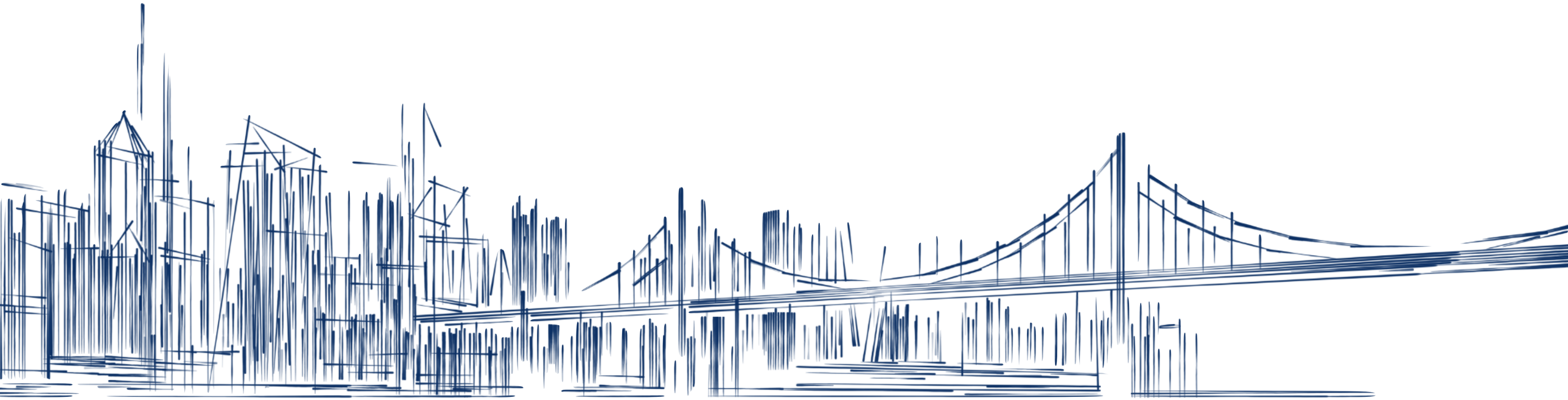


BOTTOM-UP ARCHITECTURE

BRIDGING THE ARCHITECTURE CODE GAP

Oliver Drotbohm odrotbohm oliver.drotbohm@broadcom.com

github.com/odrotbohm

odrotbohm

Type / to search

Overview | Repositories 123 | Projects 1 | Packages | Stars 77

Oliver Drotbohm

odrotbohm · he/him

Frameworks & Architecture Engineering @ VMware, OpenSource enthusiast, all things Spring, Java, data, DDD, REST, software architecture, drums & music.

Edit profile

3.4k followers · **32** following

- VMware
- Dresden, Germany
- 17:49 (UTC +01:00)
- info@odrotbohm.de
- www.odrotbohm.de
- @odrotbohm
- @odrotbohm@chaos.social
- odrotbohm
- in/odrotbohm

Pinned

Customize your pins

xmolecules/jmolecules Public
Libraries to help developers express architectural abstractions in Java code
Java ☆ 979 ⑂ 85

xmolecules/jmolecules-integrations Public
Technology integration for jMolecules
Java ☆ 48 ⑂ 11

lectures Public
Lecture scripts and slides I use during the Software Engineering course at TU Dresden
Java ☆ 68 ⑂ 22

spring-restbucks Public
Implementation of the sample from REST in Practice based on Spring projects
Java ☆ 1.2k ⑂ 404

spring-playground Public
A collection of tiny helpers for building Spring applications
Java ☆ 96 ⑂ 10

spring-projects/spring-modulith Public
Modular applications with Spring Boot
Java ☆ 531 ⑂ 67

Single sign-on to see contributions within the **pivotal** organization.

1,559 contributions in the last year

Contribution settings

Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov

Mon
Wed
Fri

Learn how we count contributions

Less More

2023
2022
2021
2020
2019
2018
2017
2016
2015

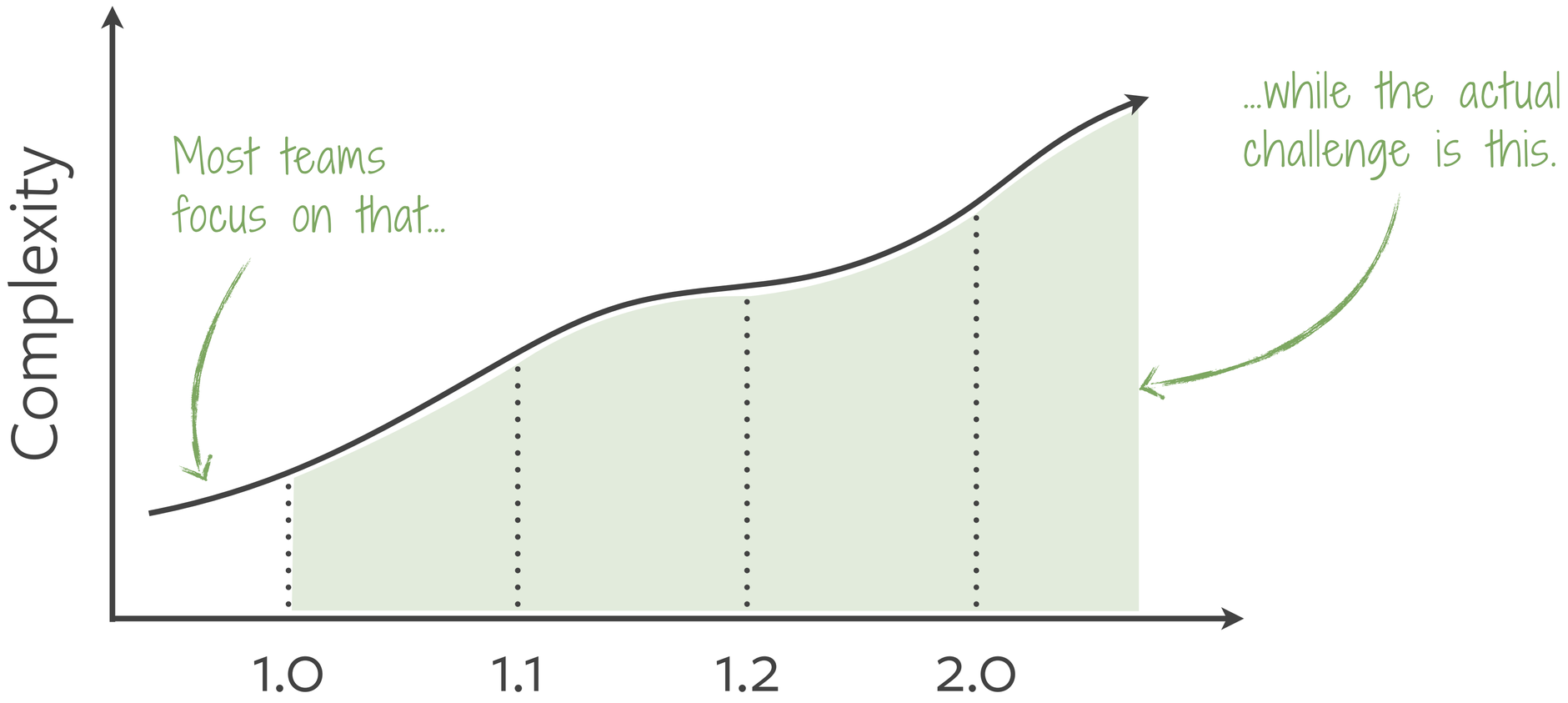
@spring-projects | @st-tu-dresden | @xmolecules | More

Activity overview

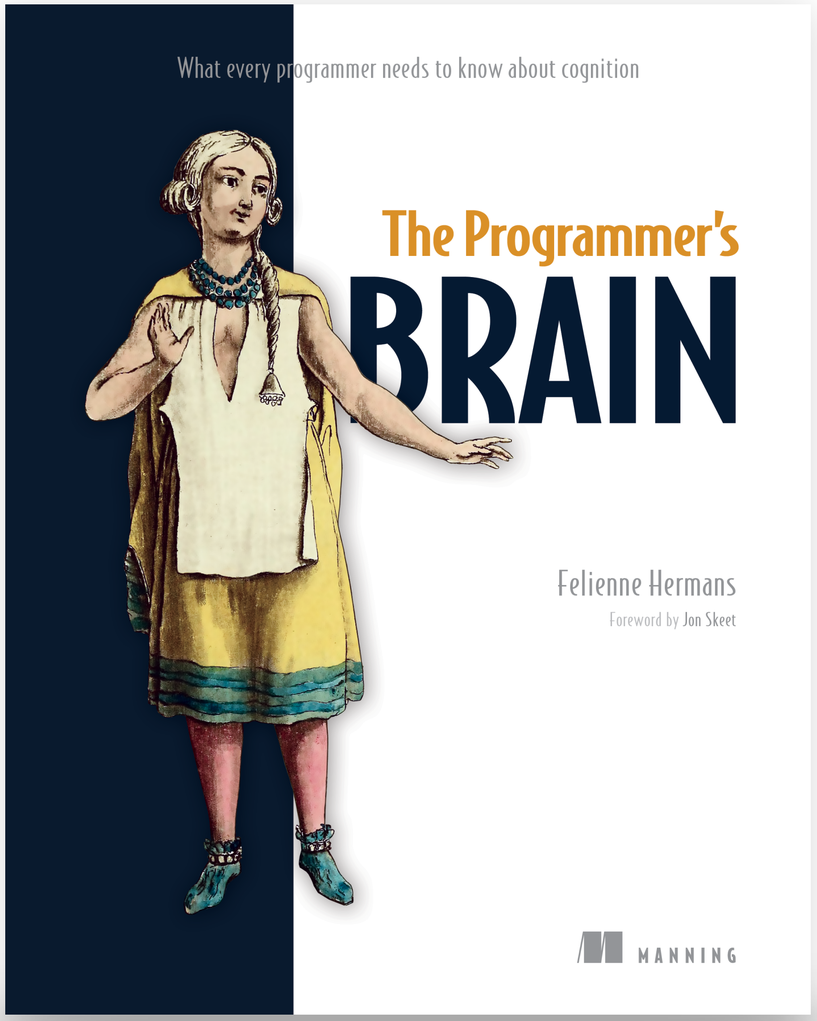
Contributed to

1%
Code review

We want to build evolvable systems.


Complexity
Most teams
focus on that...
...while the actual
challenge is this.
1.0
1.1
1.2
2.0

Understandability

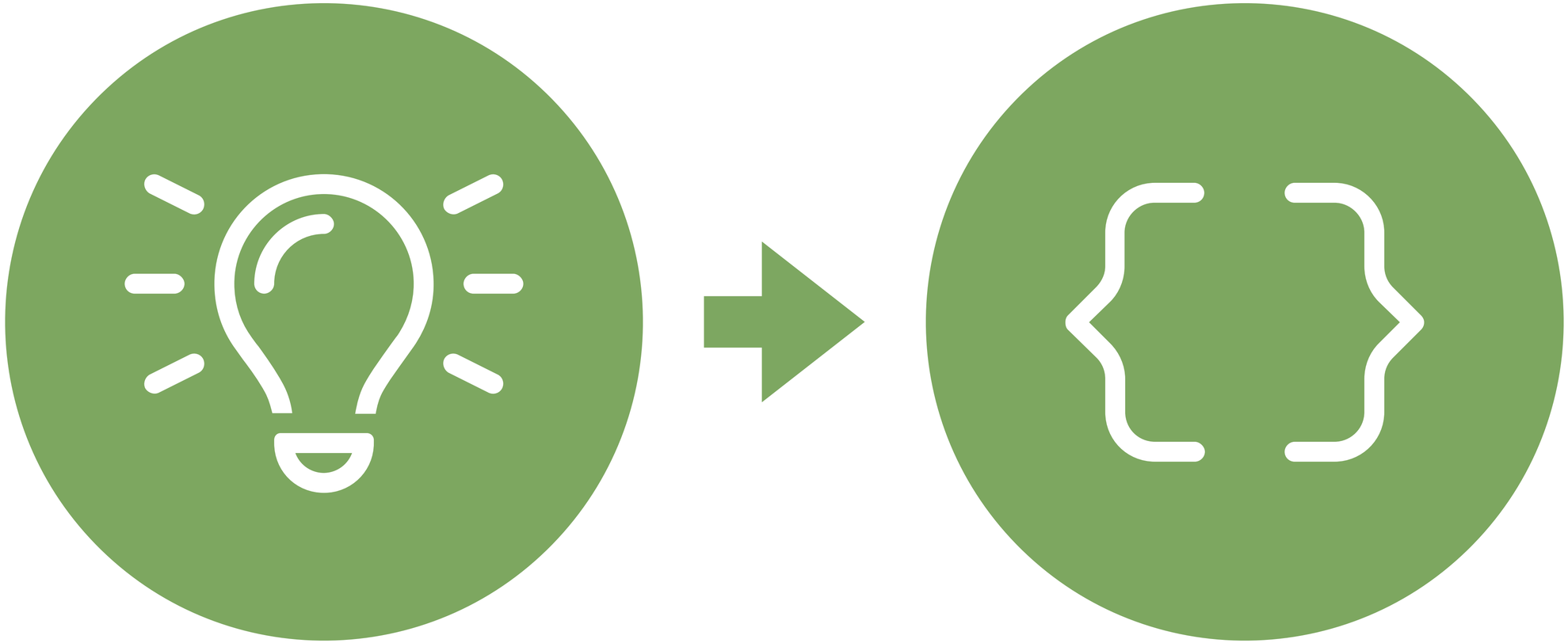

What every programmer needs to know about cognition
The Programmer's
BRAIN
Felienne Hermans
Foreword by Jon Skeet
MANNING

