DDD-rail Your Monorail

Breaking up the Rails monolith with domain driven design

Once upon a time

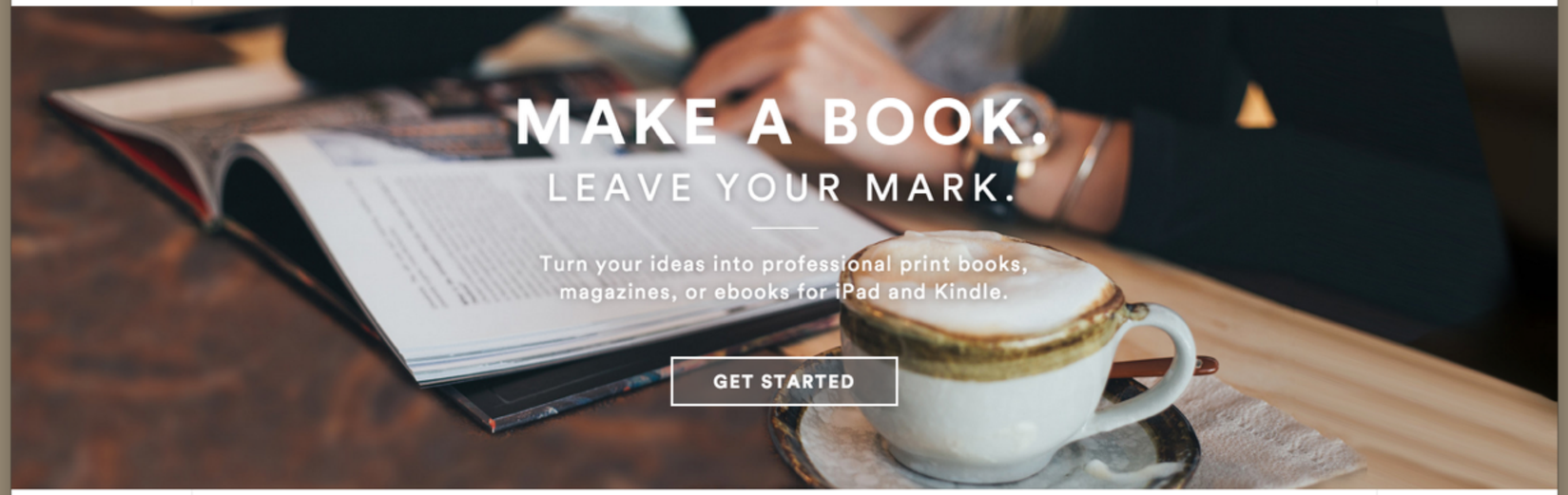
Inbox (5) - andrew | Portlandia by And | andrewhao — Bitb | andrewhao/dddra | PlantUML | Project Lead - Ca | Blurb | Andrew (Carbon…

www.blurb.com

My Dashboard Settings Sign out Help

blurb

FORMATS & PRICING CREATE SELL & DISTRIBUTE BOOKSTORE



CREATE

Make print and ebooks using our free easy-to-use software

PRINT

Professional-quality books and magazines —print one copy or many

SELL

Sell books on your website, Amazon, and wherever books are sold

FORMATS AND PRICING

Blurb: self publishing platform

Portlandia by Andrew Hao

www.blurb.com/b/2783461-portlandia

blurb Portlandia

Join Blurb Now Sign In Help

Portlandia

Tales from Stumptown

by Andrew Hao

share Like 0

Comments

Softcover	US $19.39
Hardcover, Dust Jacket	US $32.39
Hardcover, ImageWrap	US $34.39

Quantity 1

Add To Cart

ABOUT THE BOOK

An epic road trip from Oakland to Portland. Highlights include:

- Discovering the existence of Javanilla Shakes at a local Seattle's Best Coffee.
- Stumptown Coffee Roasters. 'Nuff said.
- Voodoo Donuts bacon maple bars. 'Nuff said, again.
- A Crater Lake detour
- Extended time spent at Powell's City of Books
- Street food
- More time at Powell's
- Quality time with Mssr. Dan Garcia
- Hipsters, hipsters, hipsters

All photos credit Jen Chen (www.stillhouette.com)

Dimensions Small Square 52 pgs
Category Travel
Tags road trip, javanilla shake, oregon, sbc, friends, pdx ...

