Domain-Driven Design

An approach to software development that centers on the business domain.

Shane Rowley
Architect

Javier Ochoa
Senior Consultant

Red Hat

What we'll discuss today?

- Introductions
- What is DDD?
- Bounded Context example
- DDD process
  - Event Storming
  - Context Map
  - Context Canvas
- Closing remarks
- Q&A

Red Hat

Intros

Red Hat

“Domain-Driven Design is an approach to software development that centers the development on programming a domain model that has a rich understanding of the processes and rules of a domain. The name comes from a 2003 book by Eric Evans that describes the approach through a catalog of patterns. ... The approach is particularly suited to complex domains, where a lot of often-messy logic needs to be organized.”

DomainDrivenDesign

martinfowler.com

DDD & Strawberry Ice Cream

One of the goals of Domain Driven Design, is to deliver software that directly reflects the common language of a given domain.

Let's use Strawberry Ice Cream to illustrate the concepts of DDD.

Our illustration is going to cover the process of making Ice Cream, with a focus on the Strawberry.

DDD & Strawberry Ice Cream

Let's first talk about the high-level flow:

Strawberries are:

1. Grown on a farm
2. Harvested
3. Transported to a distributor
4. Delivered to an ice cream maker
5. Put into strawberry ice cream

Forget the old ways

Don’t let the old school teaching of **Normalization** constrain you - it is not relevant anymore!

- The same entity can exist in more than one context.

- But it might look very different in each.

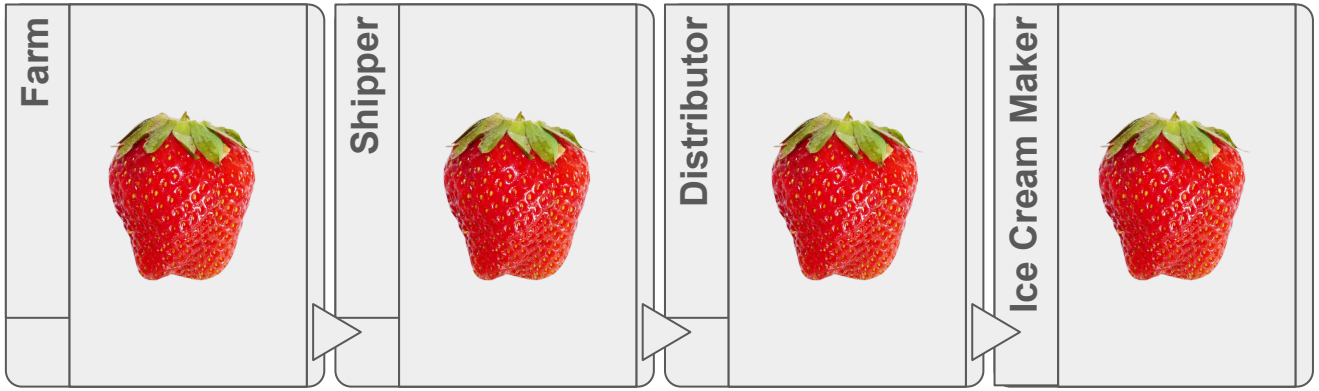
- The wrong way to approach this...
  is to build a giant database for all things Strawberry...
  Don't Do That! In fact, that's what the legacy system
  probably looks like already... (more on that in a bit)