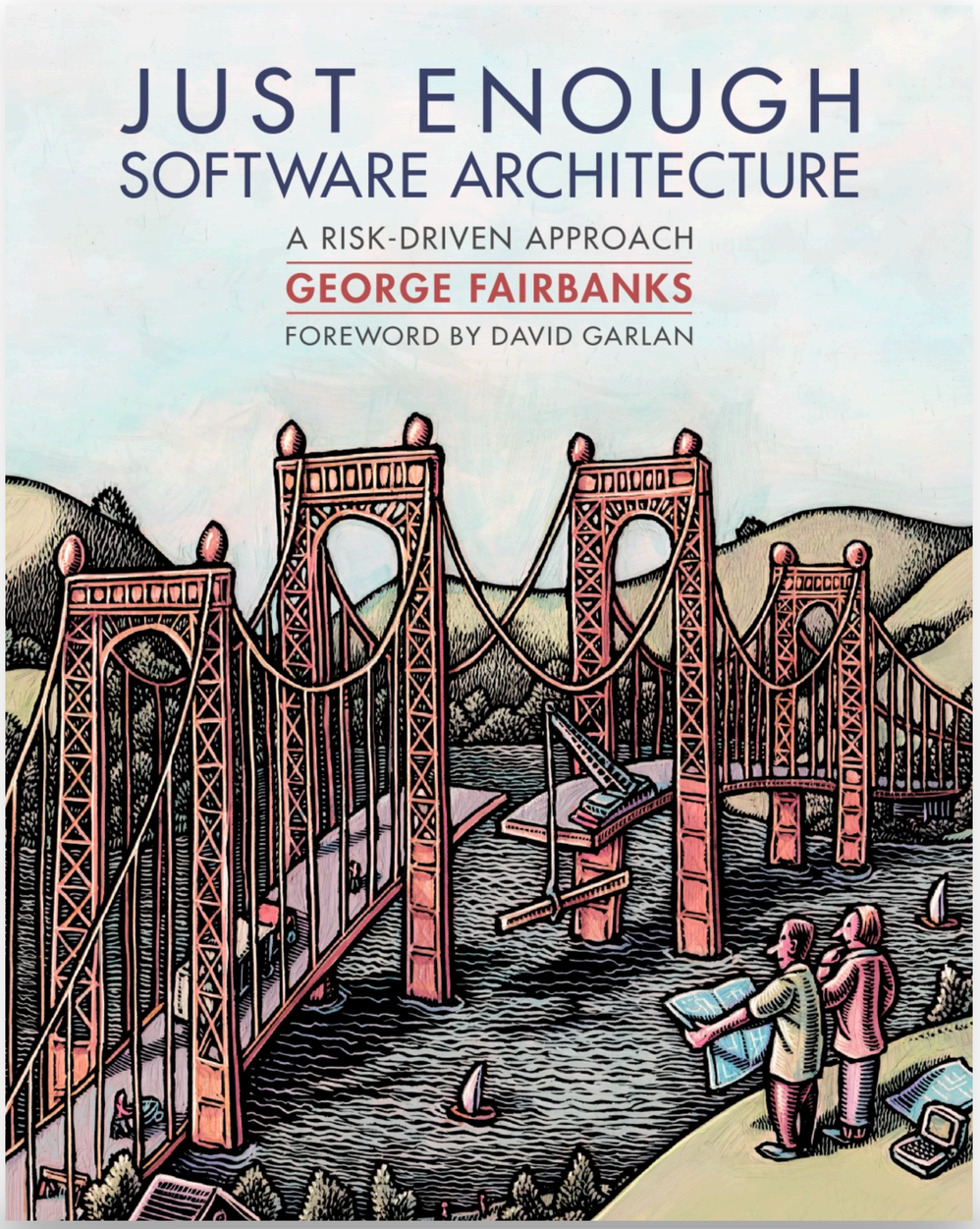

WPS WORKPLACE SOLUTIONS
dpunkt.verlag
Carola Lilienthal
Sustainable Software Architecture
Analyze and Reduce Technical Debt

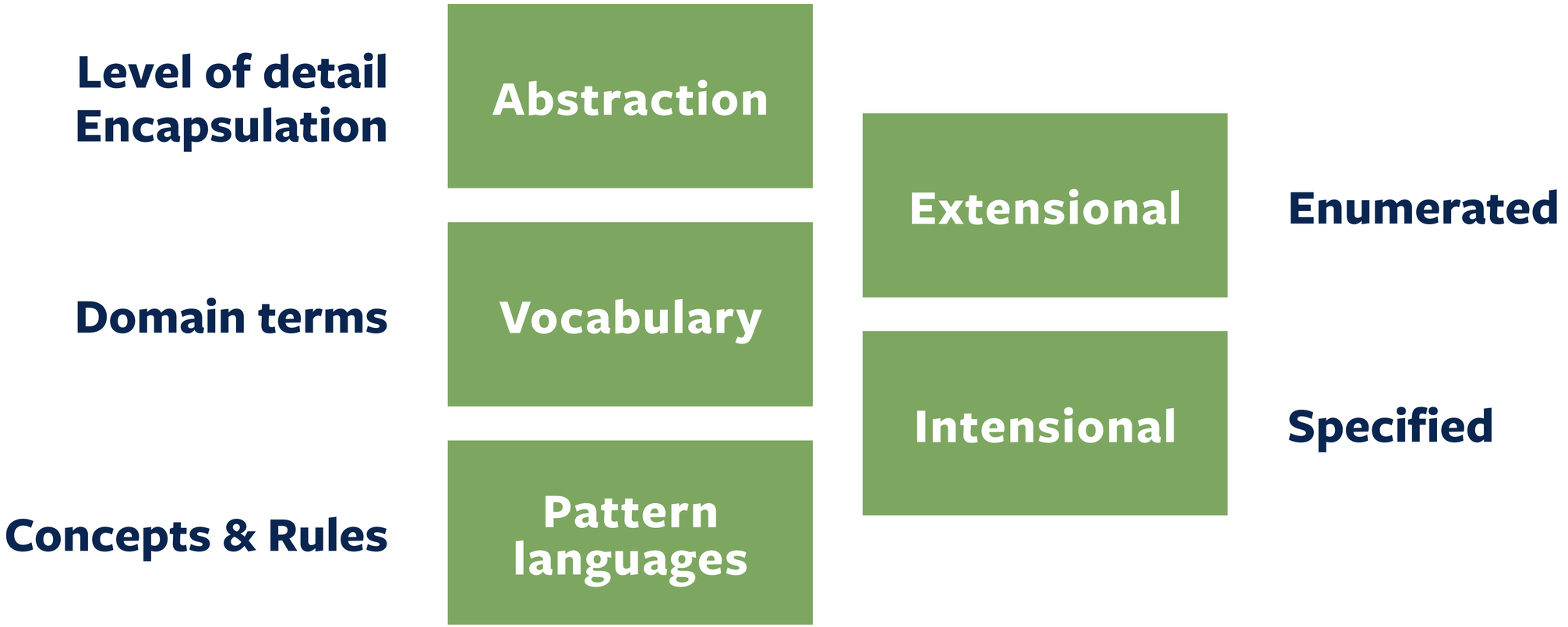
Chunking

Hierarchization

Pattern languages




JUST ENOUGH
SOFTWARE ARCHITECTURE
A RISK-DRIVEN APPROACH
GEORGE FAIRBANKS
FOREWORD BY DAVID GARLAN


Level of detail
Encapsulation
Abstraction
Domain terms
Vocabulary
Concepts & Rules
Pattern languages
Extensional
Intensional
Enumerated
Specified

Architecturally-Evident Code?