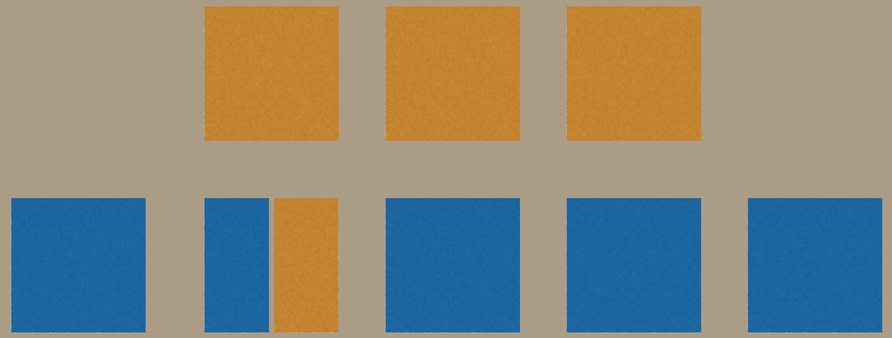
ABOUT THE AUTHOR

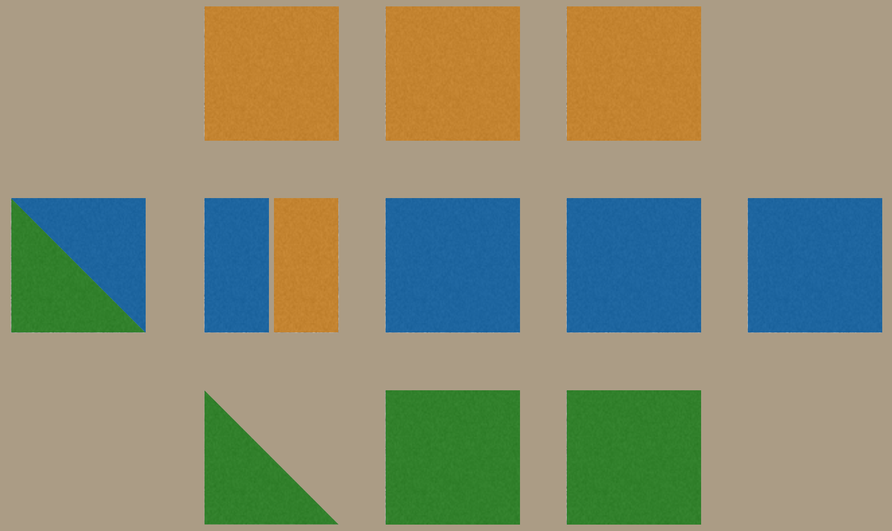
www.blurb.com/books/2783461-portlandia

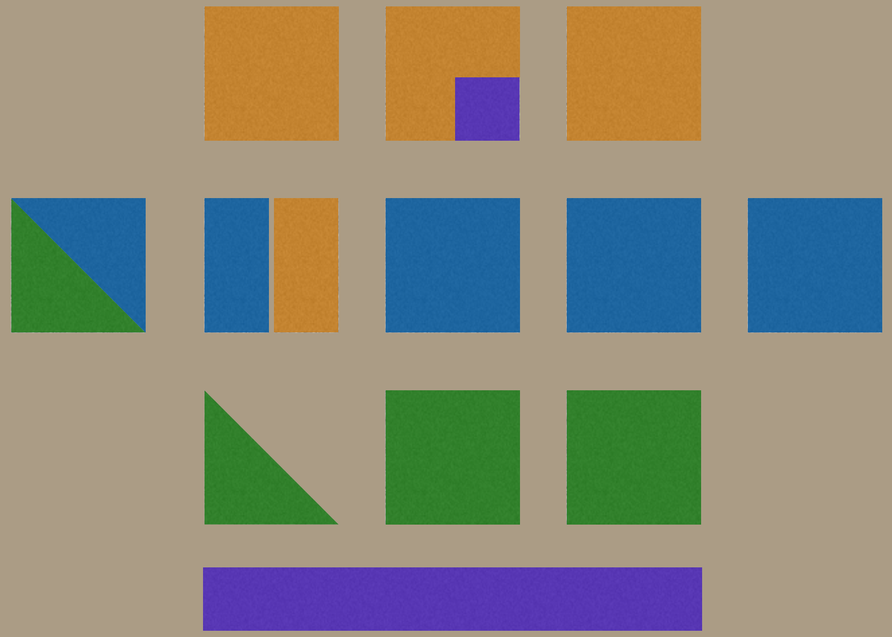
create, publish, sell

create, publish, sell
books & ebooks

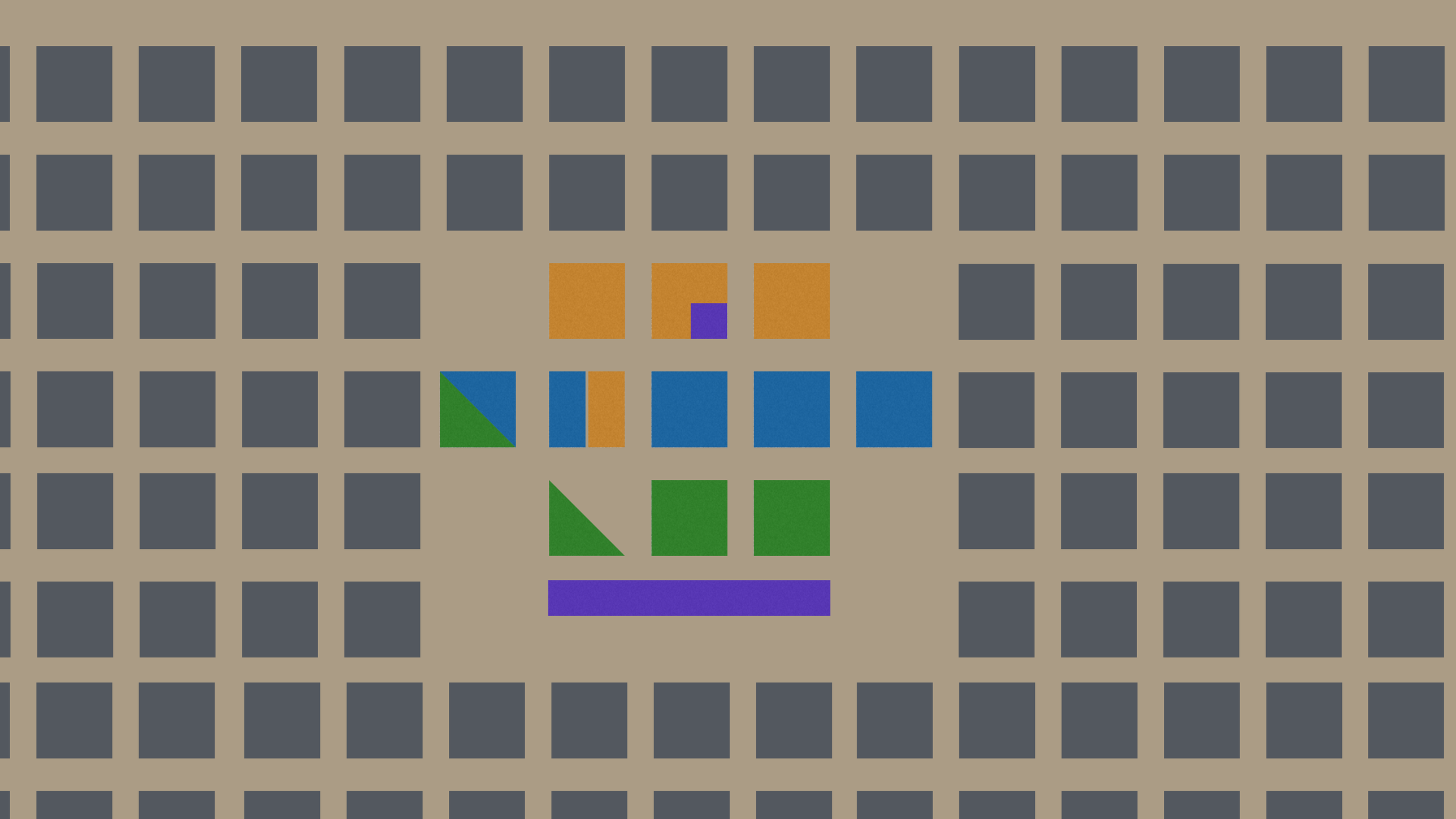
There was a Rails app







Over 8 years



@#%!

Regressions

Can’t ship

“I quit”