Think about the Domains

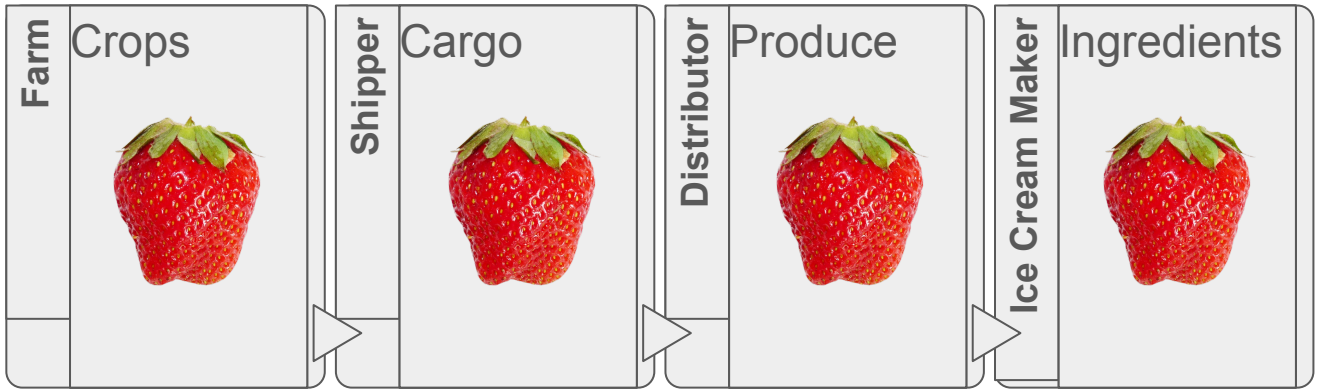
Presented for your consideration:

The **Domains** of making Strawberry Ice Cream, from the perspective of the **Strawberry**



Think about the Domains

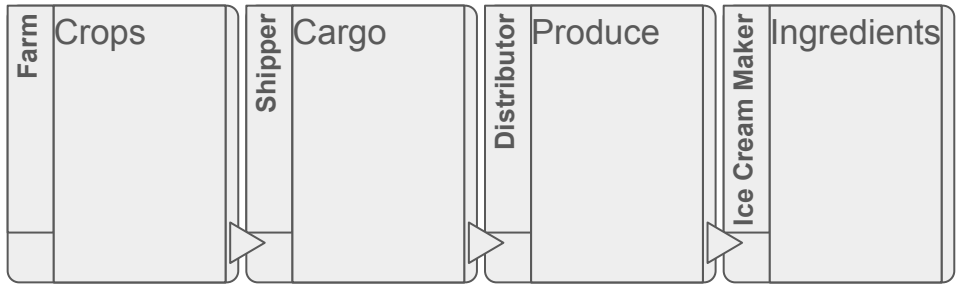
Notice how each Domain has its own unique language and vocabulary



- Farm – Produces Crops
- Shipper – Transports Cargo
- Distributor – Receives and Ships Produce
- Ice Cream Maker – Uses Ingredients

Think about the Domains

If we interview each of the four Domains, we will discover that each one has a drastically different understanding of a Strawberry.



Imagine trying to create a single data store to define the whole model.

Farmer:
- Crop Yield
- Soil Quality
- Moisture
- Sun Exposure

Distributor:
- Inventory
- Shelf Life
- Demand
- Supply

Shipper:
- Temperature
- Route
- Weight
- Volume

Ice Cream Maker:
- Color
- Flavor
- Texture

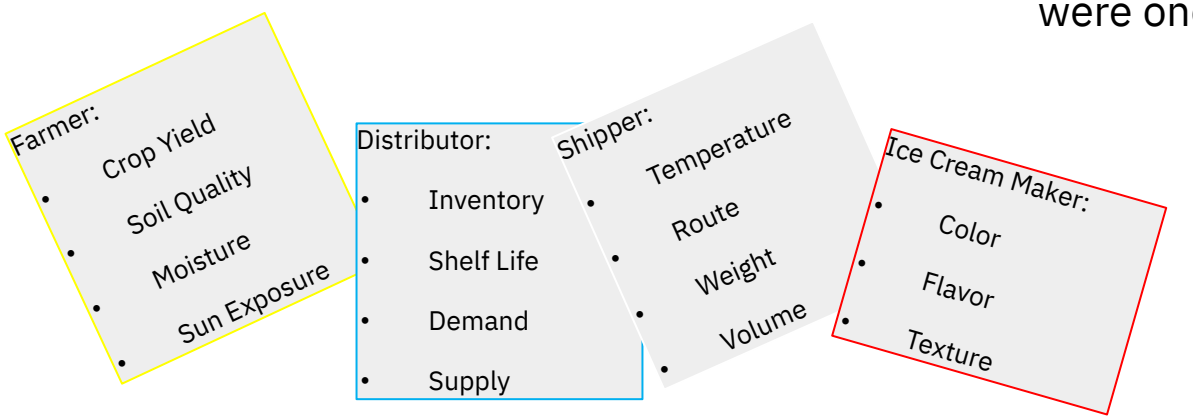
Monoliths are a Normal Evolution

If our Ice Cream Maker started out as a small business and adopted technology early on, then the data model is likely a giant monolith that evolved with the business.