Extensional

Components / Modules

Invoicing,
Shipment

Deployables / Build modules / Packages

Domain language

EmailAddress,
ZipCode

Classes, methods, fields

Intensional

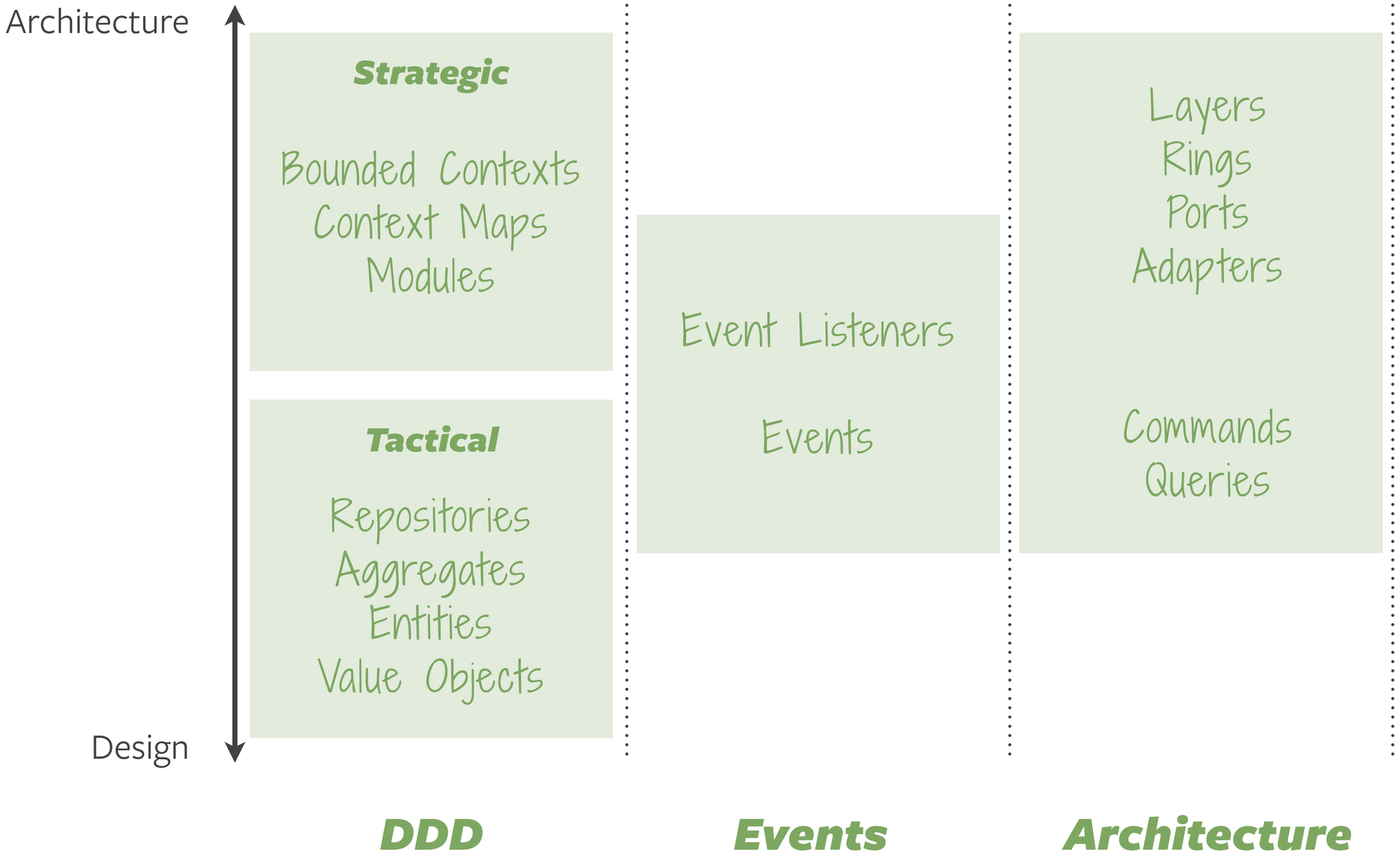
Concepts & Rules

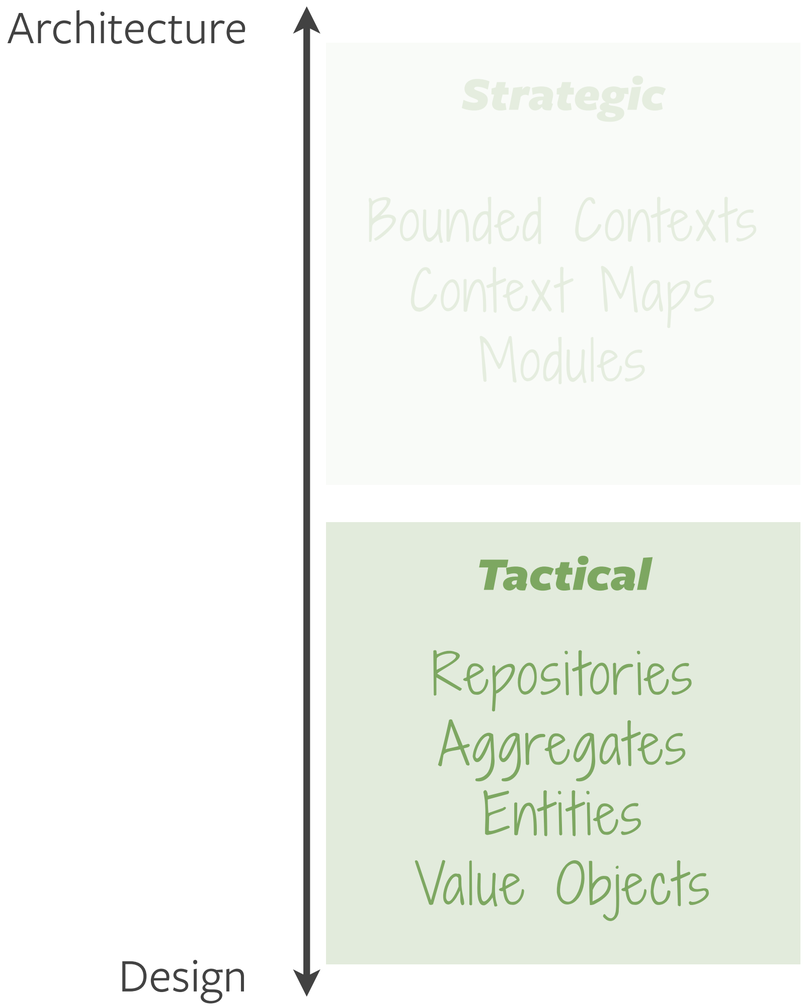
ValueObject,
Entity,
Aggregate

Layers,
Rings

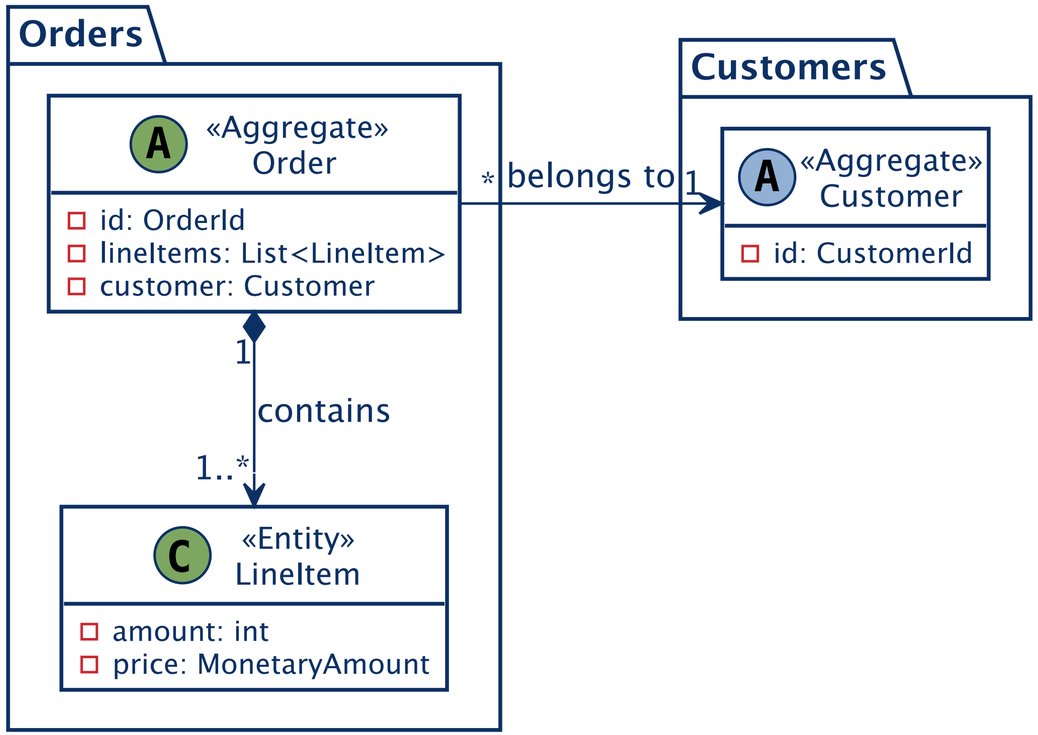
Naming conventions

What else? 🤔

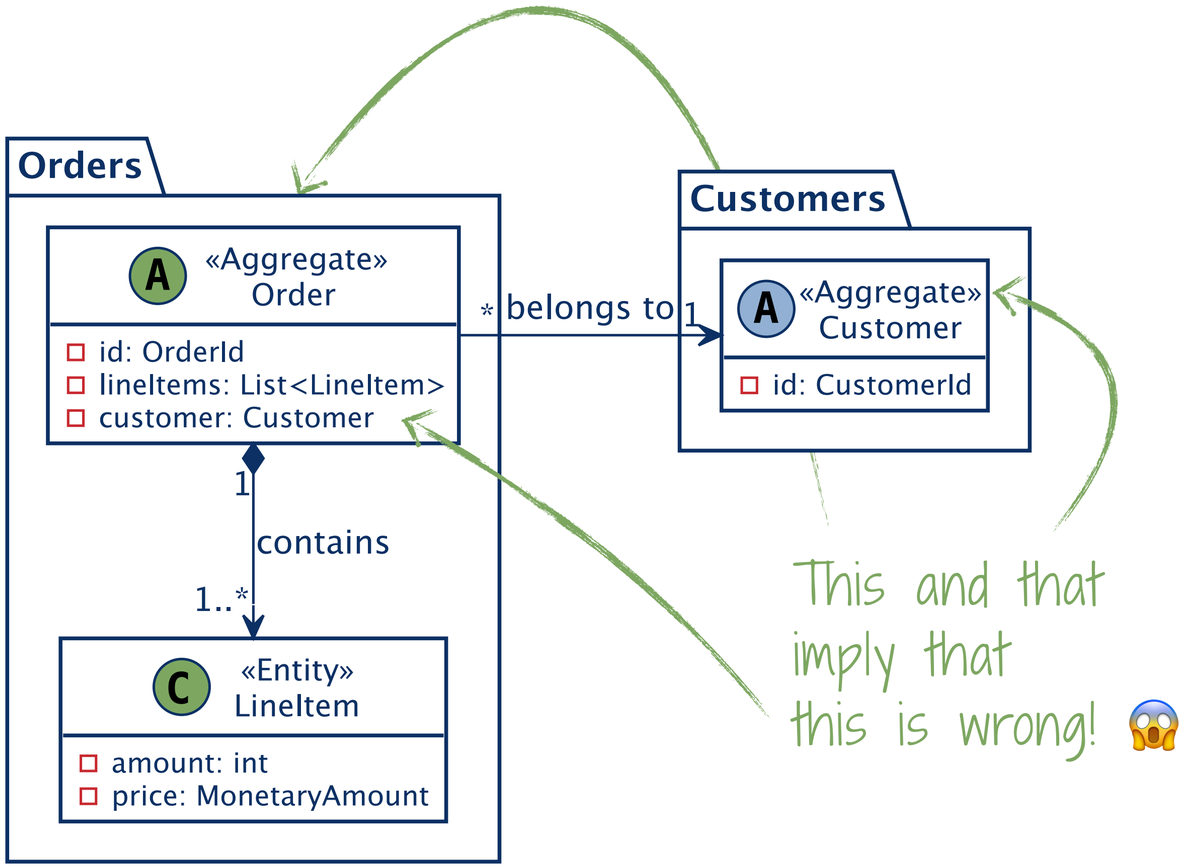

Architecture
Strategic
Bounded Contexts
Context Maps
Modules
Tactical
Repositories
Aggregates
Entities
Value Objects
Design
Event Listeners
Events
Layers
Rings
Ports
Adapters
Commands
Queries
DDD
Events
Architecture

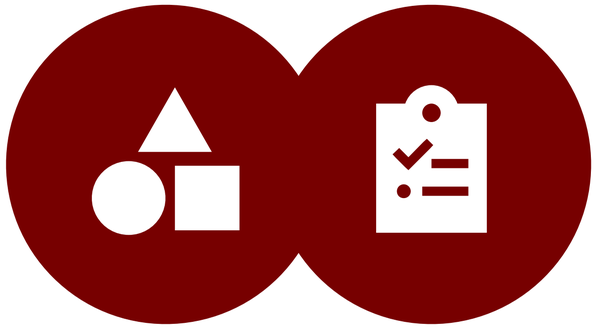

Architecture
Strategic
Bounded Contexts
Context Maps
Modules
Tactical
Repositories
Aggregates
Entities
Value Objects
Design

DDD



A simple Aggregate arrangement





A simple Aggregate arrangement

Your Software. Your Structures. Your Rules.

Establishing an Aggregate... in jQAssistant

```
MATCH
  (repo:Java:Type)
    -[:IMPLEMENTS_GENERIC]→ (superType)
    -[:OF_RAW_TYPE]→ (:Java:Type { fqn: "o.s.d.r.Repository"}),
  (superType)
    -[:HAS_ACTUAL_TYPE_ARGUMENT { index: 0 }]→ ()
    -[:OF_RAW_TYPE]→ (aggregateType)
SET
  aggregateType:Aggregate
RETURN
  repo, aggregateType
```

Reference to tech stack 😕

Establishes the concept

```
MATCH
  (aggregate:Aggregate)
    -[:DECLARES]→ (f:Field)
    -[:OF_TYPE]→ (fieldType:Aggregate)
WHERE
  aggregate <> fieldType
RETURN
  aggregate, fieldType
```

Establishes the rule

Establishing an Aggregate... in ArchUnit

```java
@AnalyzeClasses(packagesOf = Application.class)
public class ArchitectureTest {

  @ArchTest
  void verifyAggregates(JavaClasses types) {

    var aggregates = new AggregatesExtractor();
    var aggregateTypes = aggregates.doTransform(types);

    all(aggregates)
      .should(notReferToOtherAggregates(aggregateTypes))
      .check(types);
  }
}
```

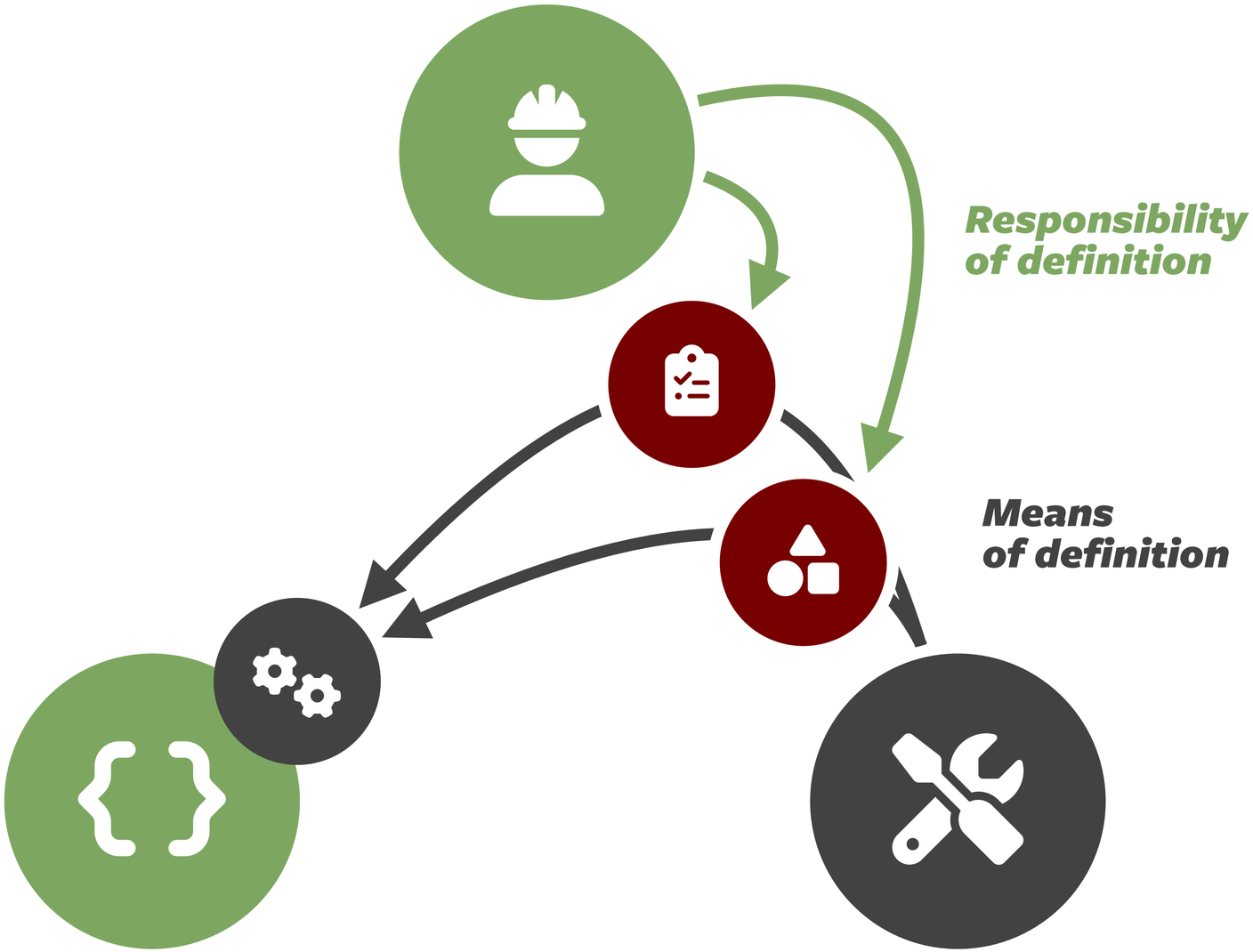
Establishes the concept

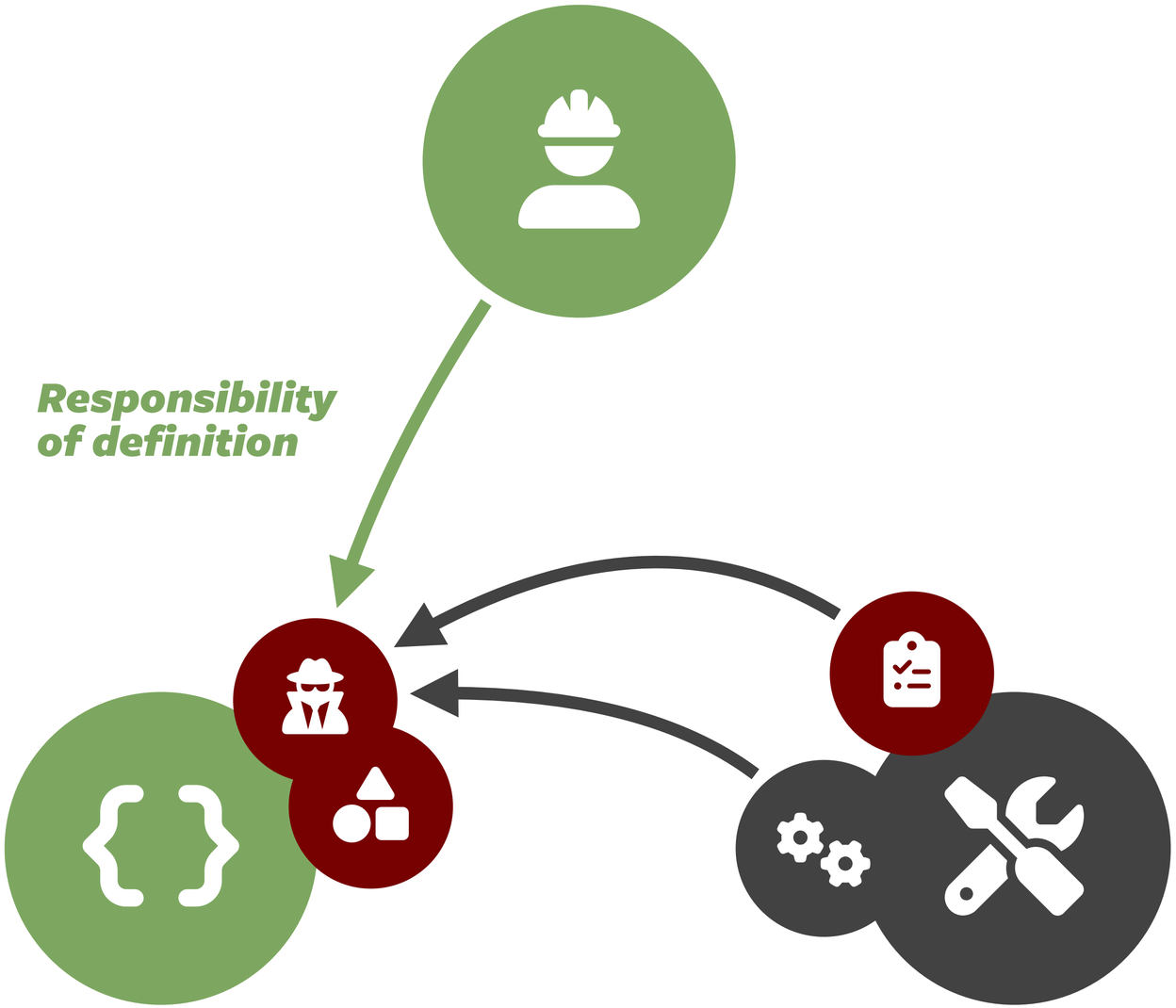
Establishes the rule

Code

Architecture

Technology


Responsibility
of definition
Means
of definition

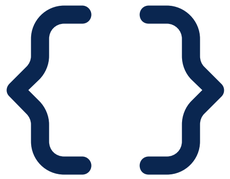

Responsibility
of definition

xMolecules

php

.NET

jMolecules



Explicit concepts

```java
@Entity
@NoArgsConstructor(force = true)
@EqualsAndHashCode(of = "id")
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order {

  private final @EmbeddedId OrderId id;

  @OneToMany(cascade = CascadeType.ALL)
  private List<LineItem> lineItems;
  private CustomerId customerId;

  public Order(CustomerId customerId) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customerId = customerId;
  }

  @Value
  @RequiredArgsConstructor(staticName = "of")
  @NoArgsConstructor(force = true)
  public static class OrderId implements Serializable {
    private static final long serialVersionUID = …;
    private final UUID orderId;
  }
}
```

```java
@Entity
@NoArgsConstructor(force = true)
@EqualsAndHashCode(of = "id")
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order implements o.j.d.t.AggregateRoot<Order, OrderId> {

  private final @EmbeddedId OrderId id;

  @OneToMany(cascade = CascadeType.ALL)
  private List<LineItem> lineItems;
  private CustomerId customerId;

  public Order(CustomerId customerId) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customerId = customerId;
  }

  @Value
  @RequiredArgsConstructor(staticName = "of")
  @NoArgsConstructor(force = true)
  public static class OrderId implements o.j.d.t.Identifier {
    private static final long serialVersionUID = …;
    private final UUID orderId;
  }
}
```