Localized refactoring can only get you so far

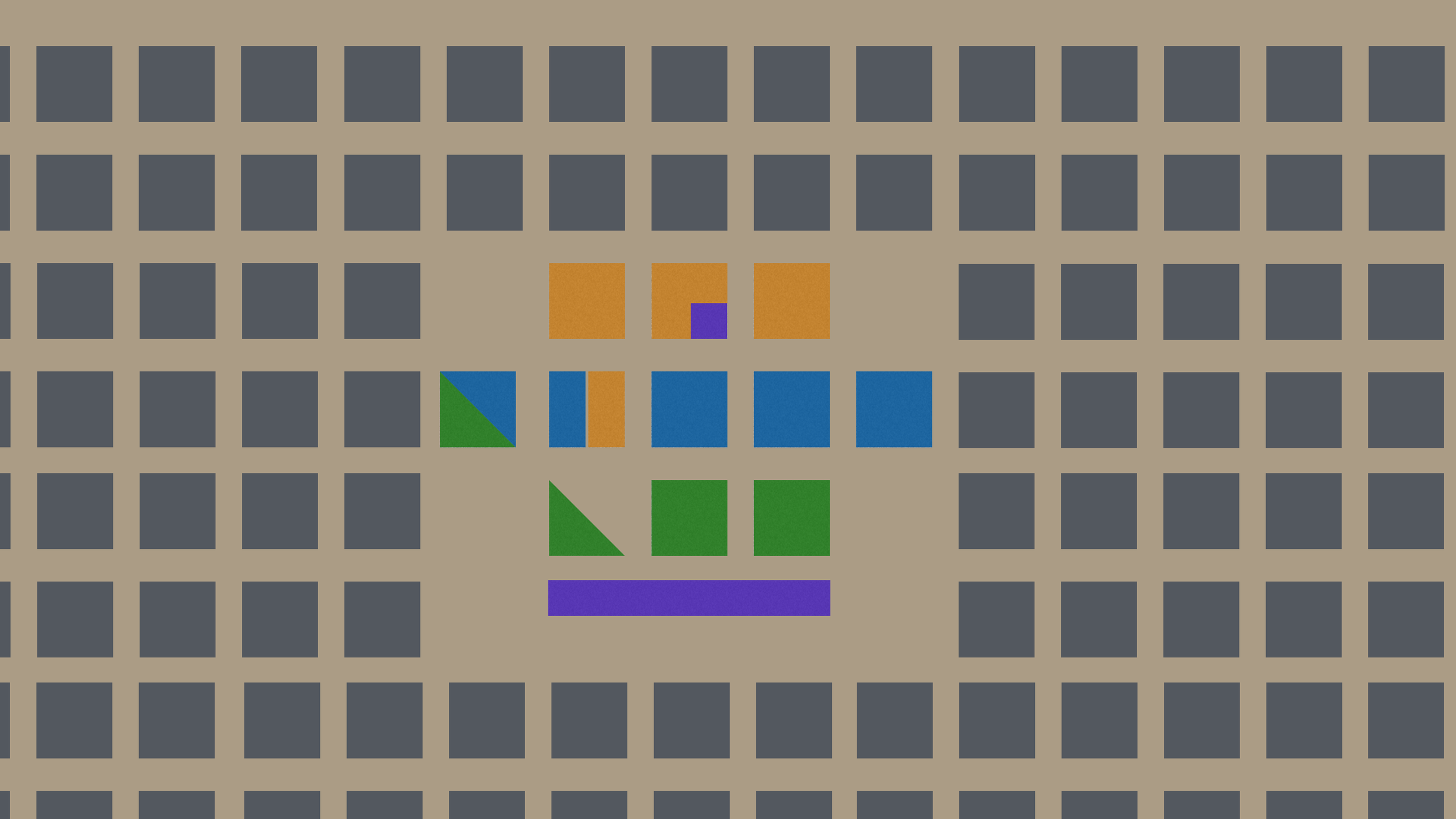
MVC, the Rails Way

ActiveRecord

Presenters, Decorators

Extract Method Object

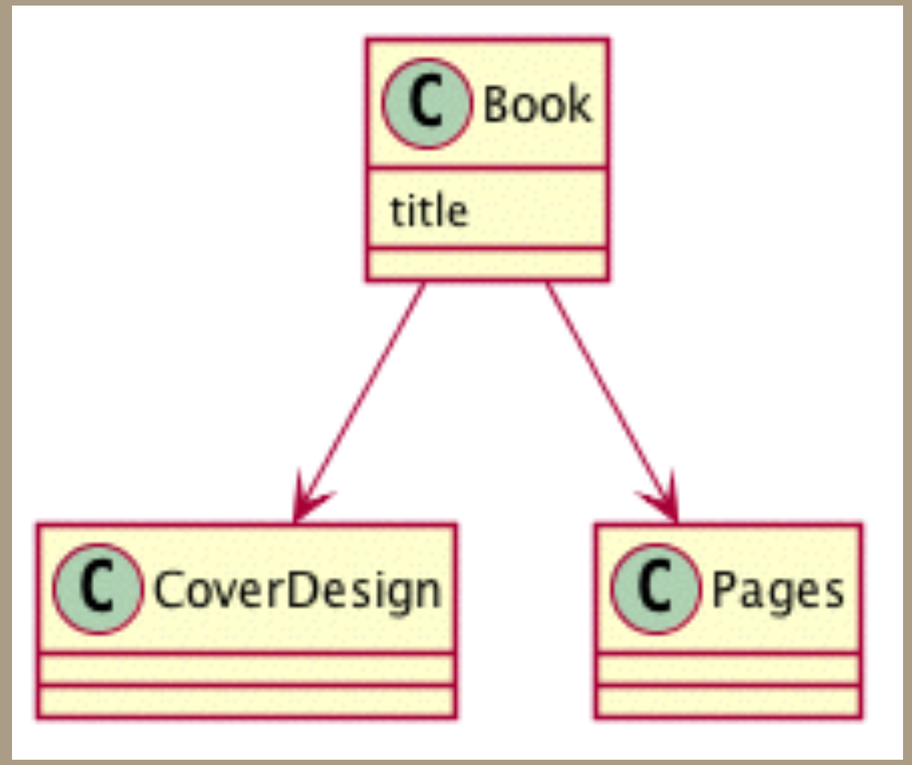
Service Objects



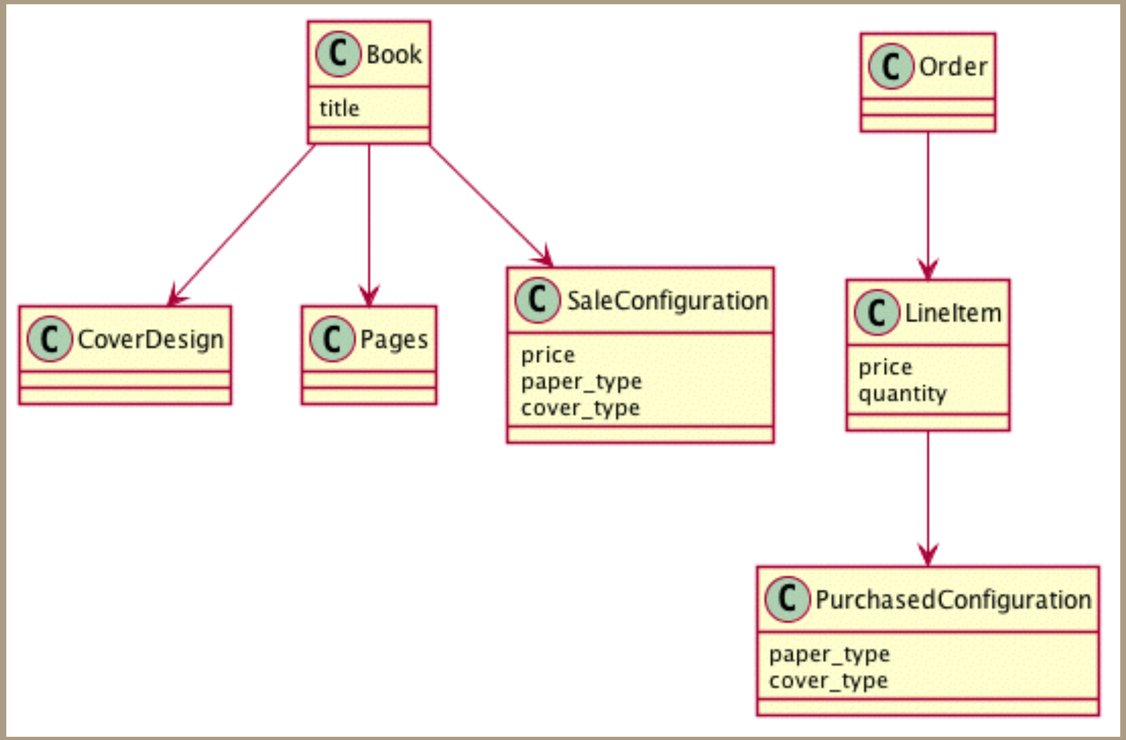
There are deeper insights to be had

Let’s get down and dirty

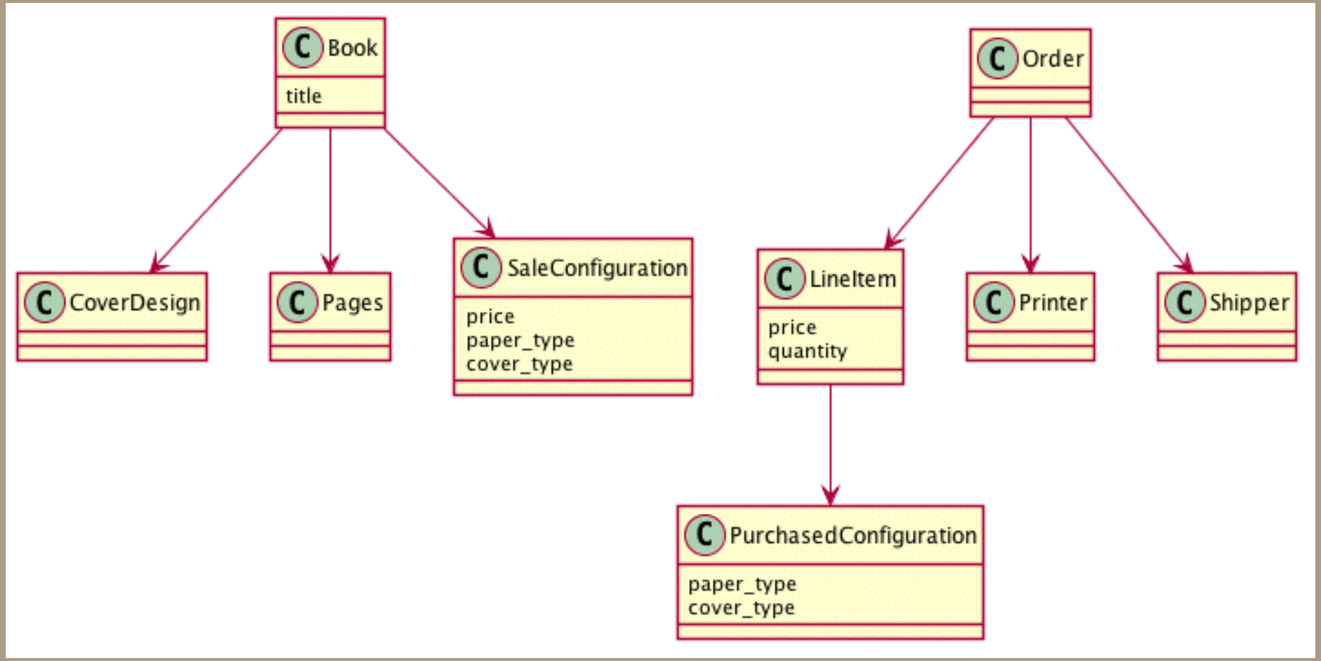
