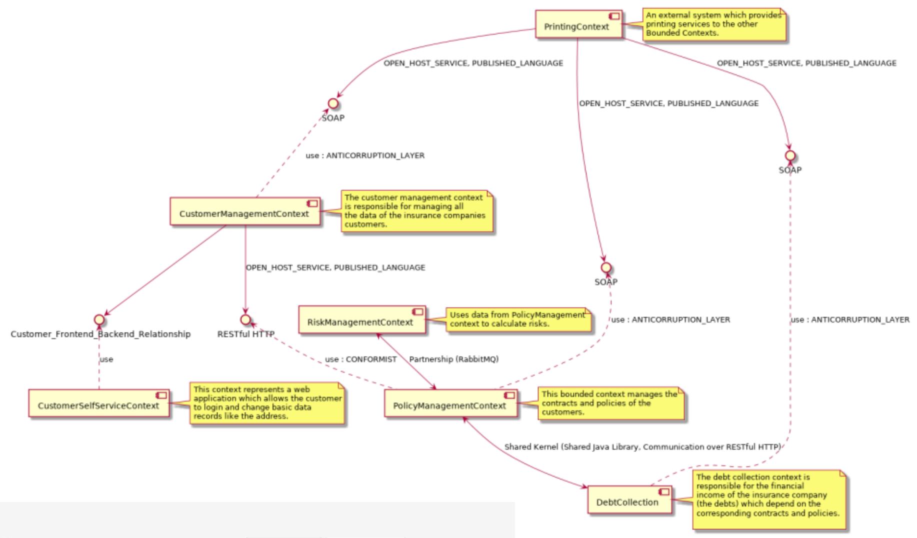
The generators allow to generate other representations of the architecture derived by a given CML Context Map.

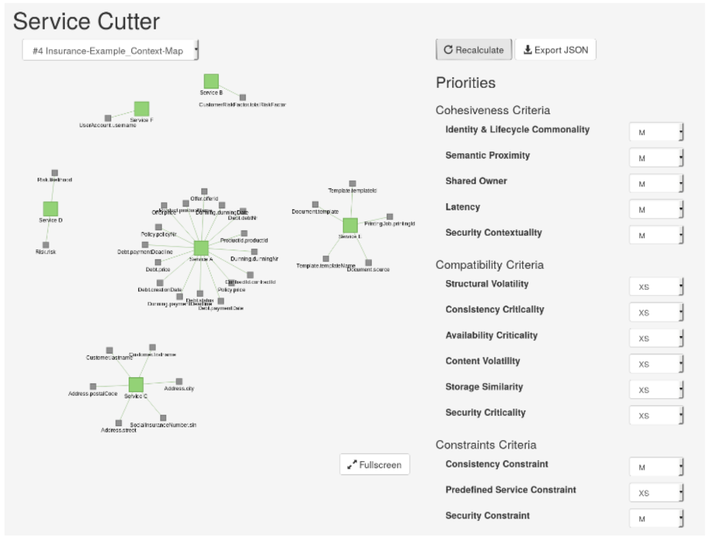
Discover Models From Existing Code

- **Strategy-based reverse engineering**
- **Discover Bounded Contexts and Context Maps**
  - Reverse engineer domain models within Bounded Contexts
  - Detect relationships between (micro-)services to derive Context Map
- **Potential approaches:**
  - Detect (micro-)services (Bounded Contexts) by the framework used for implementation, such as Spring Boot.
  - Derive relationships between Bounded Contexts by analyzing container deployment configurations, such as Docker Compose.

Context Mapper: Generators (DDD DSL as Input)

- **PlantUML generator**
  - Generate graphical representations of model
- **Service Cutter input generator**
  - Use structured approach to identify service candidates
  - Term project/bachelor thesis at HSR FHO
- **MDSL service contract generator**
  - Generate technology-agnostic (micro-)service contracts from Bounded Contexts/Aggregates