Verification

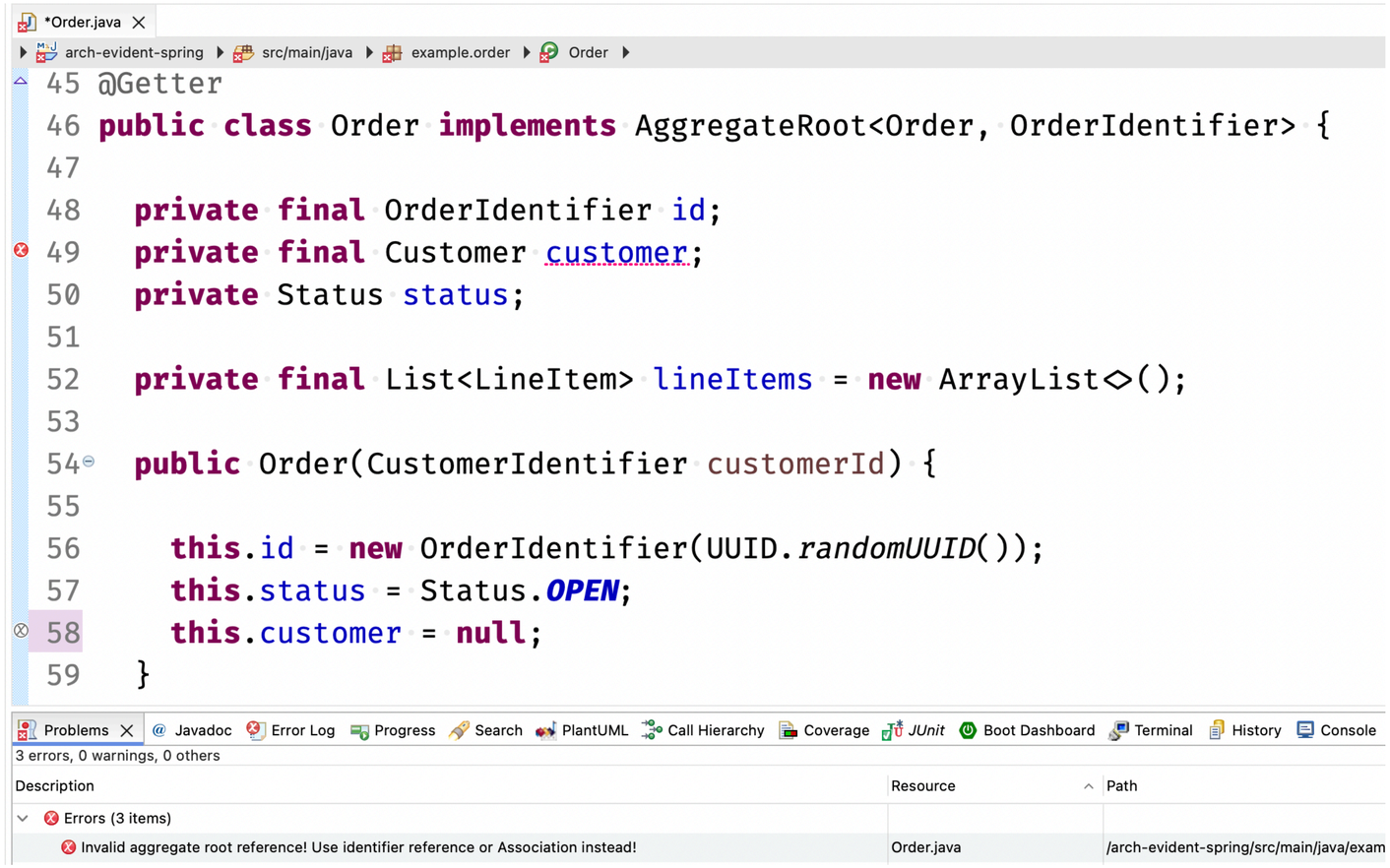
Verifying a jMolecules Aggregate … in jQAssistant

```
<plugin>
  <groupId>com.buschmais.jqassistant</groupId>
  <artifactId>jqassistant-maven-plugin</artifactId>
  <version>…</version>
  <executions>
    <execution>
      <id>default-cli</id>
      <goals>
        <goal>scan</goal>
        <goal>analyze</goal>
      </goals>
      <configuration>…</configuration>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>org.jqassistant.contrib.plugin</groupId>
      <artifactId>jqassistant-jmolecules-plugin</artifactId>
      <version>…</version>
    </dependency>
  </dependencies>
</plugin>
```

Verifying a jMolecules Aggregate … in ArchUnit

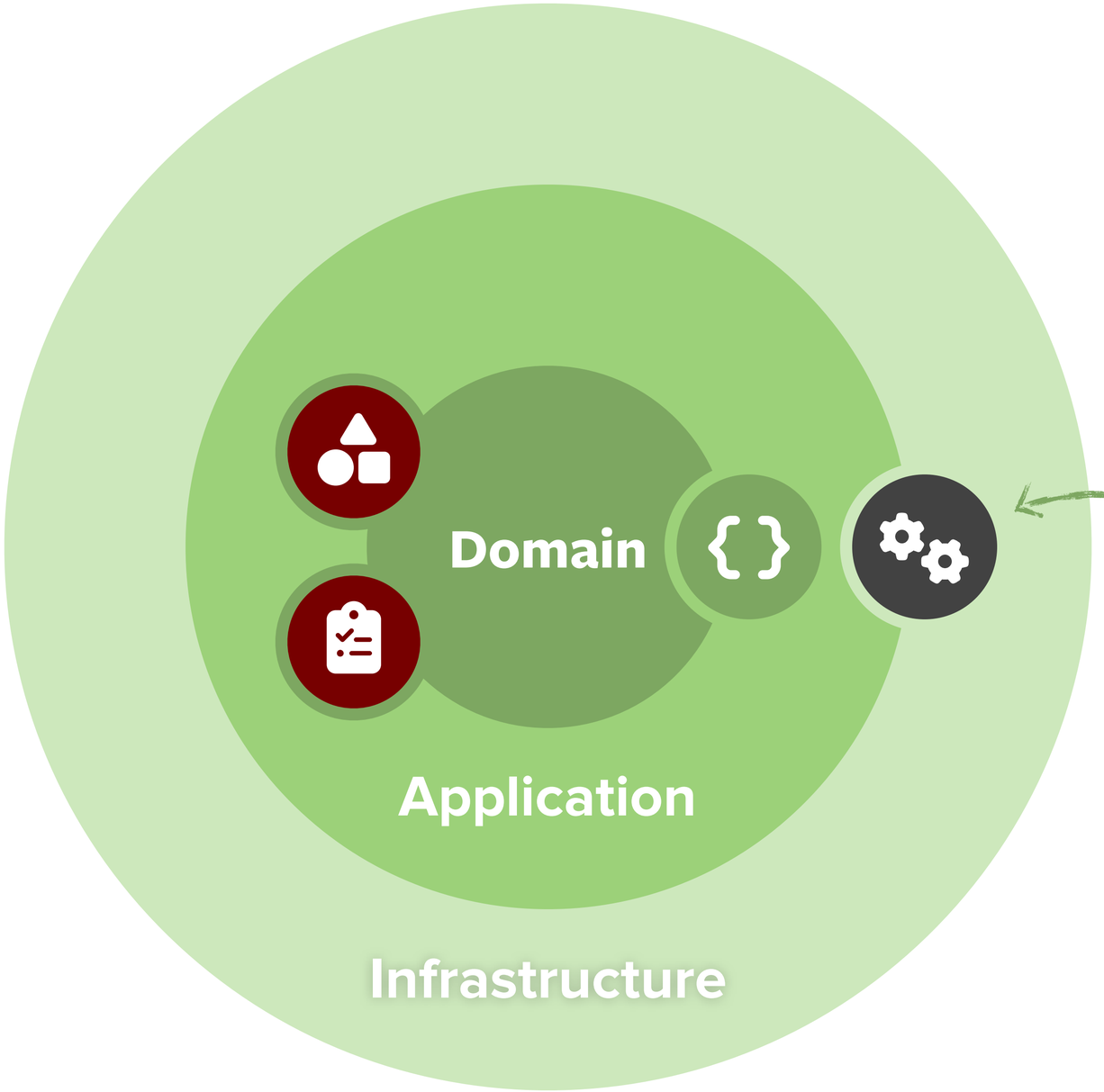
```
@AnalyzeClasses(packagesOf = Application.class)
class ArchitectureTests {

  @ArchTest
  ArchRule ddd = JMoleculesDddRules.all();
}
```

Simply execute the predefined rules



☝️ **Invalid aggregate root reference!**
Use identifier or Association instead!

Eliminate boilerplate



- Spring Framework
- JPA
- Jackson


```java
@Entity
@NoArgsConstructor(force = true)
@EqualsAndHashCode(of = "id")
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order {

  private final @EmbeddedId OrderId id;

  @OneToMany(cascade = CascadeType.ALL)
  private List<LineItem> lineItems;
  private CustomerId customerId;

  public Order(CustomerId customerId) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customerId = customerId;
  }

  @Value
  @RequiredArgsConstructor(staticName = "of")
  @NoArgsConstructor(force = true)
  public static class OrderId implements Serializable {
    private static final long serialVersionUID = …;
    private final UUID orderId;
  }
}
```

```java
@Entity
@NoArgsConstructor(force = true)
@EqualsAndHashCode(of = "id")
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order implements AggregateRoot<Order, OrderId> {

  private final @EmbeddedId OrderId id;

  @OneToMany(cascade = CascadeType.ALL)
  private List<LineItem> lineItems;
  private Association<Customer, CustomerId> customer;

  public Order(CustomerId customerId) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customer = Association.forId(customerId);
  }

  @Value
  @RequiredArgsConstructor(staticName = "of")
  @NoArgsConstructor(force = true)
  public static class OrderId implements Identifier {
    private static final long serialVersionUID = …;
    private final UUID orderId;
  }
}
```

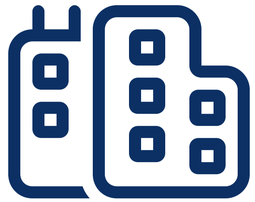
```java
@Entity
@NoArgsConstructor(force = true)
@EqualsAndHashCode(of = "id")
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order implements AggregateRoot<Order, OrderId> {

  private final @EmbeddedId OrderId id;

  @OneToMany(cascade = CascadeType.ALL)
  private List<LineItem> lineItems;
  private Association<Customer, CustomerId> customer;

  public Order(CustomerId customerId) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customer = Association.forId(customerId);
  }

  @Value
  @RequiredArgsConstructor(staticName = "of")
  @NoArgsConstructor(force = true)
  public static class OrderId implements Identifier {
    private static final long serialVersionUID = …;
    private final UUID orderId;
  }
}
```

Meanwhile in your IDE…

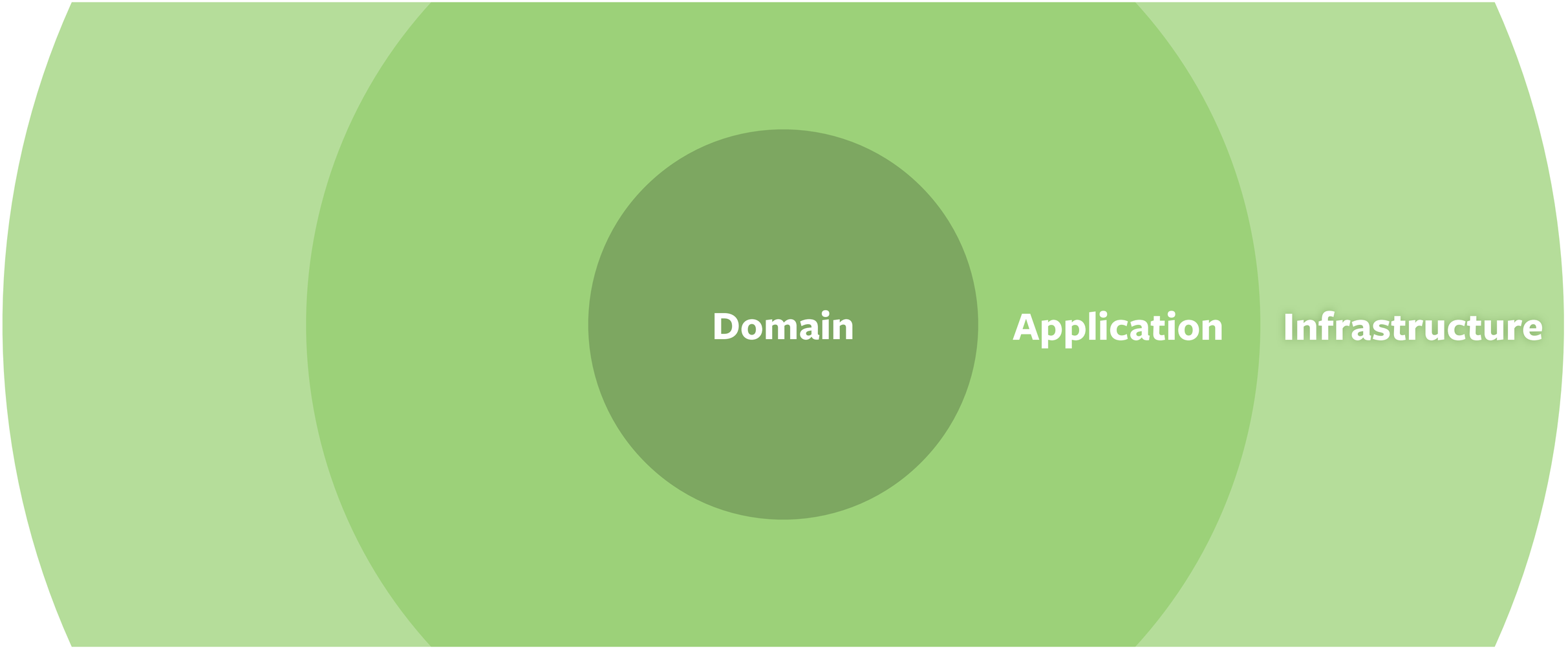
```
[INFO] ◻— example.order.Order
[INFO] ├— JPA - Adding @j.p.Entity.
[INFO] ├— JPA - Adding default constructor.
[INFO] ├— JPA - Adding nullability verification using new callback methods.
[INFO] ├— JPA - Defaulting id mapping to @j.p.EmbeddedId().
[INFO] ├— JPA - Defaulting lineItems mapping to @j.p.JoinColumn(…).
[INFO] ├— JPA - Defaulting lineItems mapping to @j.p.OneToMany(…).
[INFO] ├— Spring Data JPA - Implementing o.s.d.d.Persistable<e.o.Order$OrderIdentifier>.
[INFO] └— Spring JPA - customer - Adding @j.p.Convert(converter=…).
```