We make books



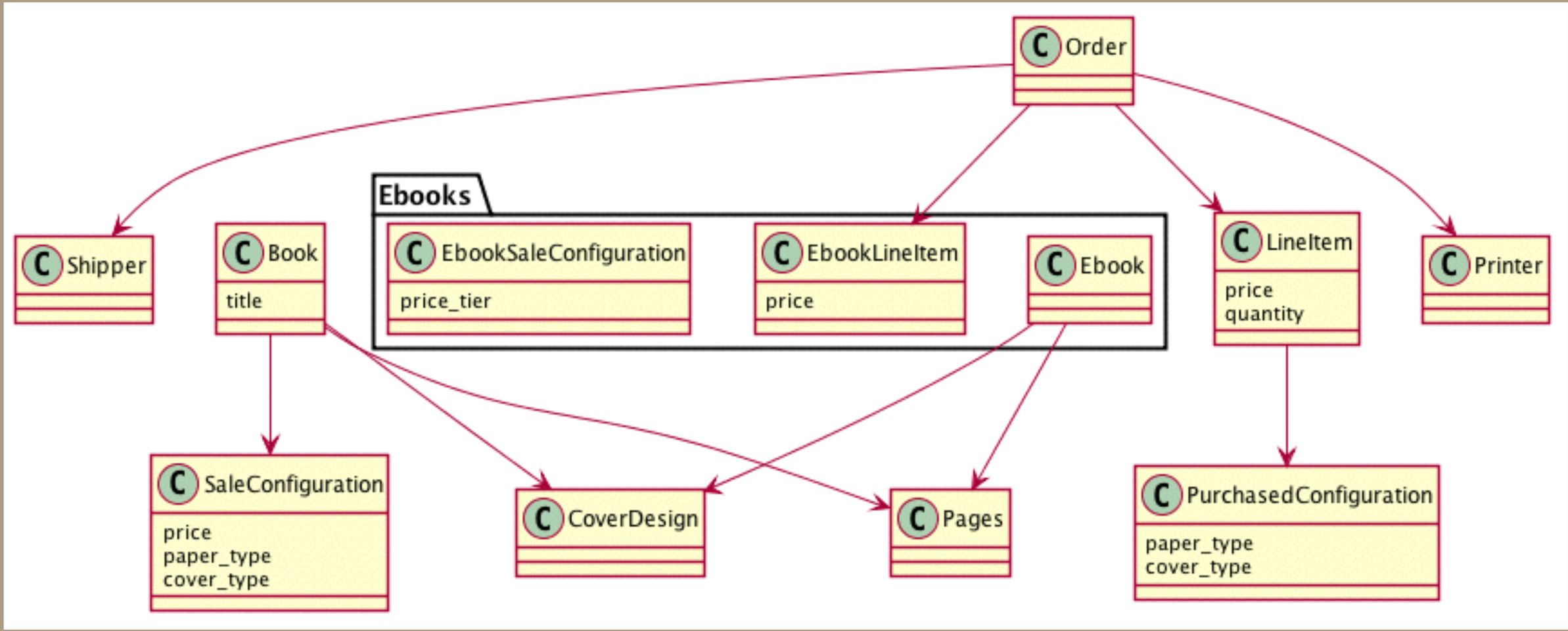
And we sell them



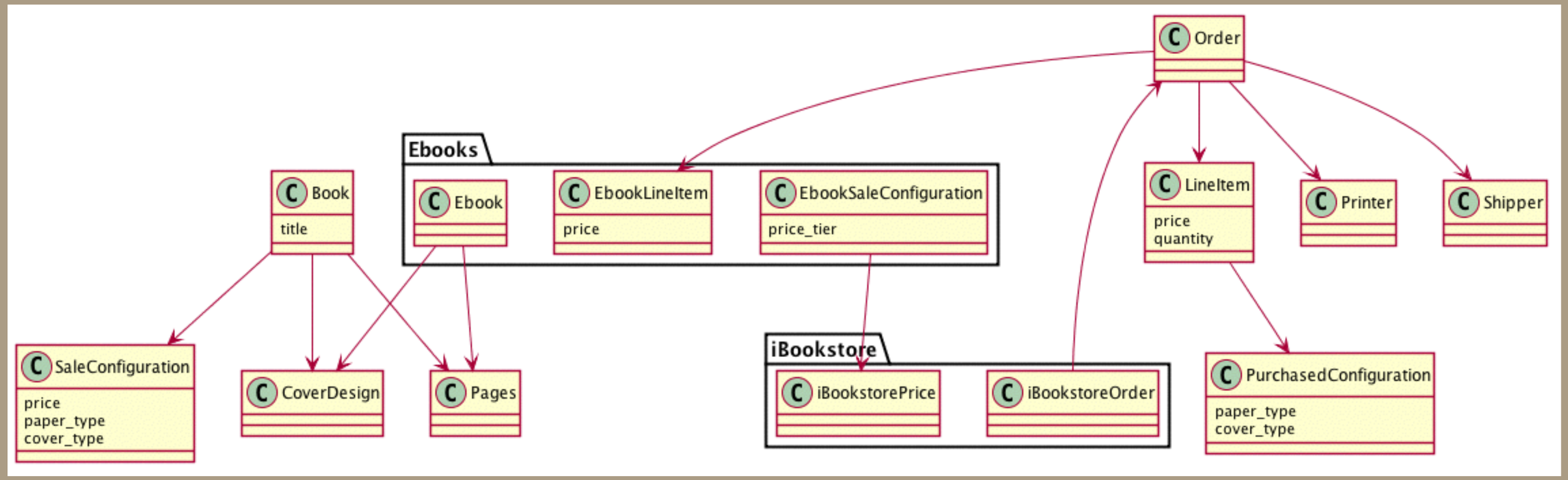
We also print and ship them



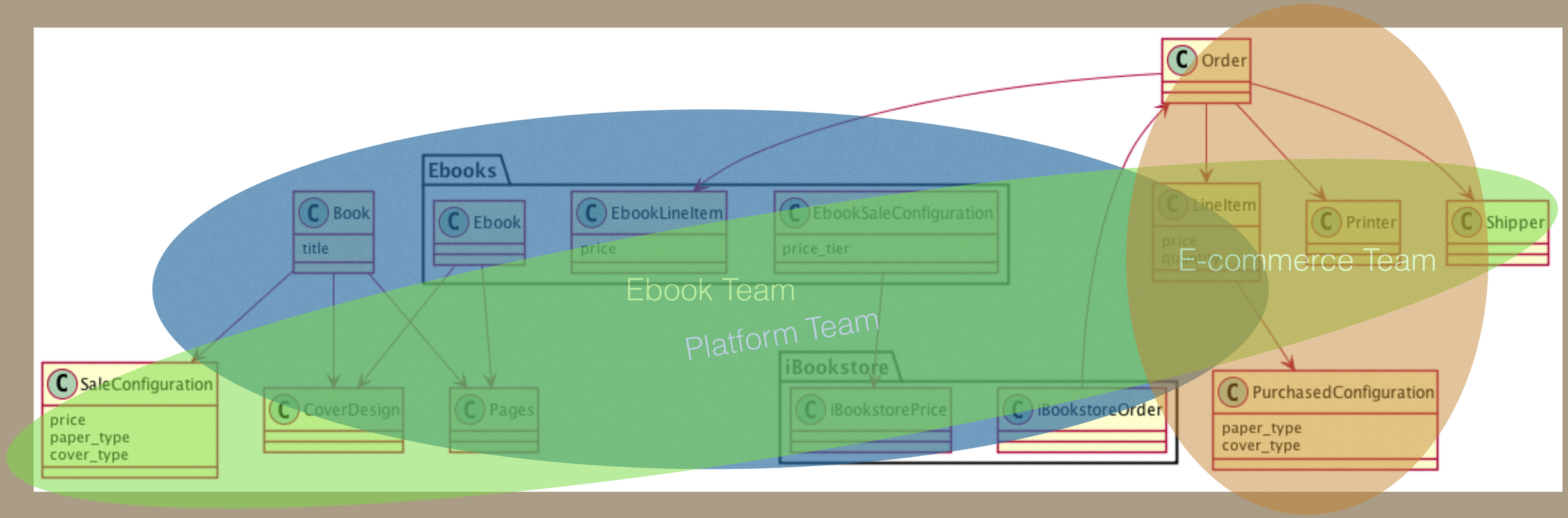
Don’t forget ebooks



...now sell it on the iBookstore



Different teams touch different parts.

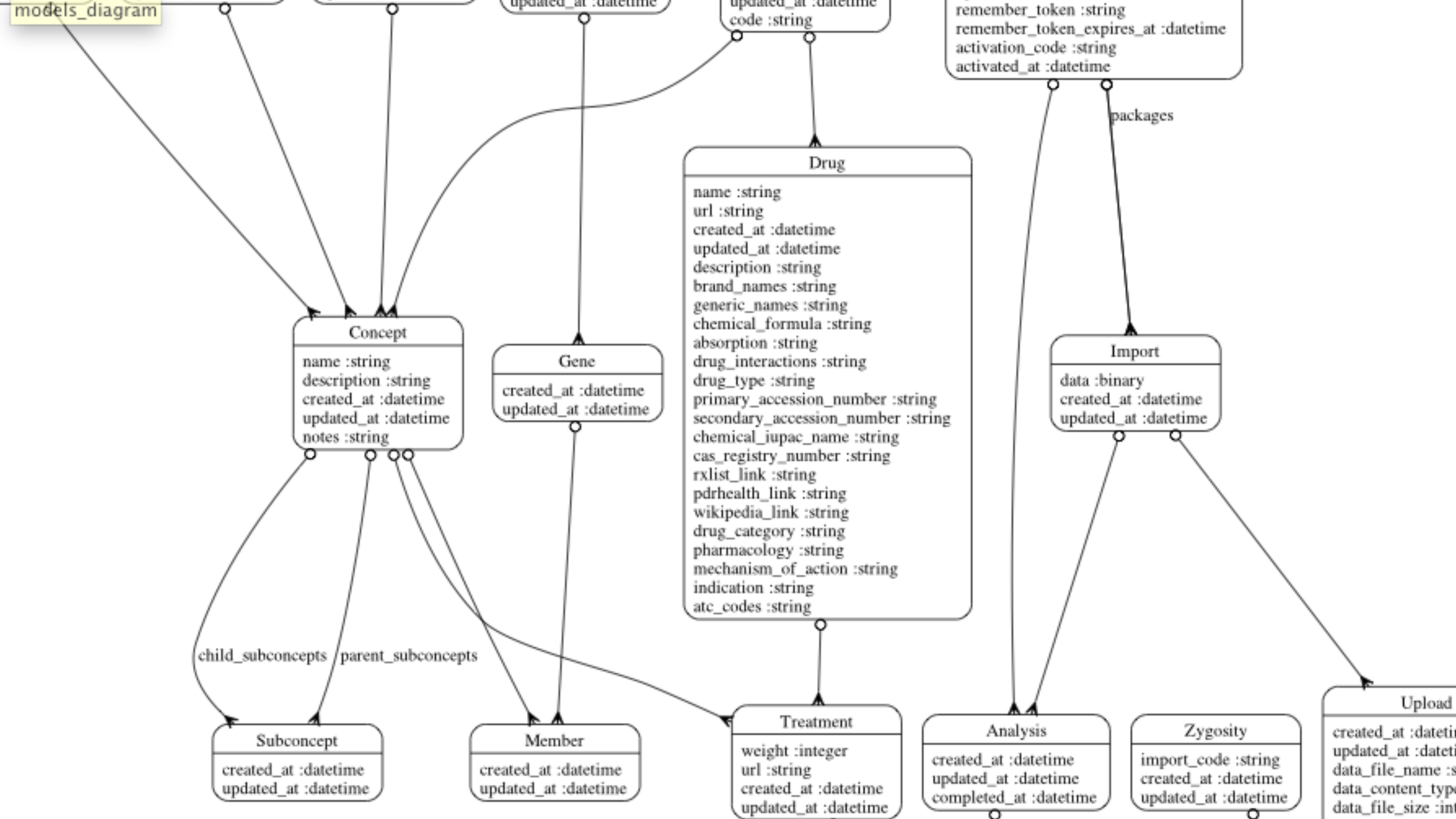


Argh!