PrintingContext
An external system which provides printing services to the other Bounded Contexts.
OPEN_HOST_SERVICE, PUBLISHED_LANGUAGE
OPEN_HOST_SERVICE, PUBLISHED_LANGUAGE
OPEN_HOST_SERVICE, PUBLISHED_LANGUAGE
SOAP
SOAP
use : ANTICORRUPTION_LAYER
CustomerManagementContext
The customer management context is responsible for managing all the data of the insurance companies customers.
OPEN_HOST_SERVICE, PUBLISHED_LANGUAGE
SOAP
Customer_Frontend_Backend_Relationship
RESTful HTTP
RiskManagementContext
Uses data from PolicyManagement context to calculate risks.
use : ANTICORRUPTION_LAYER
use : ANTICORRUPTION_LAYER
use
use : CONFORMIST
Partnership (RabbitMQ)
CustomerSelfServiceContext
This context represents a web application which allows the customer to login and change basic data records like the address.
PolicyManagementContext
This bounded context manages the contracts and policies of the customers.
Shared Kernel (Shared Java Library, Communication over RESTful HTTP)
DebtCollection
The debt collection context is responsible for the financial income of the insurance company (the debts) which depend on the corresponding contracts and policies.



Priorities

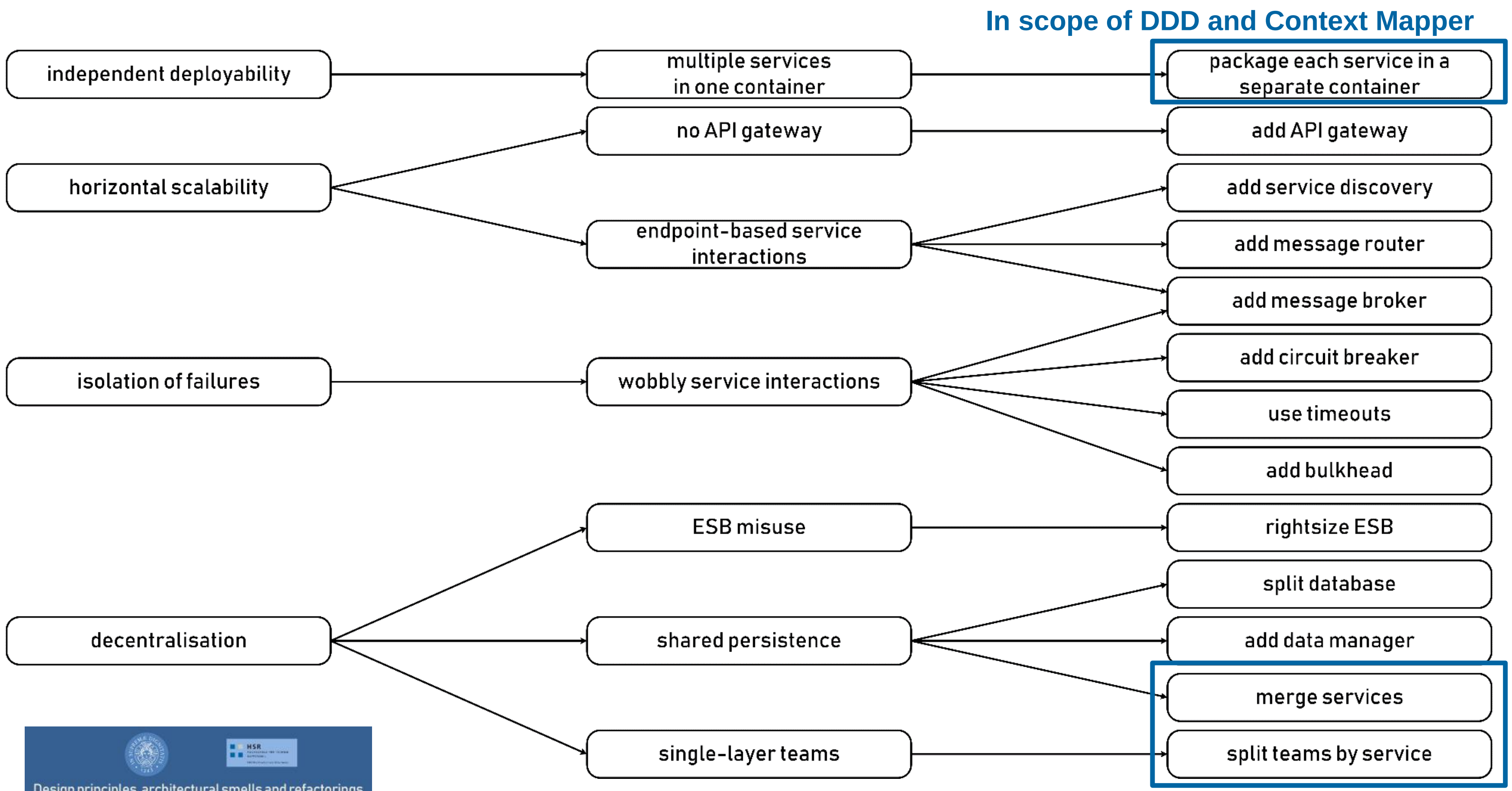
Cohesiveness Criteria	
Identity & Lifecycle Commonality	M
Semantic Proximity	M
Shared Owner	M
Latency	M
Security Contextuality	M