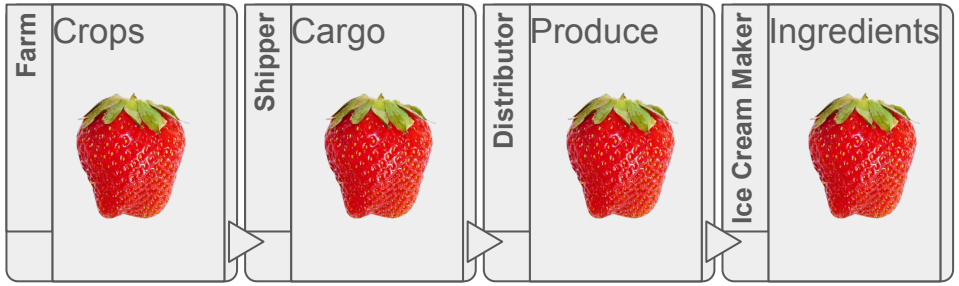
Initially, IT and the business were one unit...

As the business grew, it subdivided itself into departments...

Each of those departments began to evolve its own vocabulary...



IT became its own unit. With databases, middleware, services, websites, security, etc... & its own vocabulary...



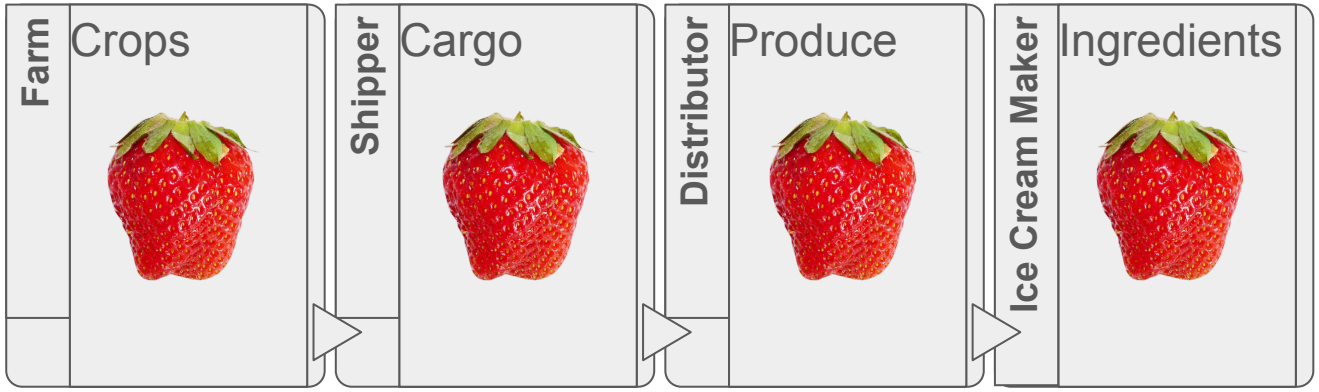
Eventually, the systems no longer aligned to the structure of the business...

It's time to modernize!

Think about the Domains

So… while all of the **Domains** have an **Entity** called a **Strawberry**, it is NOT the same entity. Each Domain has its own unique understanding of a Strawberry.

This is why separating Domains along their Context boundaries is so important.