Step 1: Visualize

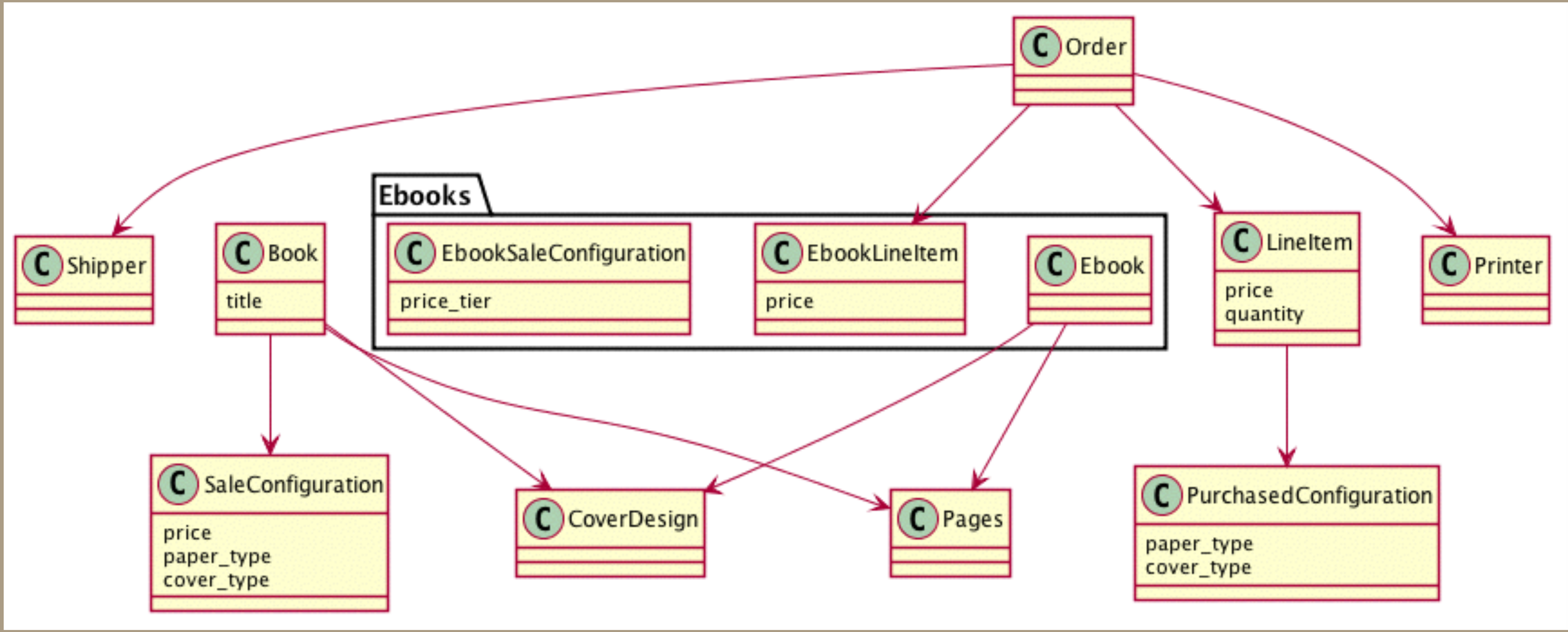
Railroady

Ruby gem to generate UML diagrams from your ActiveRecord models


models_diagram
updated_at :datetime
code :string
remember_token :string
remember_token_expires_at :datetime
activation_code :string
activated_at :datetime
packages
Drug
name :string
url :string
created_at :datetime
updated_at :datetime
description :string
brand_names :string
generic_names :string
chemical_formula :string
absorption :string
drug_interactions :string
drug_type :string
primary_accession_number :string
secondary_accession_number :string
chemical_iupac_name :string
cas_registry_number :string
rxlist_link :string
pdrhealth_link :string
wikipedia_link :string
drug_category :string
pharmacology :string
mechanism_of_action :string
indication :string
atc_codes :string
Concept
name :string
description :string
created_at :datetime
updated_at :datetime
notes :string
Gene
created_at :datetime
updated_at :datetime
Import
data :binary
created_at :datetime
updated_at :datetime
child_subconcepts
parent_subconcepts
Subconcept
created_at :datetime
updated_at :datetime
Member
created_at :datetime
updated_at :datetime
Treatment
weight :integer
url :string
created_at :datetime
updated_at :datetime
Analysis
created_at :datetime
updated_at :datetime
completed_at :datetime
Zygosity
import_code :string
created_at :datetime
updated_at :datetime
Upload
created_at :datetim
updated_at :datetim
data_file_name :s
data_content_typ
data_file_size :int

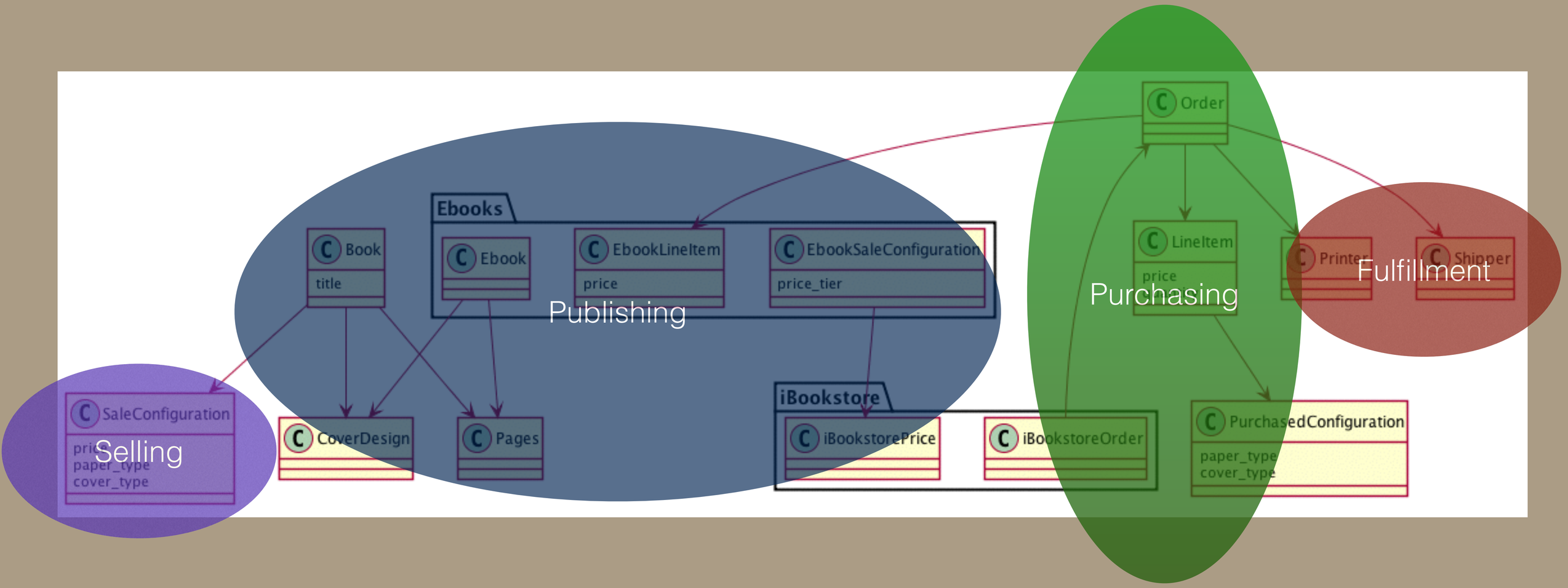
This helps you get the entire system into your mind.

Now we've got UML.