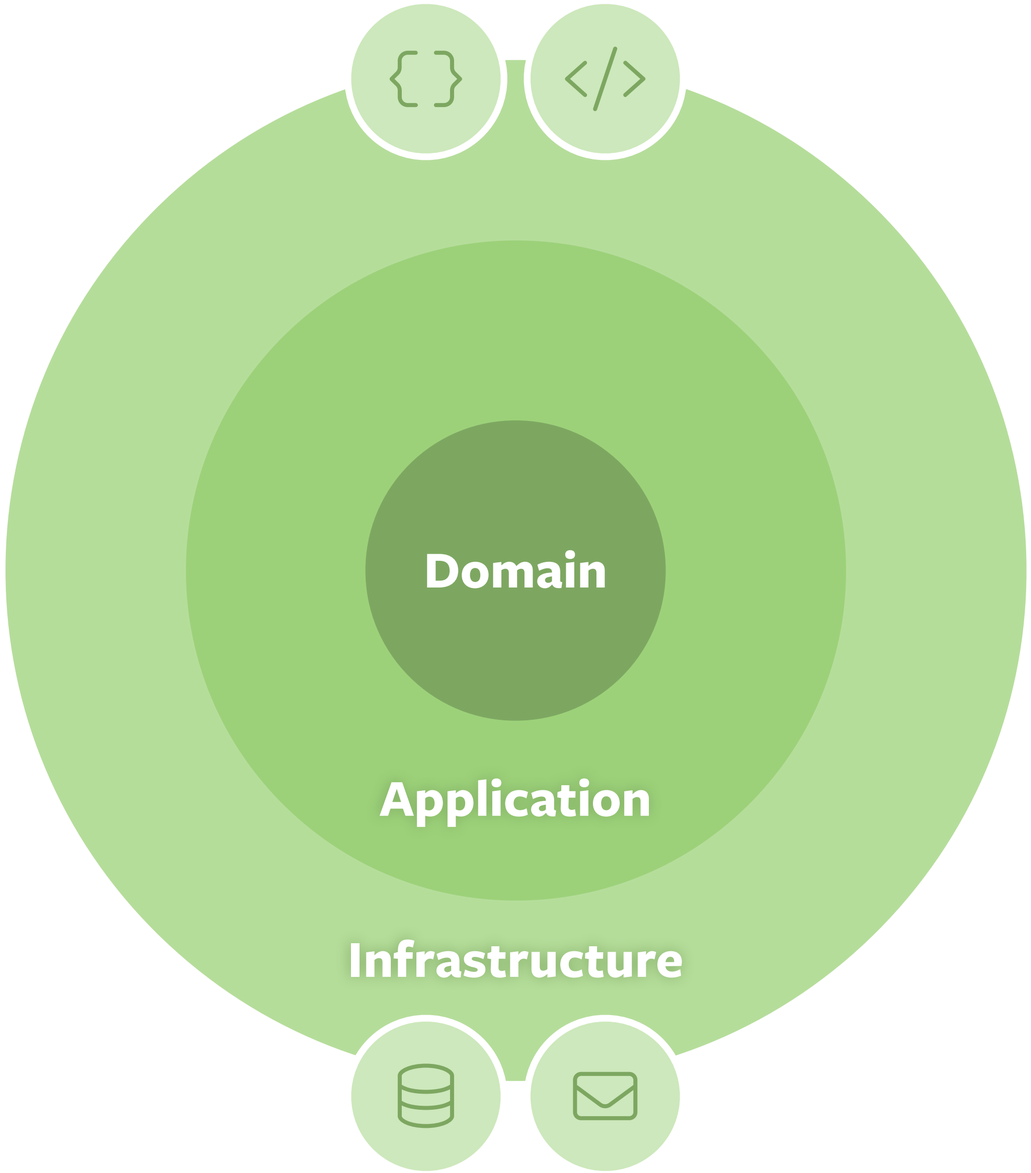
```
@Entity
@NoArgsConstructor(force = true)
@EqualsAndHashCode(of = "id")
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order {

  private final @EmbeddedId OrderId id;

  @OneToMany(cascade = CascadeType.ALL)
  private List<LineItem> lineItems;
  private CustomerId customerId;

  public Order(Customer customer) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customerId = customer.getId();
  }

  @Value
  @RequiredArgsConstructor(staticName = "of")
  @NoArgsConstructor(force = true)
  public static class OrderId implements Serializable {
    private static final long serialVersionUID = …;
    private final UUID orderId;
  }
}
```

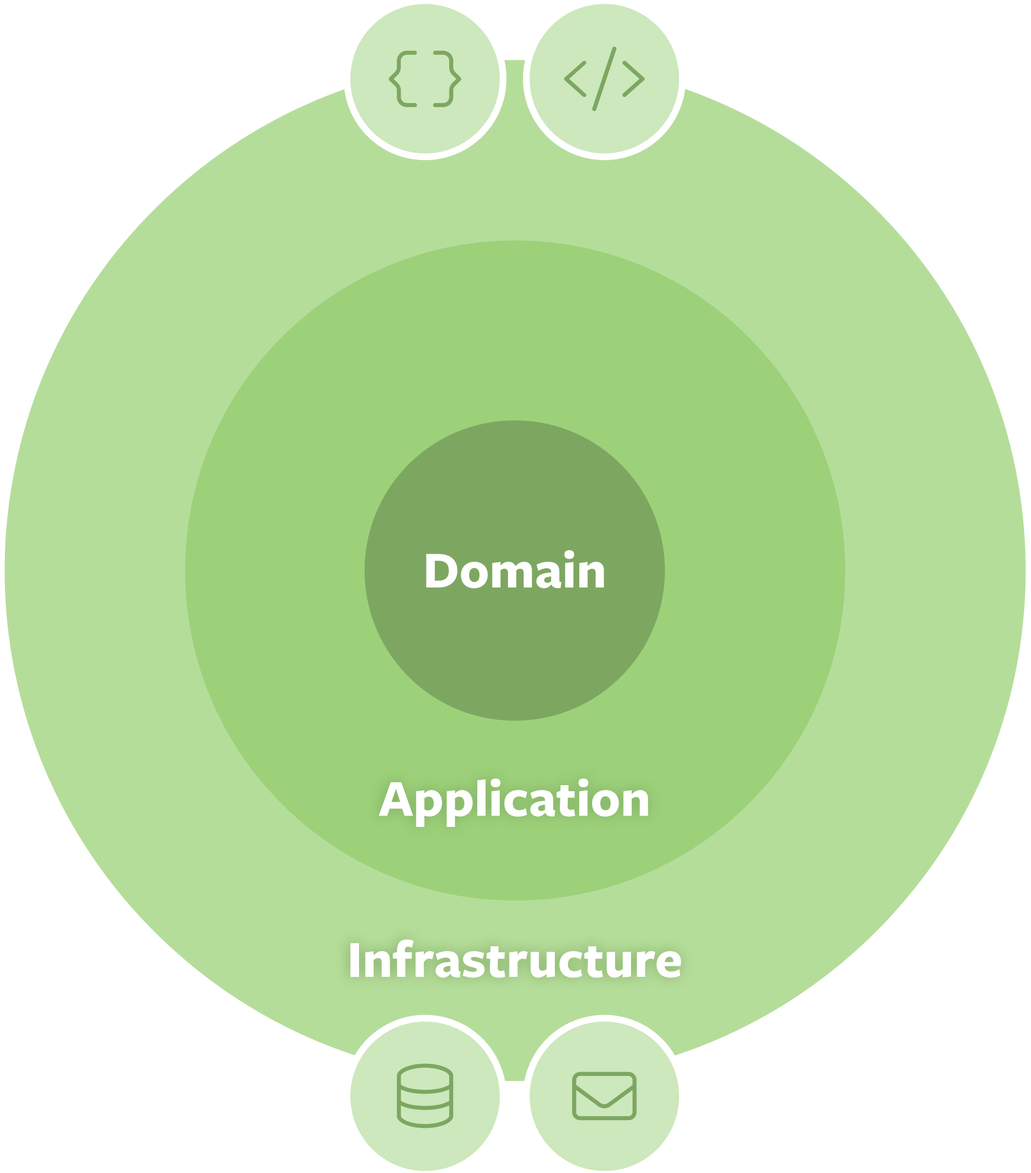
```
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order implements AggregateRoot<Order, OrderId> {

  private final OrderId id;
  private List<LineItem> lineItems;
  private Association<Customer, CustomerId> customer;

  public Order(CustomerId customerId) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customer = Association.forId(customerId);
  }

  @Value(staticConstructor = "of")
  public static class OrderId implements Identifier {
    private final UUID orderId;
  }
}
```