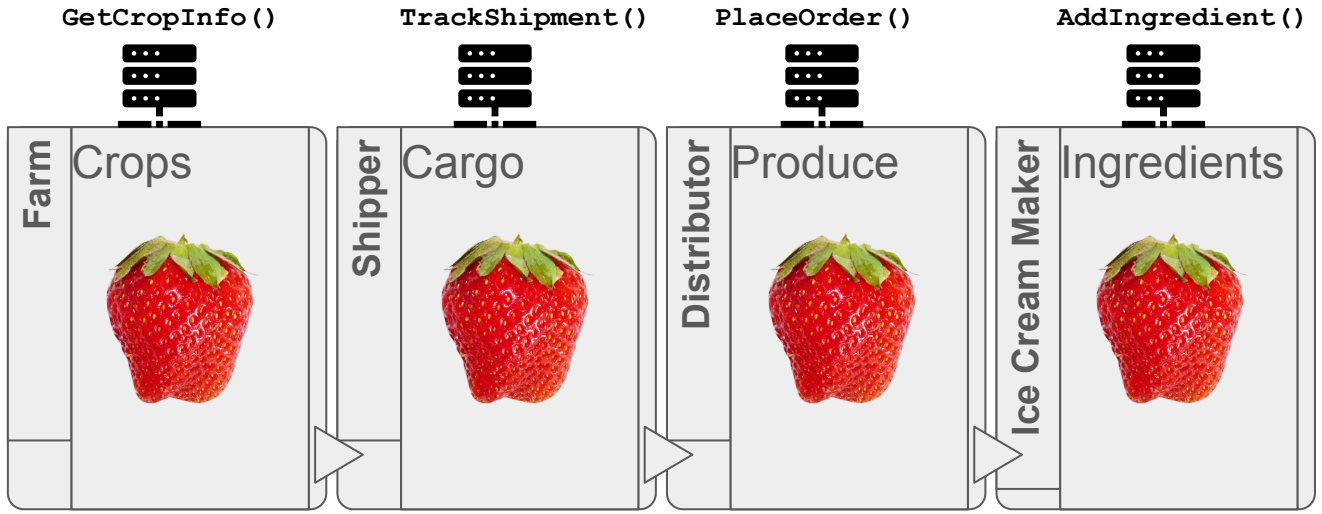


- Farm – Strawberry is a Crop
- Shipper – Strawberry is Cargo
- Distributor – Strawberry is Produce
- Ice Cream Maker – Strawberry is an Ingredient

Think about the Domains

So… How do I get a view of the whole lifecycle of a Strawberry from the perspective on an Ice Cream Maker???

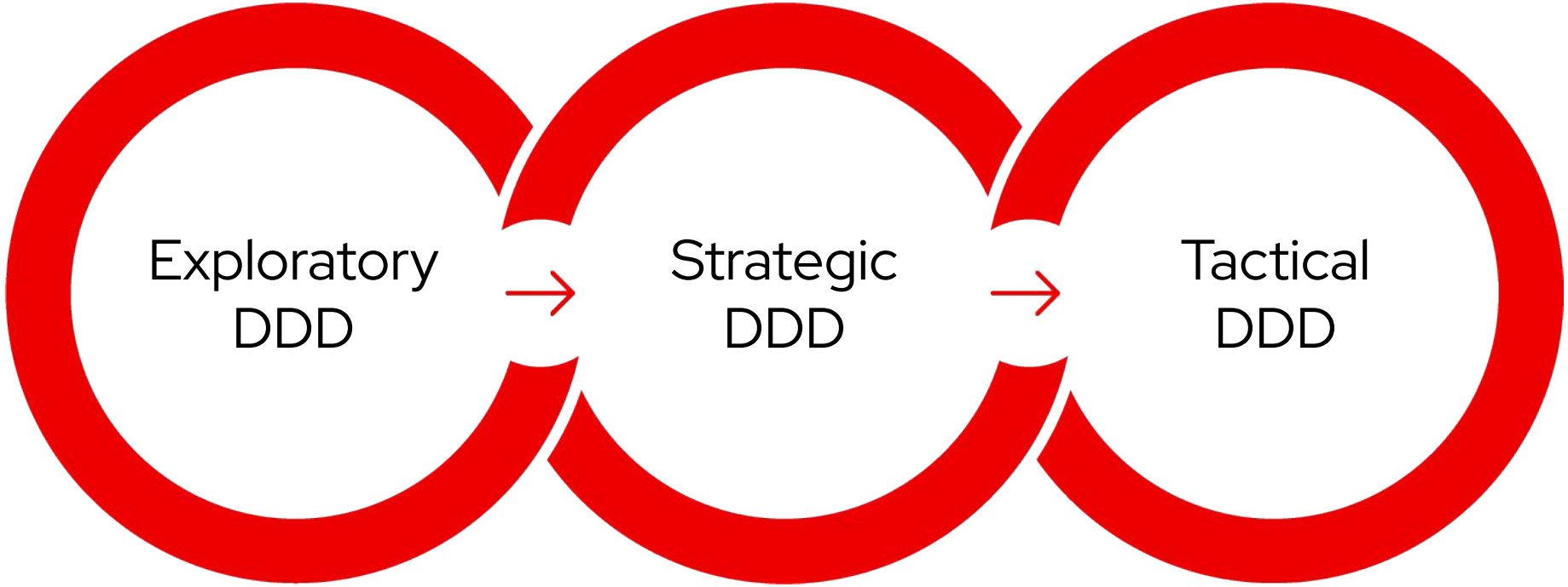
Each Domain will expose services that other domains can consume in order to implement cross-cutting business processes. aka – Experience Layer or Value Stream