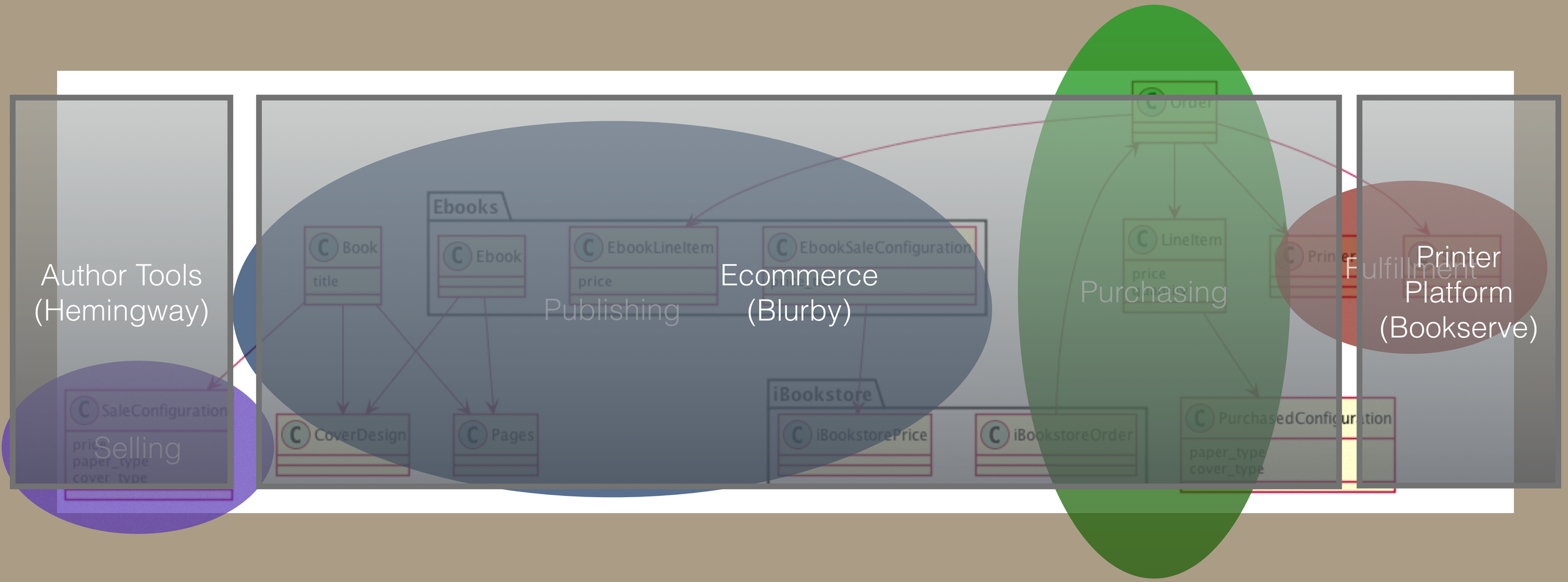


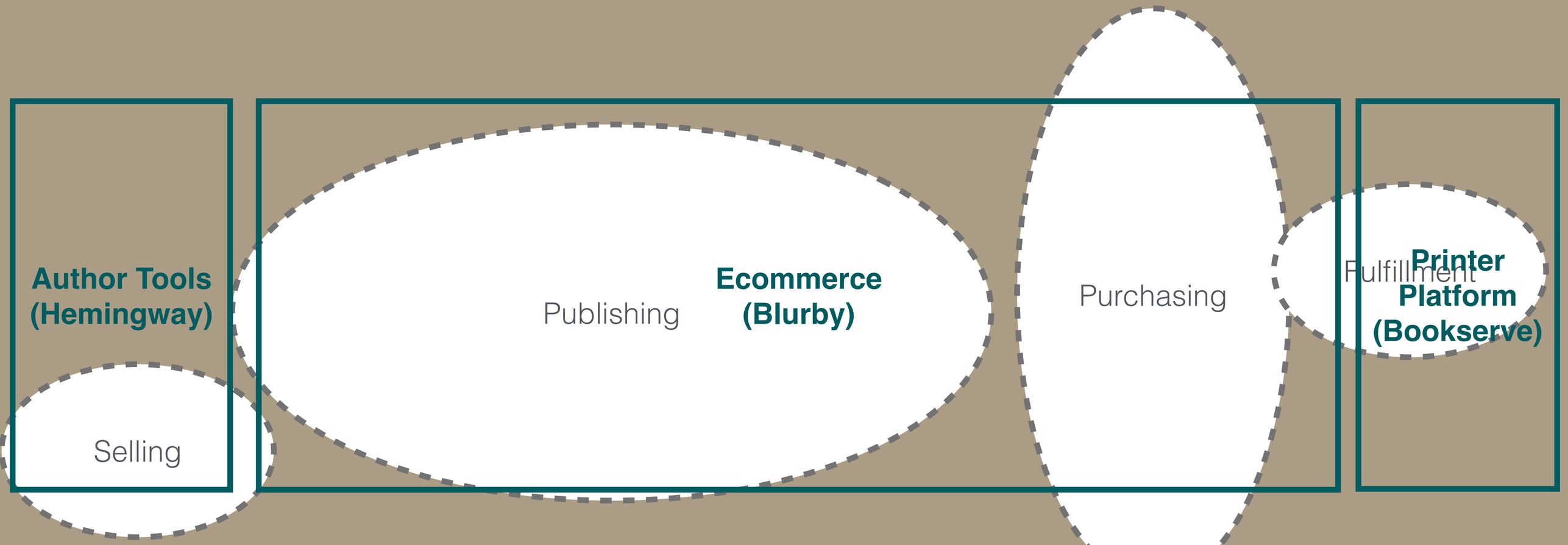
Domain Mapping

Domain: A set of related features that accomplish something for the business.

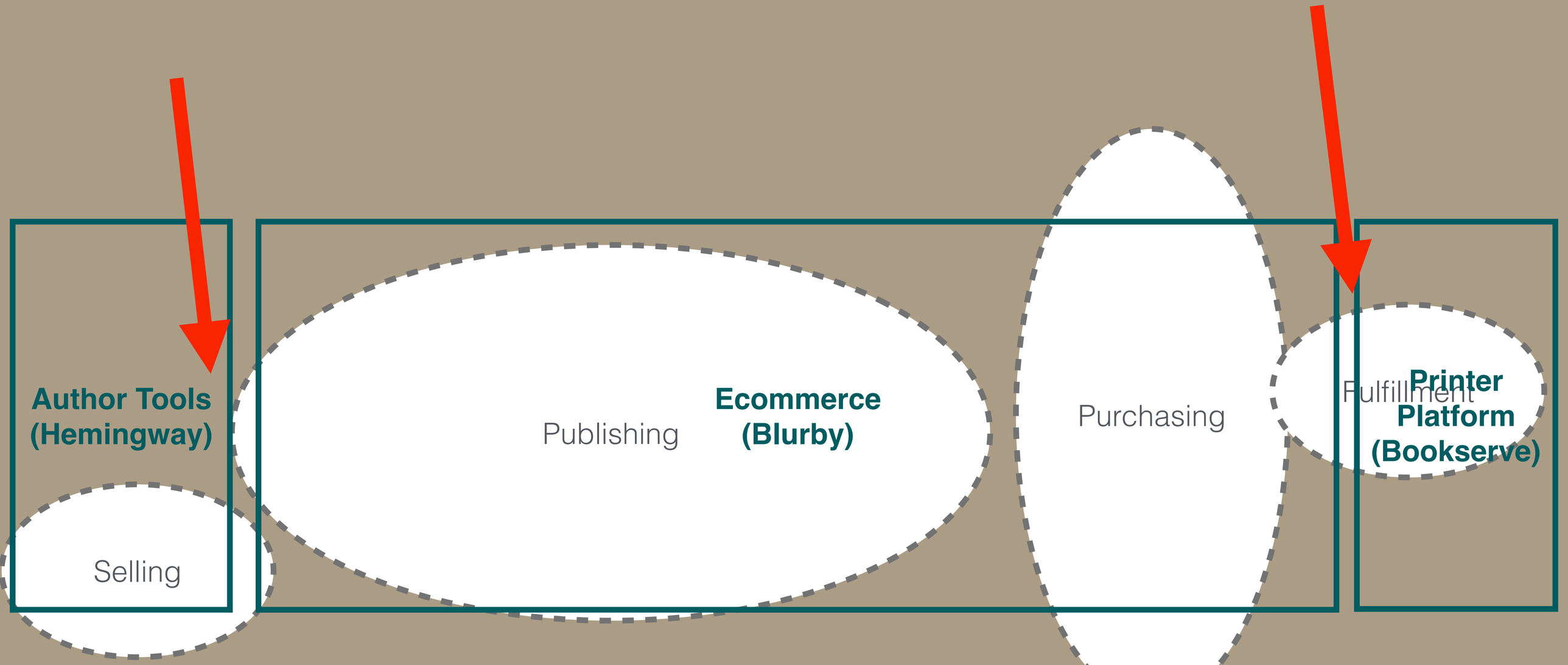

Order
Ebooks
Book
title
Ebook
EbookLineItem
price
EbookSaleConfiguration
price_tier
Publishing
LineItem
price
Purchasing
Printer
Shipper
Fulfillment
SaleConfiguration
paper_type
cover_type
Selling
CoverDesign
Pages
iBookstore
iBookstorePrice
iBookstoreOrder
PurchasedConfiguration
paper_type
cover_type

Bounded Context: A software system that accomplishes some set of features.

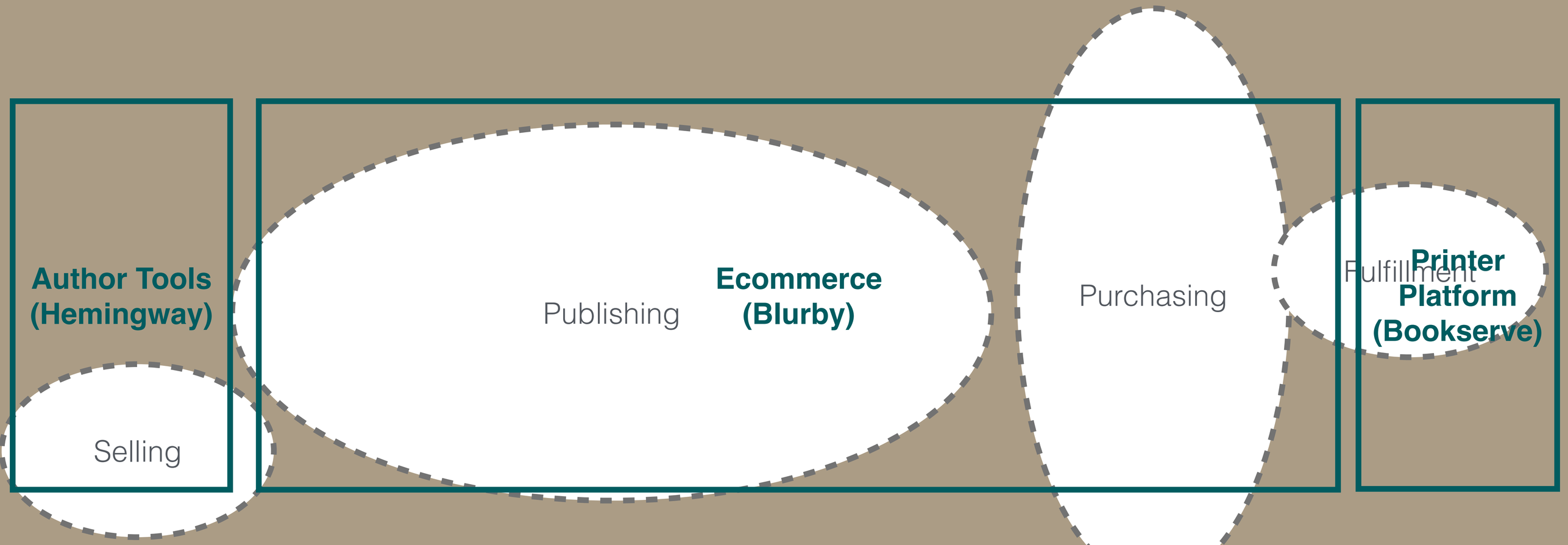

Author Tools
(Hemingway)
Ecommerce
(Blurby)
Printer
Platform
(Bookserve)
Ebooks
Book
title
Ebook
EbookLineItem
price
EbookSaleConfiguration
Publishing
Order
LineItem
price
Purchasing
Fulfillment
SaleConfiguration
paper_type
cover_type
Selling
CoverDesign
Pages
iBookstore
iBookstorePrice
iBookstoreOrder
PurchasedConfiguration
paper_type
cover_type


Author Tools
(Hemingway)
Selling
Publishing
Ecommerce
(Blurby)
Purchasing
Fulfillment
Printer
Platform
(Bookserve)

Ideally, your bounded contexts map 1:1 to your domains.