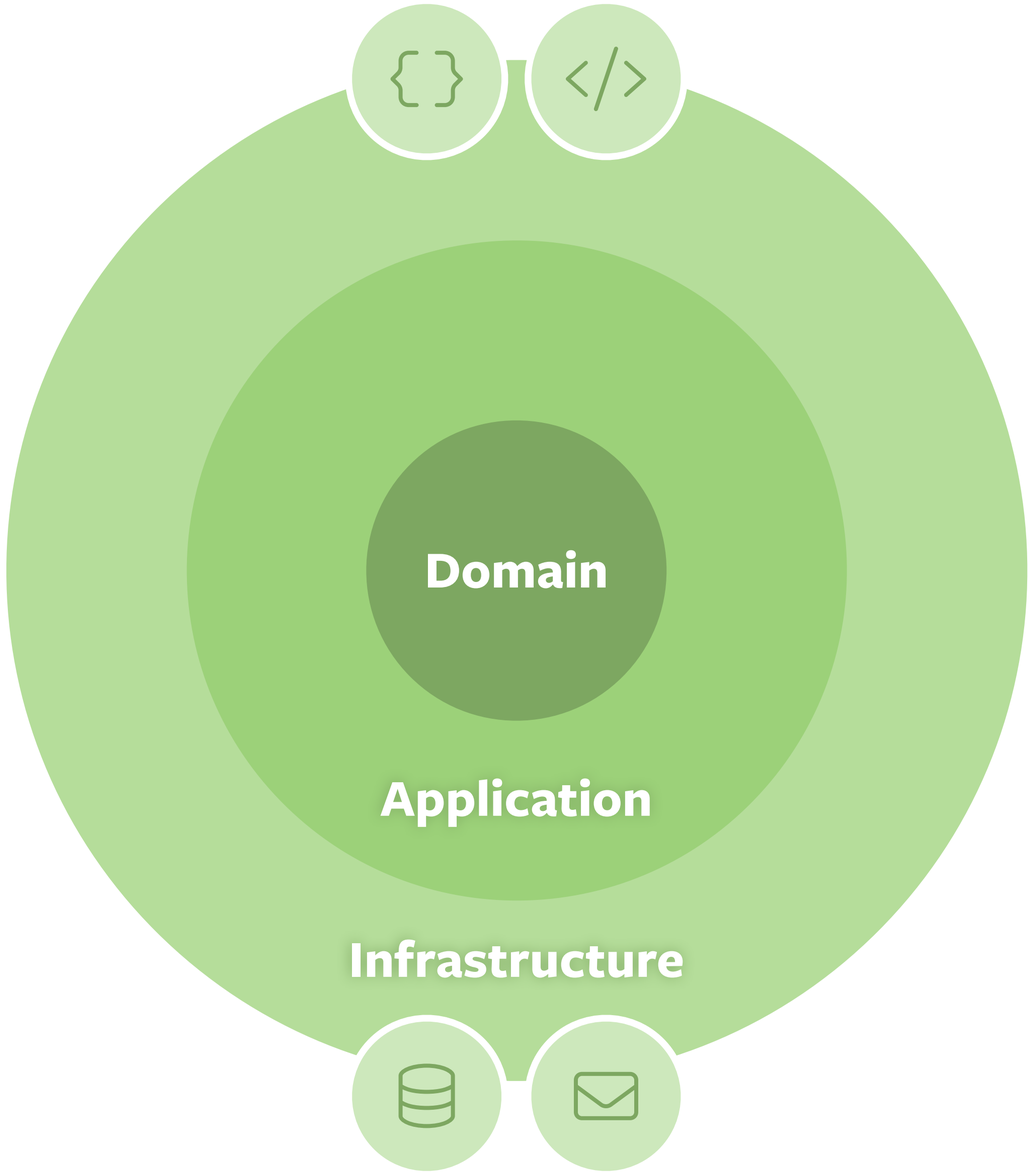


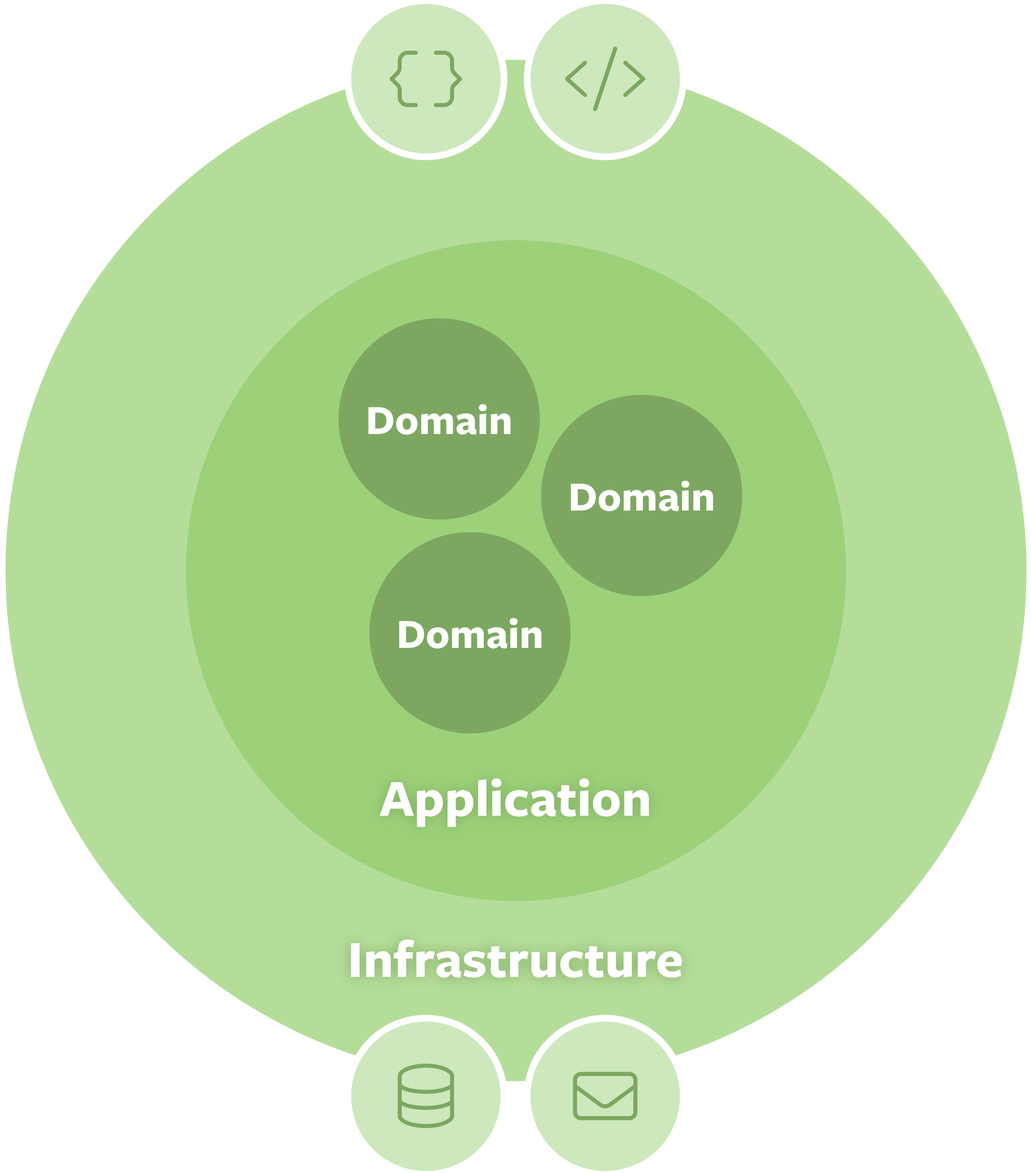
Separation of Concerns Architectures

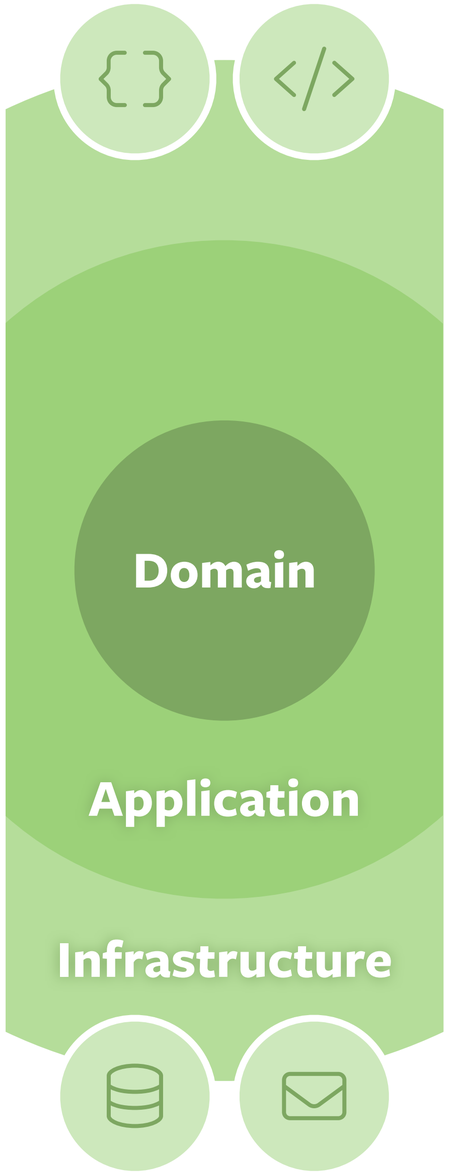

Domain
Application
Infrastructure

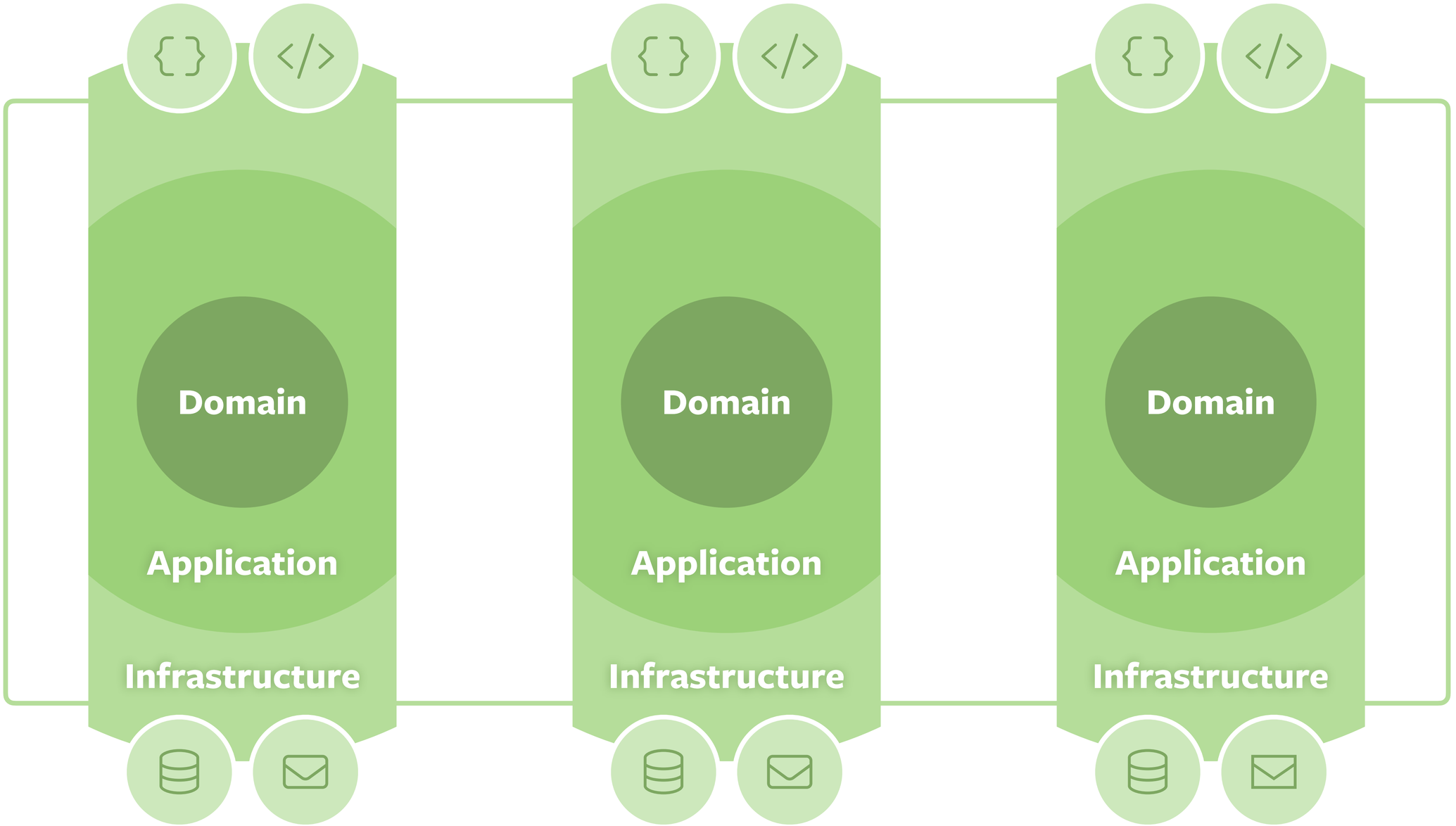

Domain
Application
Infrastructure

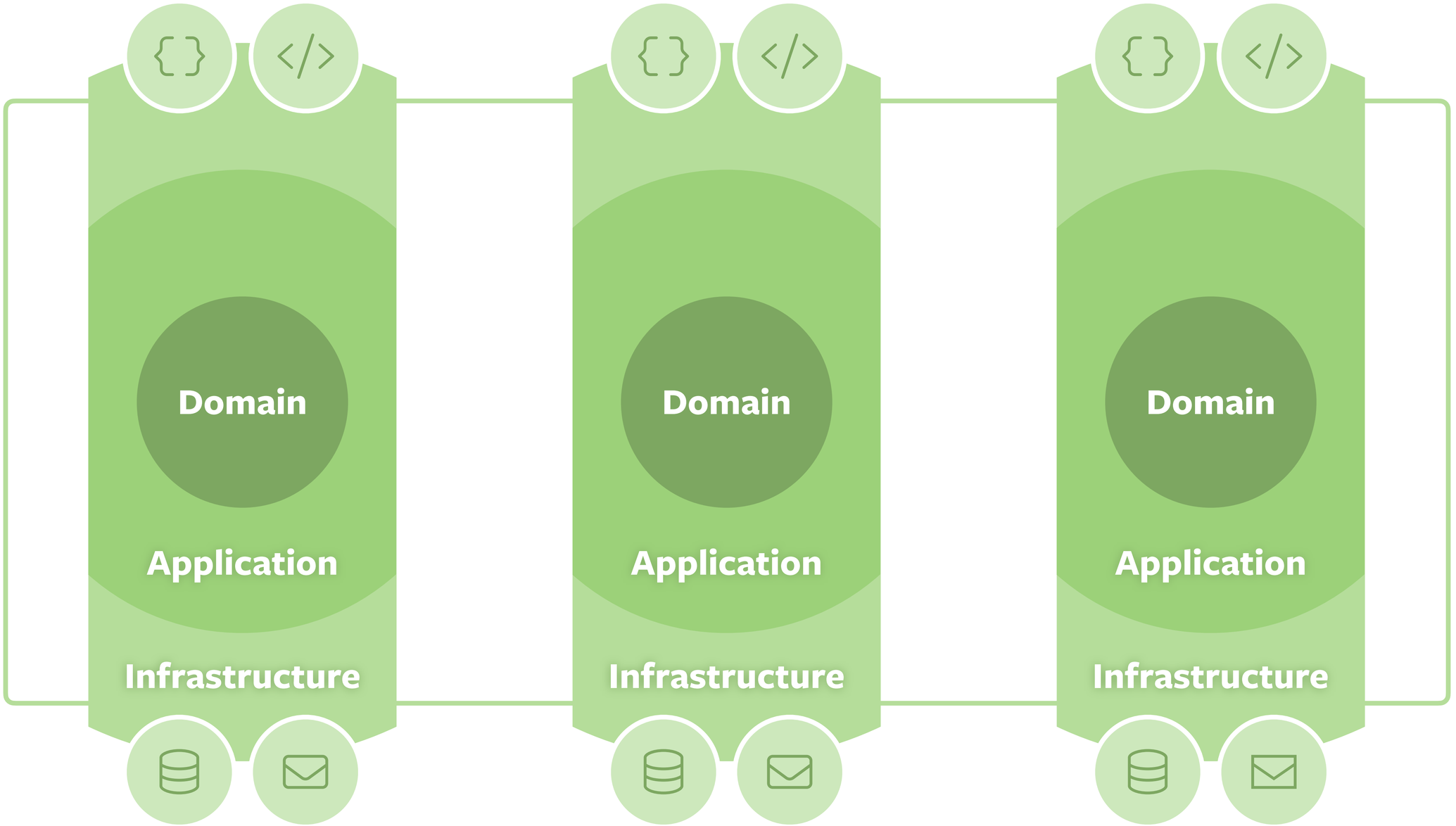

Domain
Application
Infrastructure

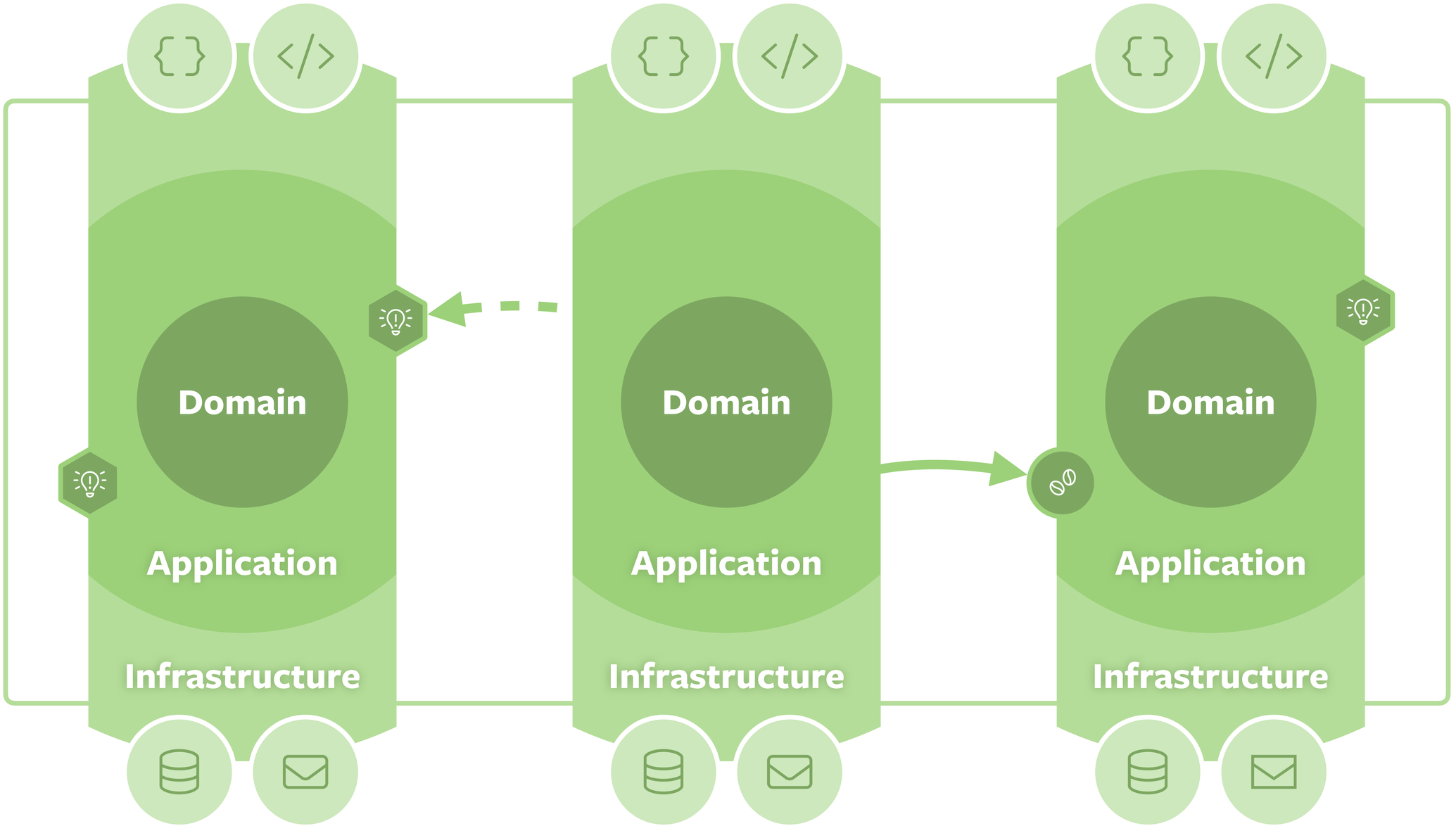

Domain
Application
Infrastructure

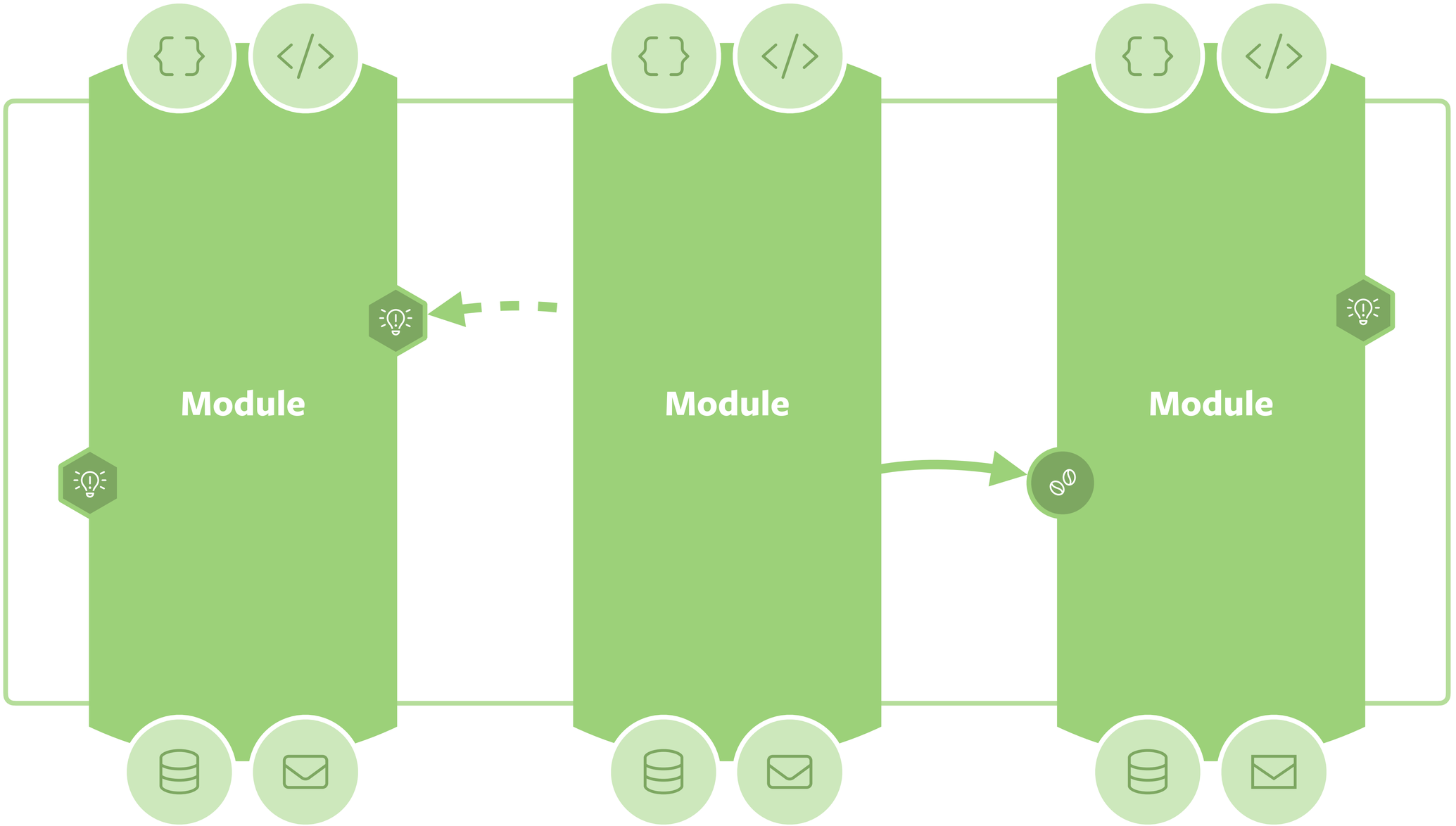

Domain
Domain
Domain
Application
Infrastructure


Domain
Application
Infrastructure


Domain
Application
Infrastructure
Domain
Application
Infrastructure
Domain
Application
Infrastructure


Domain