Domain Driven Design

A philosophy for developing software systems that encourages

Domain Thinking at each step



Red Hat

DDD Process

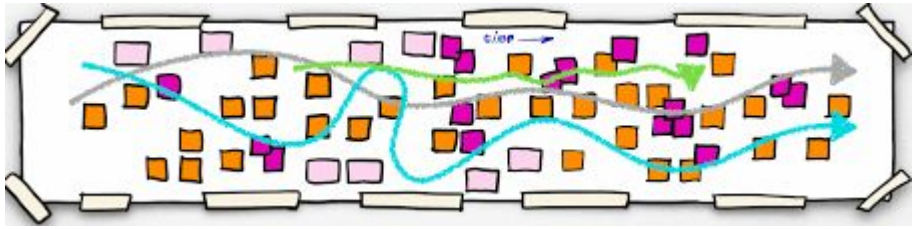
01	Event Storming	• Business and Technology SMEs • Capture domain events based on business language • Include system names and highlight hotspots
02	Context Mapping	• Group events into a possible separate context • Diagram how will it interact with other contexts? • Include dependencies to external systems
03	Context Canvas	• For each context identified, define the index card • Fill in as much information as captured during event storming: hotspots, risks/unknowns, etc.

Red Hat

Event Storming

Event Storming is a communicative brainstorming method in which knowledge and understanding of a specific, delimited field of knowledge (a domain of expertise) is jointly developed and visualised in a workshop. The starting point are so-called domain events.

https://en.wikipedia.org/wiki/Event_storming



Red Hat

Event Storming with colored stickies!

Organized chaos!

Domain Event

Domain Event
An event that occurs in the business process.
Written in past tense.

User/ actor

User/Actor
A person who executes a command through a view.

Flow/ feature

Flow/feature
Function within your domain/sub-domain that can be used as a topic for event storming

Policy

Policy
Special policy for certain domain events to enhance the understanding of the flow.