Author Tools
(Hemingway)
Selling
Publishing
Ecommerce
(Blurby)
Purchasing
Fulfillment
Printer
Platform
(Bookserve)

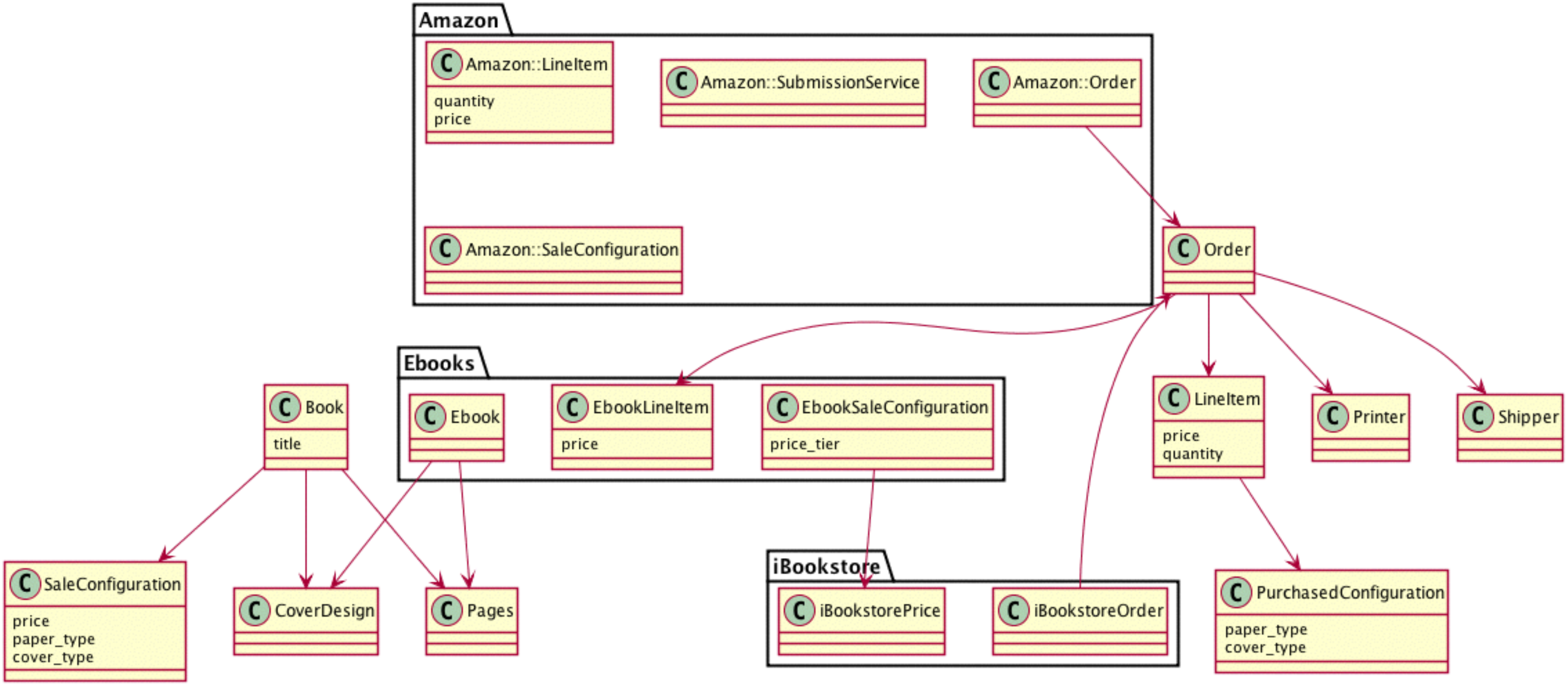
Now add directional dependencies

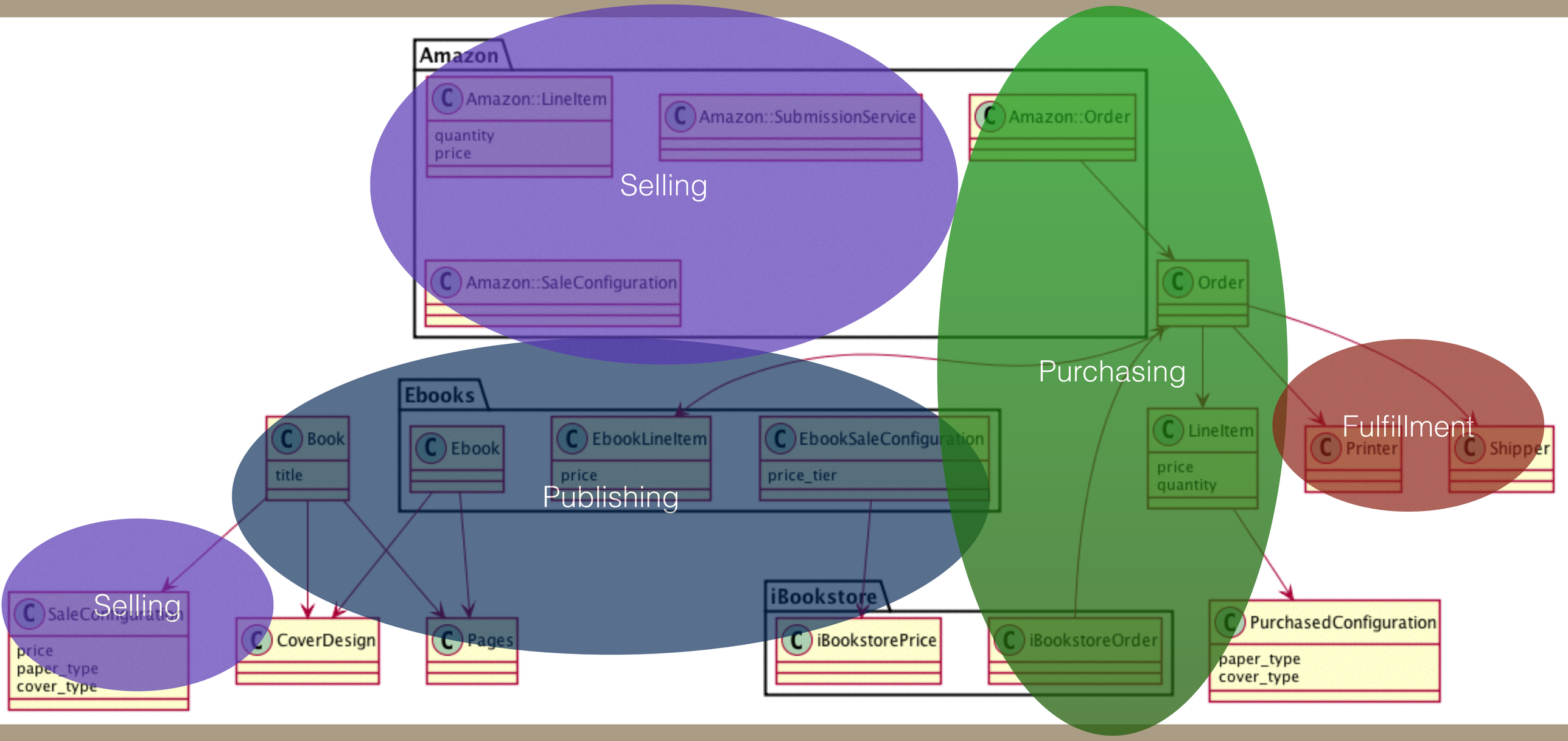

Author Tools
(Hemingway)
Selling
Publishing
Ecommerce
(Blurby)
Purchasing
Fulfillment
Printer
Platform
(Bookserve)

This helps you see dependencies between teams, where communication will be most important.

Step 2: Use a business driver to make a change.

Hey! Let’s sell our books on Amazon!