Application
Infrastructure
Domain
Application
Infrastructure
Domain
Application
Infrastructure


Domain
Application
Infrastructure
Domain
Application
Infrastructure
Domain
Application
Infrastructure

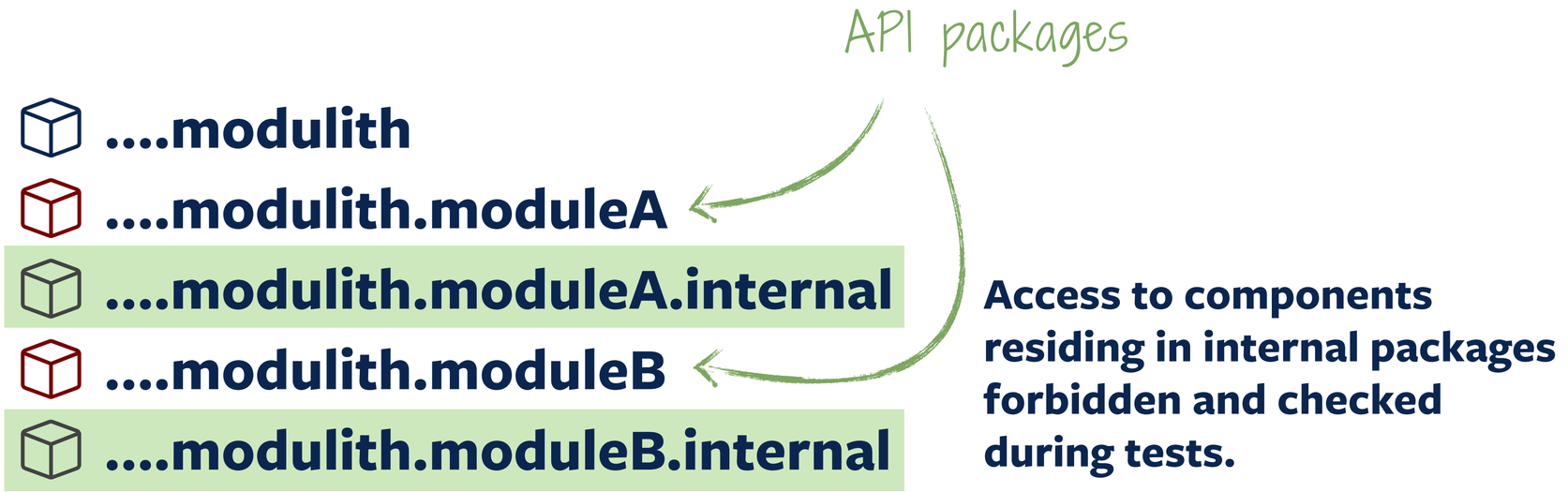

Module
Module
Module

Spring Modulith

```
package com.acme.modulith

@SpringBootApplication
class MyApplication { … }
```

Standard Spring Boot Application

Package Conventions

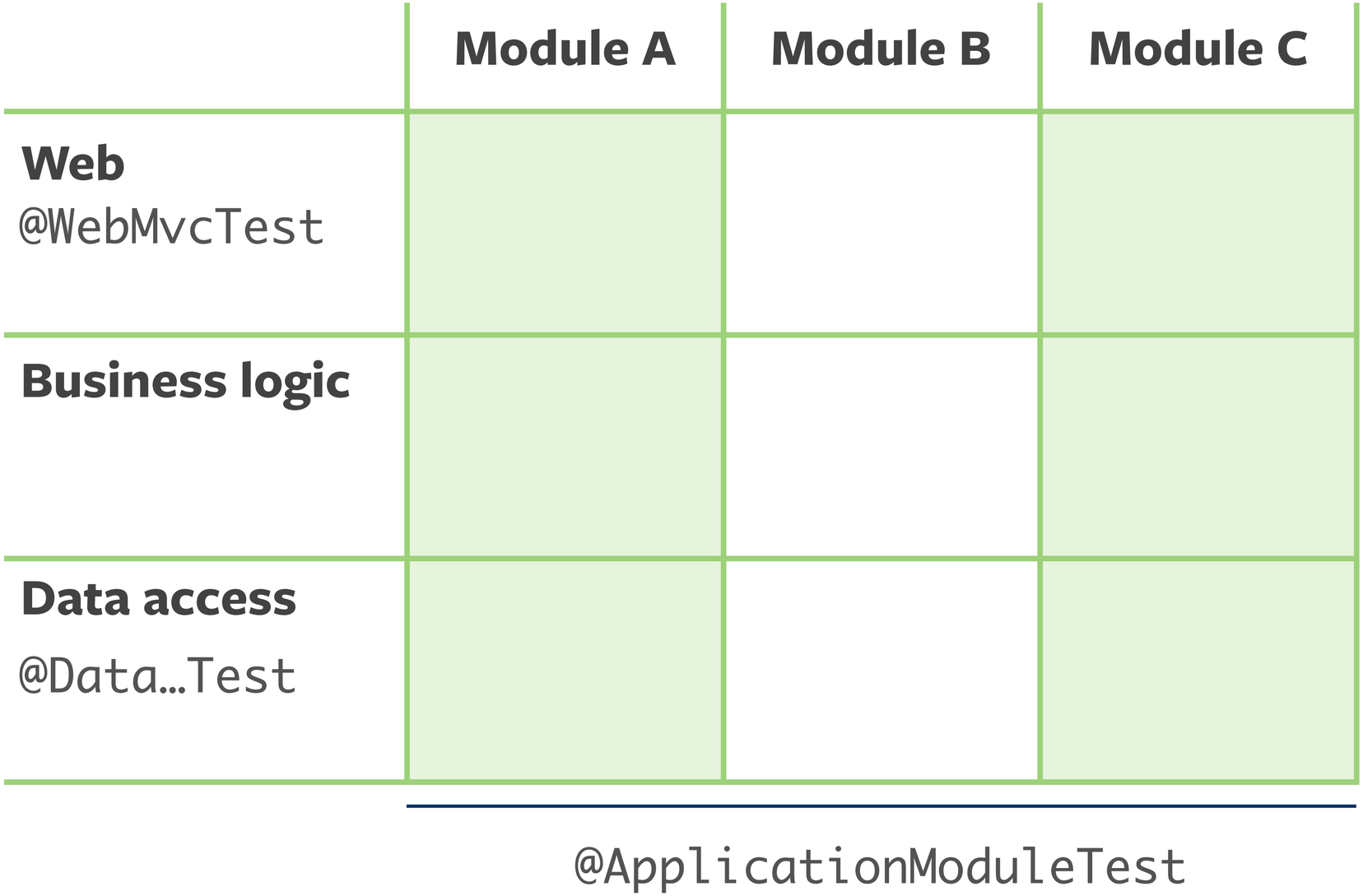


```
package com.acme.modulith

@SpringBootApplication
class MyApplication { … }
```

Standard Spring Boot Application

```
var modules =
  ApplicationModules.of(MyApplication.class);
modules.verify(…);
```