Compatibility Criteria	
Structural Volatility	XS
Consistency Criticality	XS
Availability Criticality	XS
Content Volatility	XS
Storage Similarity	XS
Security Criticality	XS

Constraints Criteria	
Consistency Constraint	M
Predefined Service Constraint	XS
Security Constraint	M

http://servicecutter.github.io/

From Biz and Dev to Ops: Bad Smells and Refactorings



Design principles, architectural smells and refactorings for microservices: A multivocal review

Antonio Brogi, Davide Neri, Jacopo Soldani
University of Pisa, Italy

Olaf Zimmerman
HSR FHO, Switzerland

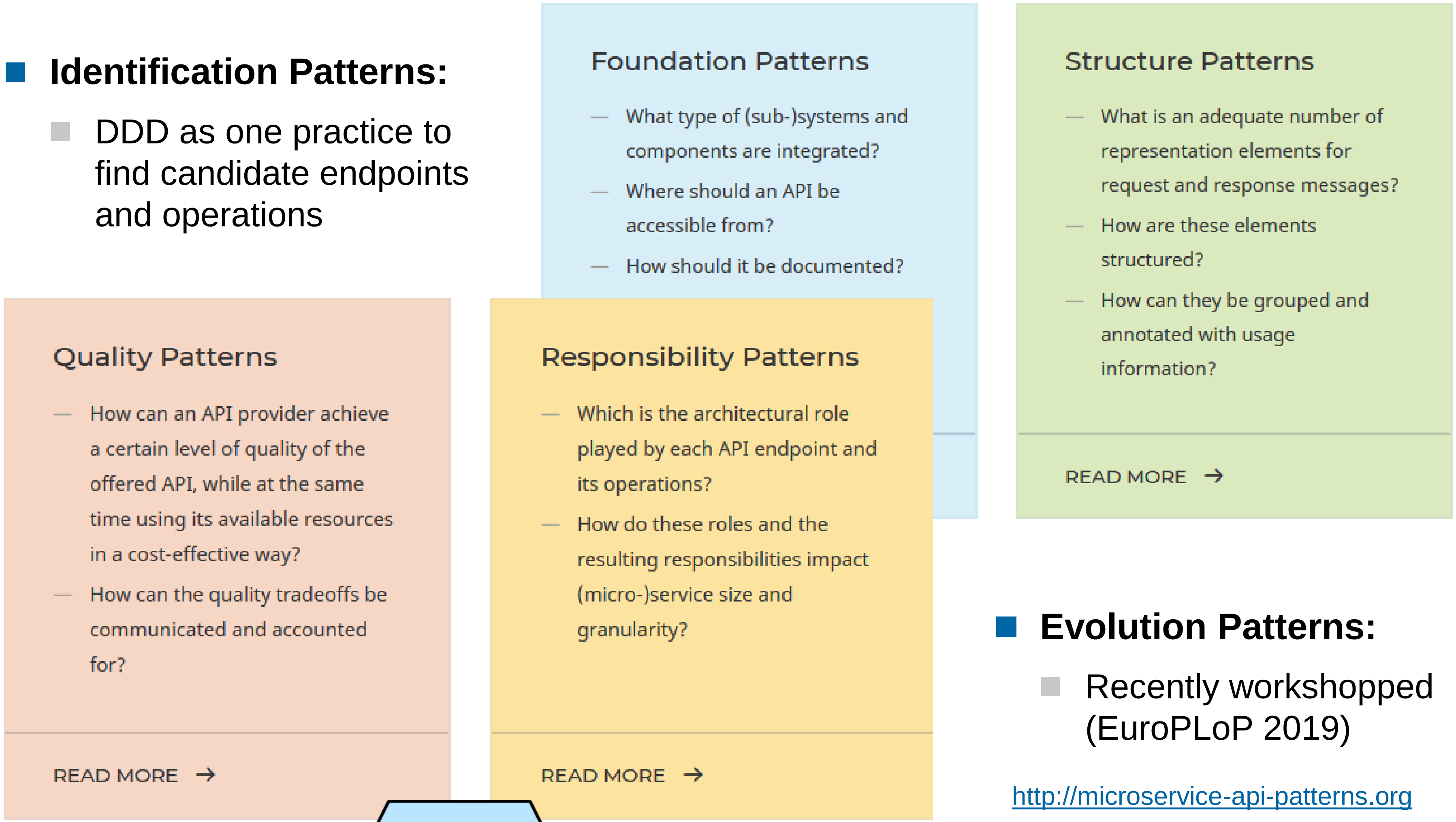
SummerSoc
20 june 2019, Crete, Greece

davide.neri@di.unipi.it

Reference: Brogi, A., Neri D., Soldani, J., Zimmermann, O., *Design Principles, Architectural Smells and Refactorings for Microservices*: A Multivocal Review. CoRR abs/1906.01553 and Springer SICS (2019) (online, report PDF, short presentation)

Microservice API Patterns (MAP) Categories

- **Identification Patterns:**
  - DDD as one practice to find candidate endpoints and operations



Foundation Patterns

- What type of (sub-)systems and components are integrated?
- Where should an API be accessible from?
- How should it be documented?

Structure Patterns

- What is an adequate number of representation elements for request and response messages?
- How are these elements structured?
- How can they be grouped and annotated with usage information?

READ MORE →

Quality Patterns

- How can an API provider achieve a certain level of quality of the offered API, while at the same time using its available resources in a cost-effective way?
- How can the quality tradeoffs be communicated and accounted for?

READ MORE →

Responsibility Patterns

- Which is the architectural role played by each API endpoint and its operations?
- How do these roles and the resulting responsibilities impact (micro-)service size and granularity?

READ MORE →

- **Evolution Patterns:**