System / App

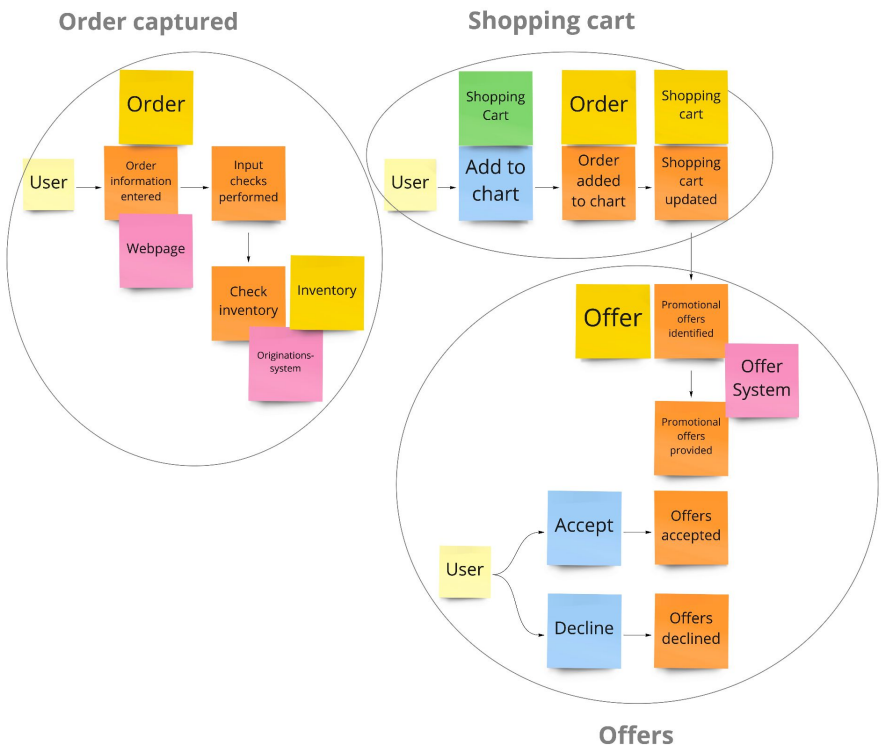
System/app
System or application in charge or related to the domain events in proximity

Concept

Concept (aggregate)
Something that is important enough to capture as business concept related to domain events

Hostpot

Question Marks / Risks / Hotspots
Use red Post-Its for unclear topics or questions that arise during the session.



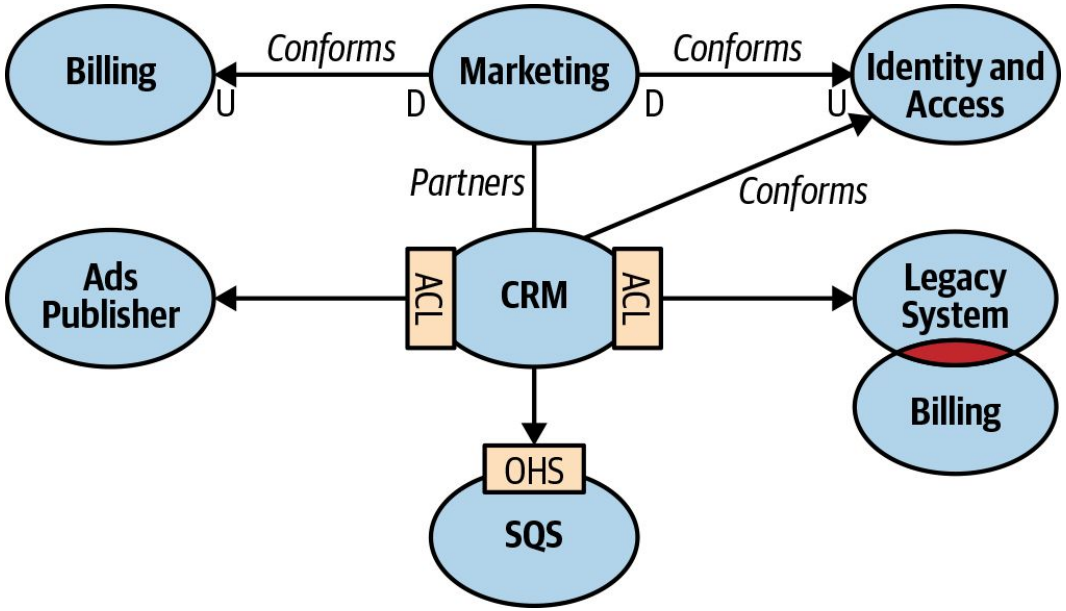
https://techsnack.orbitdigital.de/posts/event-storming

Red Hat

Context Map

"Context Maps describe the contact between bounded contexts and teams with a collection of patterns. There are nine context map patterns and three different team relationships. The context map patterns describe a variety of perspectives like service provisioning, model propagation or governance aspects. This diversity of perspectives enables you to get a holistic overview of team and bounded context relationships."