Verifies rules for MyApplication

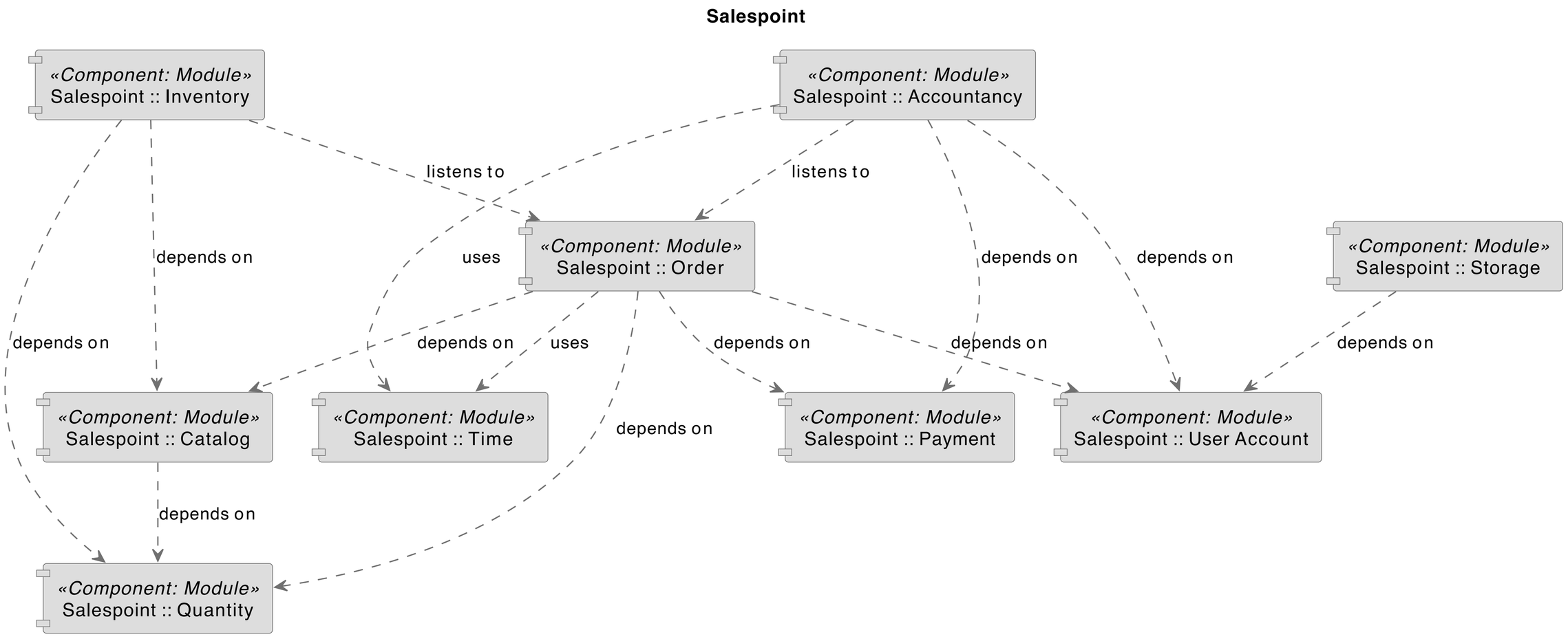

Module A
Module B
Module C
Web
@WebMvcTest
Business logic
Data access
@Data…Test
@ApplicationModuleTest

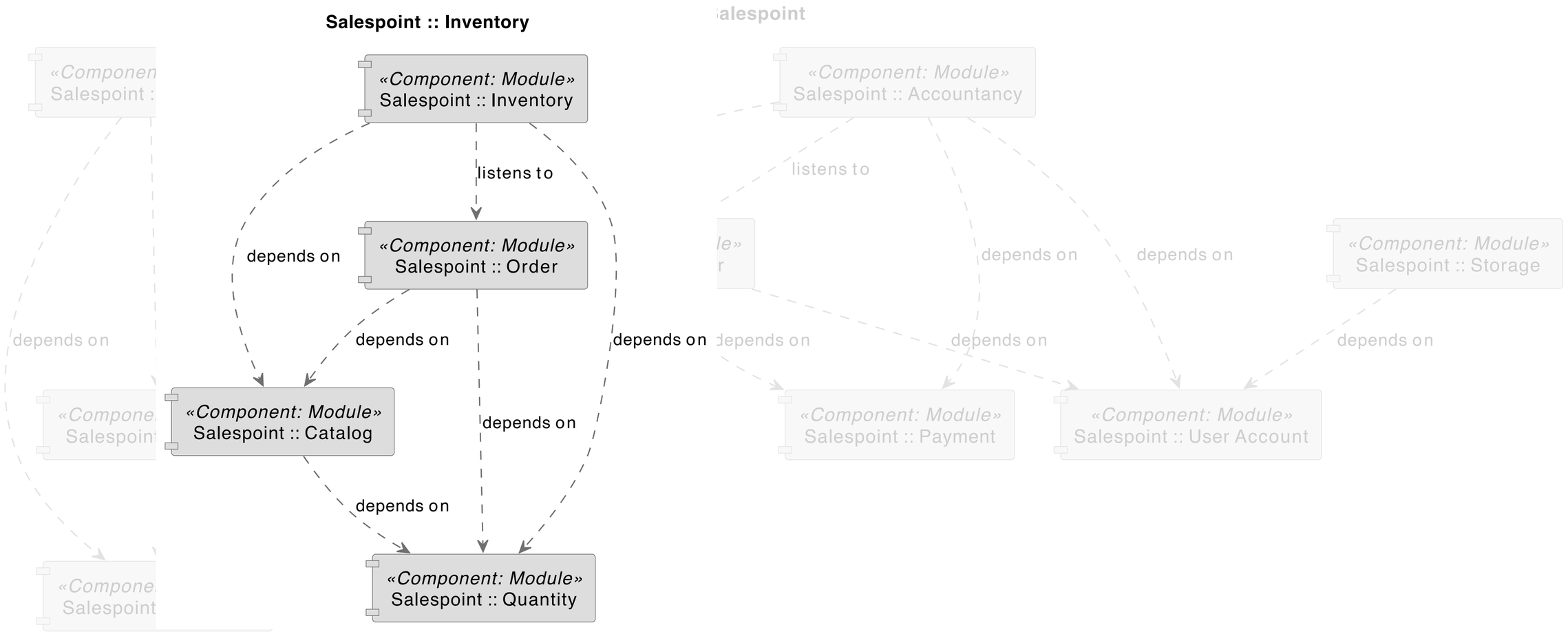
Provided Interface

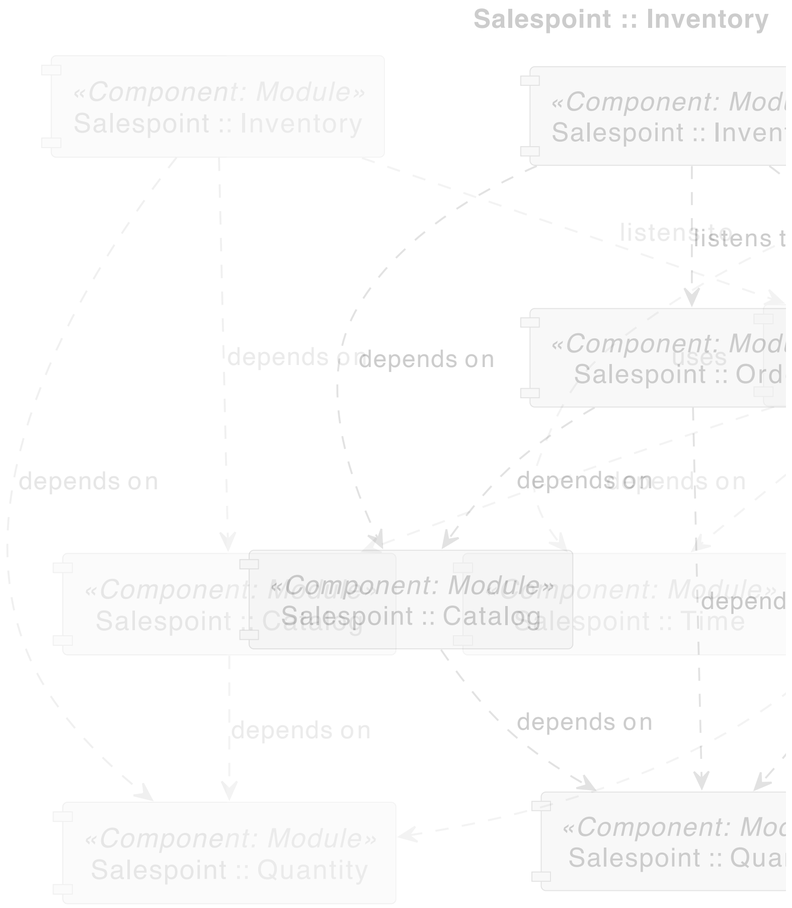
- **Exposed Service API**
  Spring Beans available for DI
- **Exposed Aggregates**
  Primary elements of the domain and constraints
- **Published Events**
  Events the component emits

Required Interface

- **Consumed Service API**
  External dependencies of Spring beans
- **Configuration**
  Spring Boot configuration properties
- **Consumed Events**
  Events that the component reacts to


Salespoint
«Component: Module»
Salespoint :: Inventory
«Component: Module»
Salespoint :: Accountancy
«Component: Module»
Salespoint :: Order
«Component: Module»
Salespoint :: Storage
«Component: Module»
Salespoint :: Catalog
«Component: Module»
Salespoint :: Time
«Component: Module»
Salespoint :: Payment
«Component: Module»
Salespoint :: User Account
«Component: Module»
Salespoint :: Quantity
listens to
listens to
depends on
uses
depends on
depends on
depends on
depends on
depends on
uses
depends on
depends on
depends on
depends on
depends on


Salespoint :: Inventory
«Component: Module»
Salespoint :: Inventory
listens to
depends on
«Component: Module»
Salespoint :: Order
depends on
depends on
depends on
depends on
«Component: Module»
Salespoint :: Catalog
depends on
depends on
«Component: Module»
Salespoint :: Quantity
alespoint
«Component: Module»
Salespoint :: Accountancy
listens to
depends on
depends on
depends on
depends on
depends on
«Component: Module»
Salespoint :: Storage
«Component: Module»
Salespoint :: Payment
«Component: Module»
Salespoint :: User Account



Base package	`org.salespointframework.useraccount`
Spring components	*Services* • `o.s.u.UserAccountManagement` (via `o.s.u.PersistentUserAccountManagement`) *Others* • `o.s.u.AuthenticationManagement` (via `o.s.u.SpringSecurityAuthenticationManagement`)
Aggregate roots	• `o.s.u.UserAccount`
Value types	• `o.s.u.EncryptedPassword` • `o.s.u.UnencryptedPassword` • `o.s.u.Role`
Published events	• `o.s.u.UserAccountCreated` created by: ◦ `o.s.u.UserAccount.onCreate()`
Properties	• `salespoint.authentication.login-via-email` — `java.lang.Boolean`, default `false`. Enables the login procedure to use the email address to lookup a user instead of their username. Defaults to `false`.

Summary

Architectural concepts...

... are explicitly expressed in the code.

... are predefined based on established pattern languages.

... are defined by jMolecules (concepts)
and tool & framework integration (rules).

Architecturally Evident Applications – How to Bridge the Model-Code Gap?

Oliver Drotbohm (odrotbohm@vmware.com) in collaboration with
Henning Schwentner (henning.schwentner@wps.com) & Stephan Pirnbaum (stephan.pirnbaum@buschmais.de)
February 2022

Abstract

Over the course of its lifetime, every non-trivial piece of software will significantly grow in complexity. The extent of that growth significantly affects the ability to evolve it to avoid having to replace it with a costly rewrite eventually. Thus, managing that complexity has been the topic of interest in the software community, and architectural and design pattern languages have been identified as a means to achieve that. But even if the conceptual models of an application use that language, a fundamental challenge remains: how to express those abstract concepts in the actual codebase?

This paper explores a novel approach that enables developers to explicitly express architectural and design concepts in application code, which ultimately enables:

- *Understandability* – By finding the architectural language in code, it is easier for developers to understand the code base, relate individual elements of it to the bigger picture and, ultimately, make architecture more accessible.
- *Documentation* – With abstract concepts present in the code base, we can extract documentation about it that is correct by definition and describes it at an architectural abstraction level.
- *Verification* – We can verify that our implementation adheres to the rules associated with the concepts that the individual elements of the code base implement at different levels of architectural abstraction.
- *Reduction of boilerplate code* – At the application boundaries, domain model elements have to be persisted into some data store or exposed to clients by APIs. Architectural concepts, directly expressed in those elements, allow transparently defaulting such mappings into popular implementation technologies.

This paper presents the fundamental idea in detail, as well as a Java library to express architectural and design concepts, and contrasts it to alternative approaches. It concludes with a presentation of the support of that library in a variety of associated integration technologies to implement the aspects described above.

Introduction

Bridging the gap between architectural patterns and code bases has been an ongoing challenge when writing long-living business software. We would like to present an approach to express these patterns directly in code by using programming-language-specific means and describe how that approach becomes an enabler to create code that is more expressive, more understandable, more correct and ultimately easier to change. The paper uses Java as an example because it is a very ubiquitous language in enterprise applications. However, the approach can be transferred to other languages, too.

Over the last 1.5 years, a prototypical implementation has been implemented in a cross-company collaboration effort between VMware, WPS Solutions (Hamburg) BUSCHMAIS (Dresden). It can be found under a pr named jMolecules on GitHub [jmolecules].

Fundamentally, we need a mechanism to express architectural artifacts in the codebase. In Java, two primary language constructs are great candidates t achieve this: annotations and types. We will have a detailed look at the pros and cons of each later. jMolecules currently provides annotations for the following architectural concepts: the Domain-Driv Design (DDD) building blocks described in [evanso events and event listeners, and the parts of particu architectural styles, such as onion architecture [palermo08], layered architecture, and CQRS syste The DDD and event concepts are also available as J interfaces alternatively.

Developers can refer to the concept library in their application build files so that the architectural definitions become an inherent part of the code base. This results in more expressive code that has a more direct connection to the architectural model in the first place and, thus, supports understanding the implementation. The metadata available within the code enables extensive integration with external technology to verify the implementation against the model expressed in the code and extract architecture and developer documentation. To run on ubiquitous technical platforms (such as Spring Framework [spring]) and integrate seamlessly with persistence technology (such as the Jakarta Persistence API (JPA) [jpa] or commonly used serialization APIs like Jackson [jackson]), domain code usually has to be augmented with boilerplate code, like annotations or additional models which significantly increases the accidental complexity of applications.

@mawspring

Links

- **xMolecules**
  https://xmolecules.org
- **jMolecules**
  https://jmolecules.org
- **jMolecules Examples**
  https://github.com/xmolecules/jmolecules-examples
- **Gitter – Join the community!**
  https://gitter.im/xmolecules/xmolecules

Resources

- **Software Architecture for Developers**
  Simon Brown – Books
- **Just Enough Software Architecture**
  George Fairbanks – Book
- **Architecture, Design, Implementation**
  Ammon H. Eden, Rick Kazman – Paper
- **Sustainable Software Architecture**
  Carola Lilienthal – Book
- **The Programmer's Brain**
  Felienne Hermans – Book

Thank you!
Questions?

Paper

Book

Oliver Drotbohm odrotbohm oliver.drotbohm@broadcom.com