Amazon
Amazon::LineItem
quantity
price
Amazon::SubmissionService
Amazon::Order
Amazon::SaleConfiguration
Order
Ebooks
Book
title
Ebook
EbookLineItem
price
EbookSaleConfiguration
price_tier
LineItem
price
quantity
Printer
Shipper
SaleConfiguration
price
paper_type
cover_type
CoverDesign
Pages
iBookstore
iBookstorePrice
iBookstoreOrder
PurchasedConfiguration
paper_type
cover_type


Amazon
Amazon::LineItem
quantity
price
Amazon::SubmissionService
Amazon::Order
Selling
Amazon::SaleConfiguration
Order
Purchasing
Ebooks
Book
title
Ebook
EbookLineItem
price
EbookSaleConfiguration
price_tier
Publishing
LineItem
price
quantity
Fulfillment
Printer
Shipper
Selling
SaleConfiguration
price
paper_type
cover_type
CoverDesign
Pages
iBookstore
iBookstorePrice
iBookstoreOrder
PurchasedConfiguration
paper_type
cover_type

Slim down your core domain objects

SaleConfiguration
EbookSaleConfiguration
AmazonSaleConfiguration

SaleConfiguration

EbookSaleAttributes

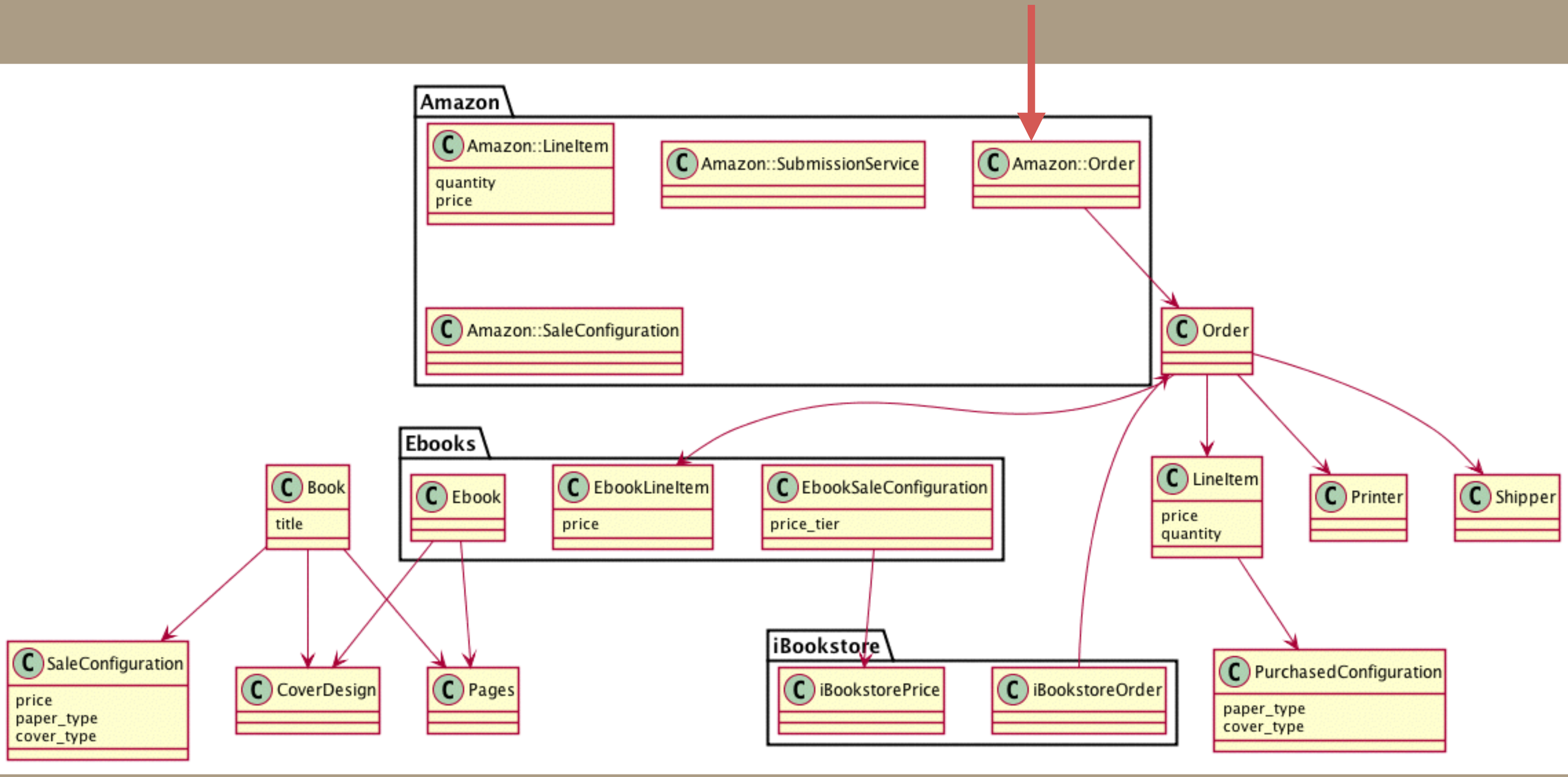
AmazonSaleAttributes

BlurbSaleAttributes

Enforce Domain Purity

No joins between domains

Use aggregate roots to enforce a single entry point

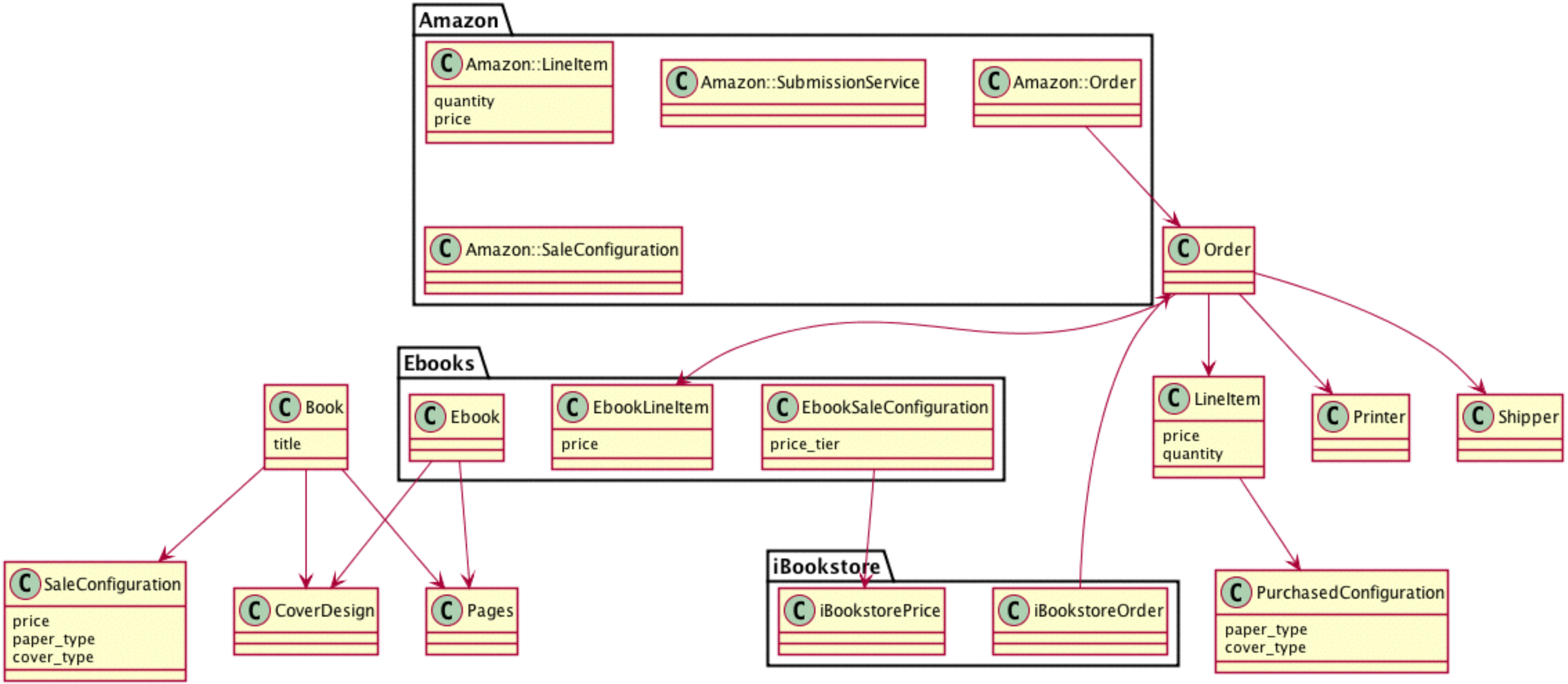

Amazon
Amazon::LineItem
quantity
price
Amazon::SubmissionService
Amazon::Order
Amazon::SaleConfiguration
Order
Ebooks
Book
title
Ebook
EbookLineItem
price
EbookSaleConfiguration
price_tier
LineItem
price
quantity
Printer
Shipper
SaleConfiguration
price
paper_type
cover_type
CoverDesign
Pages
iBookstore
iBookstorePrice
iBookstoreOrder
PurchasedConfiguration
paper_type
cover_type

Namespace the Domain

app/services/amazon/...

Amazon::Foo

Use a domain events to communicate between contexts


Amazon
Amazon::LineItem
quantity
price
Amazon::SubmissionService
Amazon::Order
Amazon::SaleConfiguration
Order
Ebooks
Book
title
Ebook
EbookLineItem
price
EbookSaleConfiguration
price_tier
LineItem
price
quantity
Printer
Shipper
SaleConfiguration
price
paper_type
cover_type
CoverDesign
Pages
iBookstore
iBookstorePrice
iBookstoreOrder
PurchasedConfiguration
paper_type
cover_type

You can do this all in small steps!

Step 3: Toward Maturity

Toward SOA

Namespaced Classes -> Rails Engine -> Separate Rails Service

Introduce a pub/sub mechanism

Shared Kernel - models in a gem

So what happened?

Launched new features

Iterative refactoring - still shipping features

Domain insights

Longer-term SOA efforts

Thanks!