  - Recently workshopped (EuroPLoP 2019)

http://microservice-api-patterns.org

Microservice API Patterns (MAP)

Other Directions and Ideas (for Consideration)

- **Increase target audience**
  - Support more IDEs and Web editing (from Xtext to Theia?)
- **Context Mapper as …**
  - An agile planning tool (or input provider for such tool)?
  - An architectural decision identifier and facilitator ("knowledge navigator")?
  - An enterprise architecture or portfolio manager (TOGAF, Safe, …)?
- **DSLs and supporting tools for …**
  - Rapid application prototyping (e.g., generate JHipster configurations)?
  - Low code/no code development
  - Cross-protocol service design (messaging, for HTTP)?

Looking forward to your comments and ideas –
and opportunities to collaborate (?)

Summary and Outlook

- **DDD is a trending approach for concept modeling and service decomposition**
  - Applied by many practitioners
  - DDD Context Maps are created manually so far
- **Context Mapper supports practitioners in applying DDD**
  - DSL for modeling strategic DDD Context Maps
  - Tool support to evolve models iteratively (ARs)
  - PlantUML, Service Cutter, and MDSL generation
- **Part of a modular and extensible modeling framework for strategic DDD (and more)**
  - Reverse engineer models
  - Generate code (micro-) service project stubs
  - Systematic service decomposition

Thank you very much! Let's move on to Q&A and discussion…

More Information

- **Master thesis and previous term project reports:**
  - http://eprints.hsr.ch/id/eprint/821
  - https://eprints.hsr.ch/784/ and https://eprints.hsr.ch/722/
- **Context Mapper on the Web:**
  - https://contextmapper.org/ and https://contextmapper.org/docs/home/
- **Eclipse update site:**
  - https://dl.bintray.com/contextmapper/context-mapping-dsl/updates/
- **GitHub repositories:**
  - DSL: https://github.com/ContextMapper/context-mapper-dsl
  - Examples: https://github.com/ContextMapper/context-mapper-examples