https://github.com/ddd-crew/context-mapping

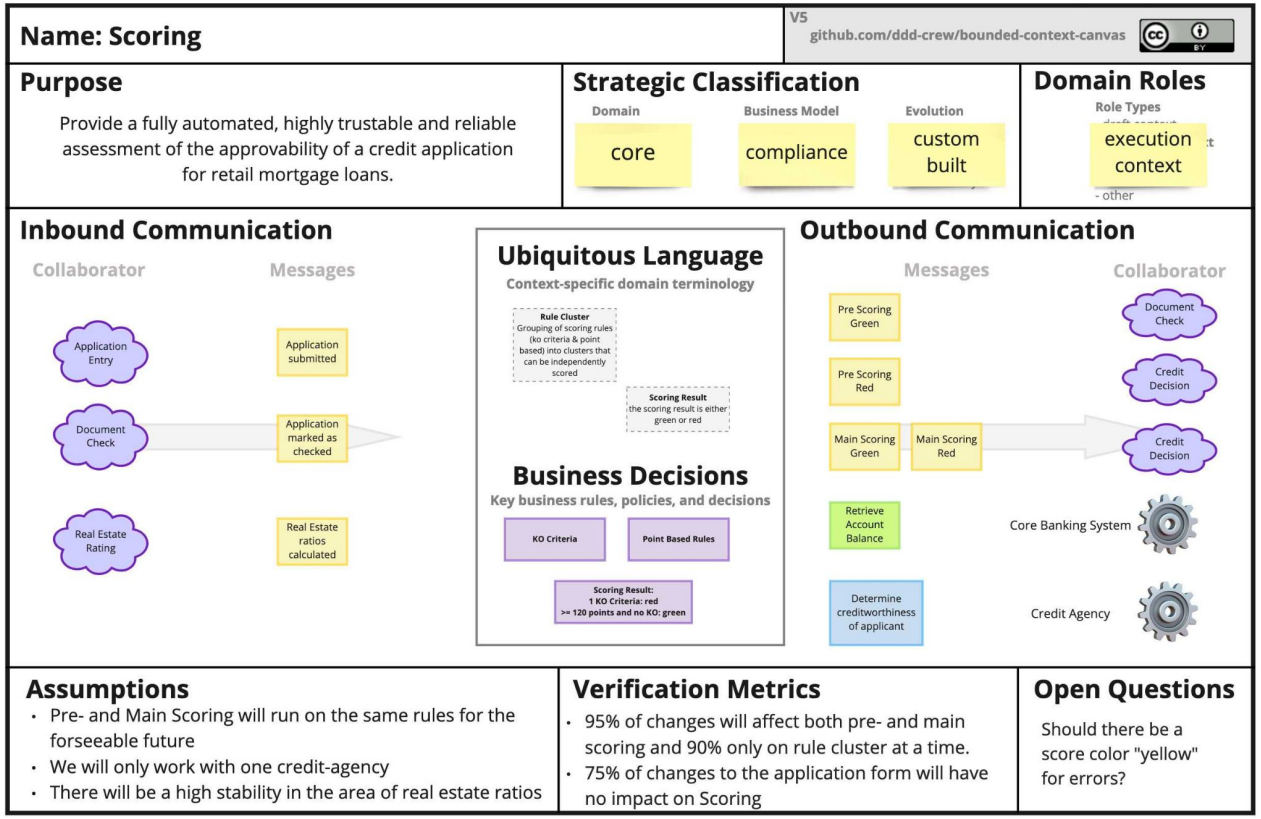


Red Hat

Context Canvas

The Bounded Context Canvas is a collaborative tool for designing and documenting the design of a single bounded context.

https://github.com/ddd-crew/bounded-context-canvas



Red Hat

Links:

Domain Driven Design Quickly
domain-driven design is a development approach to managing software for complex domains
GitHub - ddd-crew/welcome-to-ddd: Definitions of DDD and fundamental concepts to reduce the learning curve and confusion
GitHub - ddd-crew/bounded-context-canvas: A structured approach to designing and documenting each of your bounded contexts
Virtual DDD

Books:

Domain--Driven Design Reference
Patterns, Principles, and Practices of Domain-Driven Design

Red Hat

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat