Successfully Decomposing Your Monolith (or `UpdateUser();` Means Nothing To Me)

Sean Farmar

Particular Software

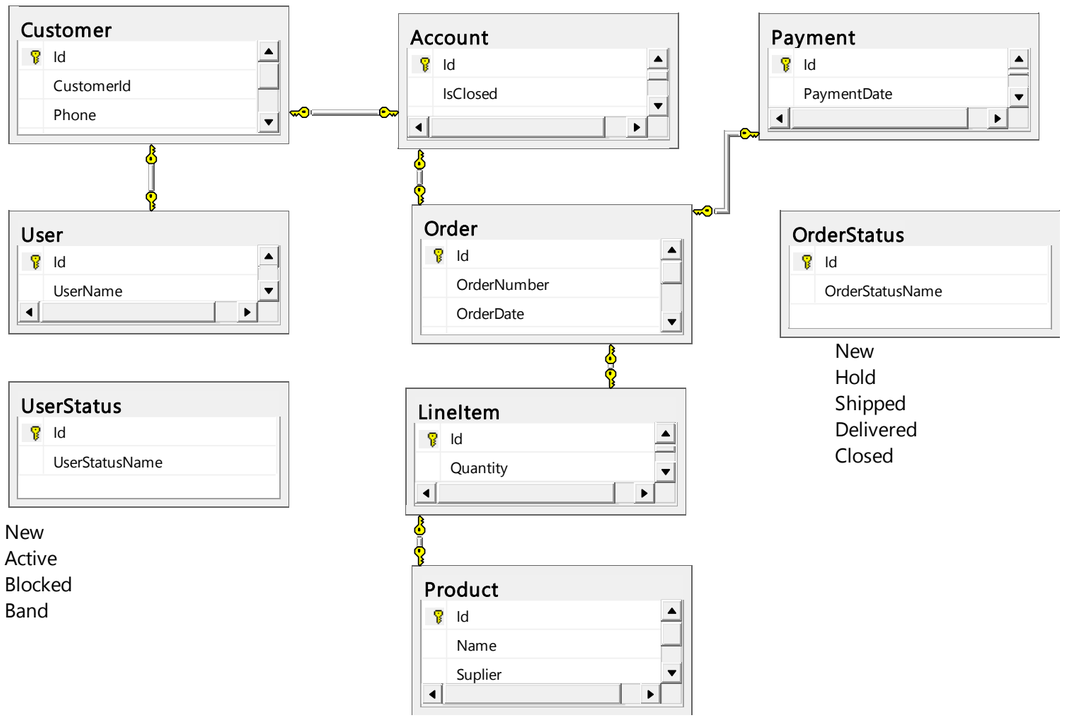
Monolith Design

Analyse the domain

Build a data model that describes the domain

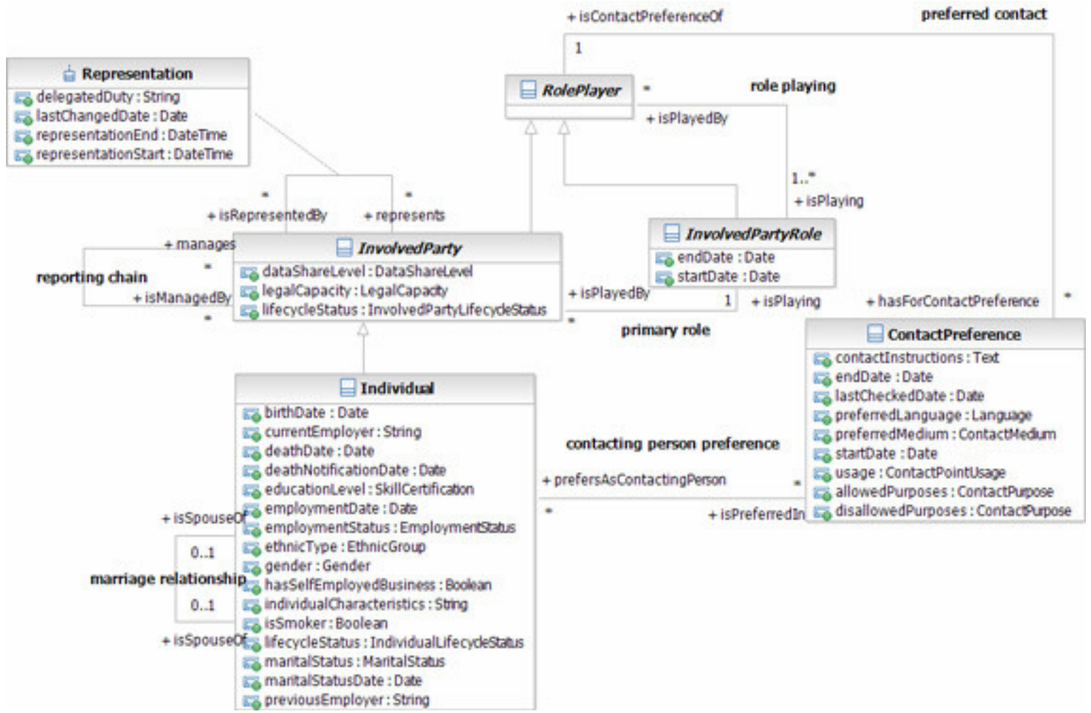
Add relationships and dependencies

Something like this:



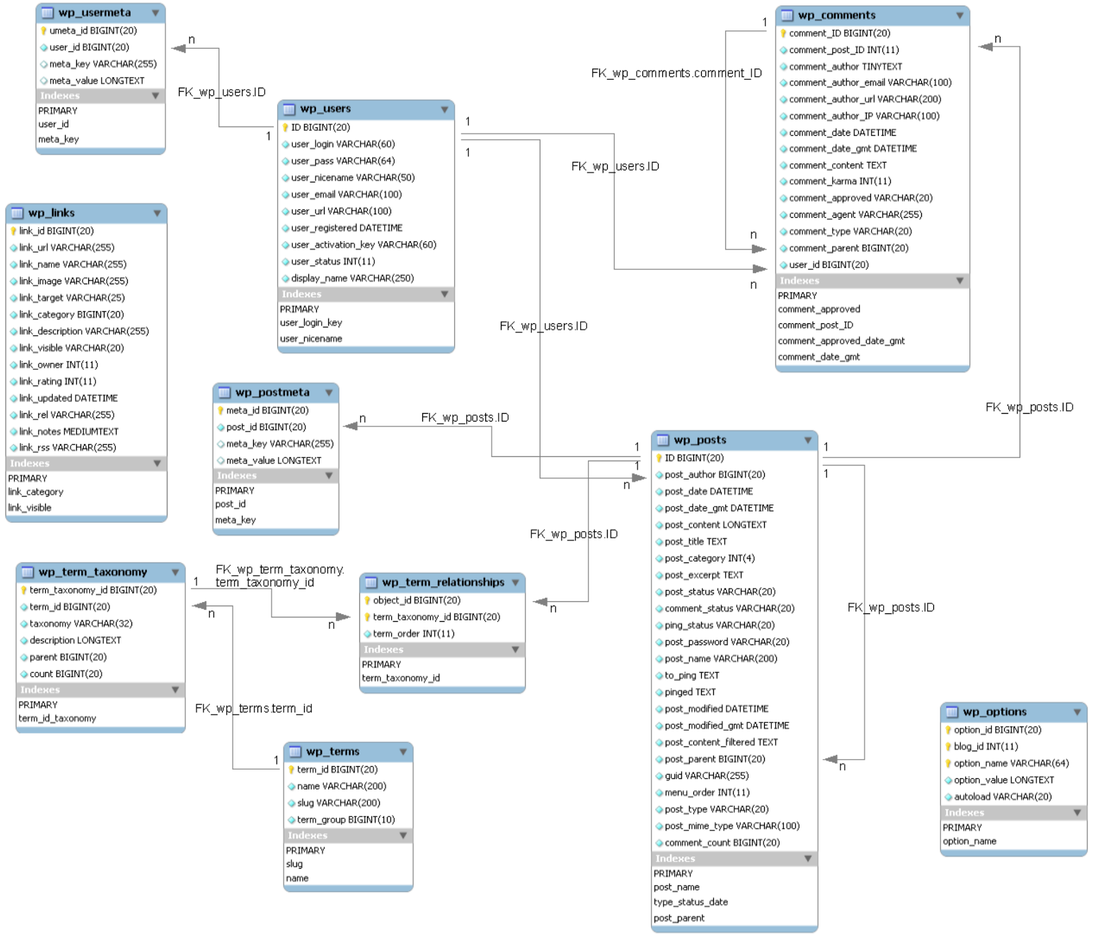
Monolith Design

Or this:



Monolith Design

Or this:



Monolith Design

Add a data layer

Add the Business Logic Layer

Build the UI

… don’t forget the kitchen sink

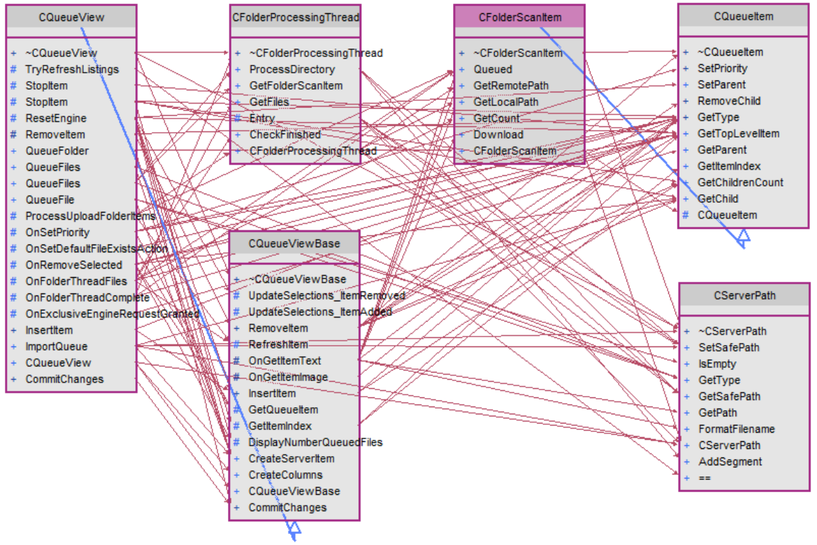
Monolith Design

Can we validate the email on registration

And then send a welcome email?…

Oh, and add some integration with…

And soon enough we end up with a big ball of mud…



So, do we hack it??

Refactor??

Redesign??

Rewrite??

Do not re-write

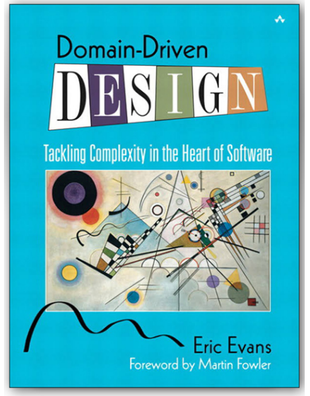
Decompose…

DDD

Domain Driven Design

"Every software program relates to some activity or interest of its user."

Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software



"DDD deals with large models by dividing
them into different Bounded Contexts
and being explicit about their interrelationships."

Martin Fowler

Context

"The setting in which a work or a statement appears that determines its meaning"

Eric Evans

Context

```
UpdateUser(User user);
```

A Step in a workflow

An Item in a CMS

Boundary

Inside: Autonomy, Ownership, Context

Outside: No direct impact, no side effects, ready for change

Bounded Context

"The condition under which a particular model is defined and applicable"

Eric Evans

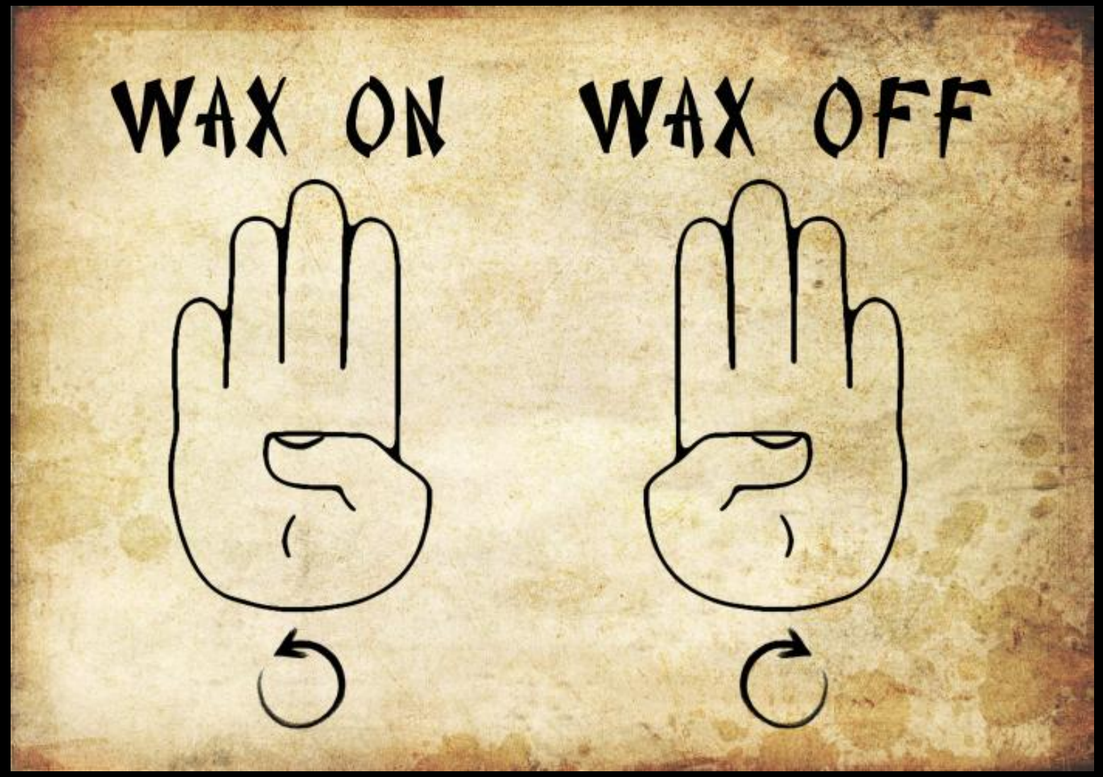
Bounded Context

Where Context and Boundary make it
a distinct business process

Be explicit, Keep it clean

Keep the boundaries

Encapsulate



Find Units Of Work

As a user I can authenticate [UserAuthentication?]:

[UserName] (Could be email?)

[Password]

Does Authorization (access control) belong here as well?

CustomerContactDetails:

[CustomerId]

[ContactEmail] (is that the logging email?)

[CustomerPhoneNumberId] (optional?)

[CustomerAddressId] (optional?)

Email and Mobile Validation Process

Find Units Of Work

CustomerAccount:

[UserAuthenticationId]

[AccountName]

[AccountType]

[First Name]?

[Last Name]?

[AccountActive]

PaymentDetails:

Add one or more payment methods

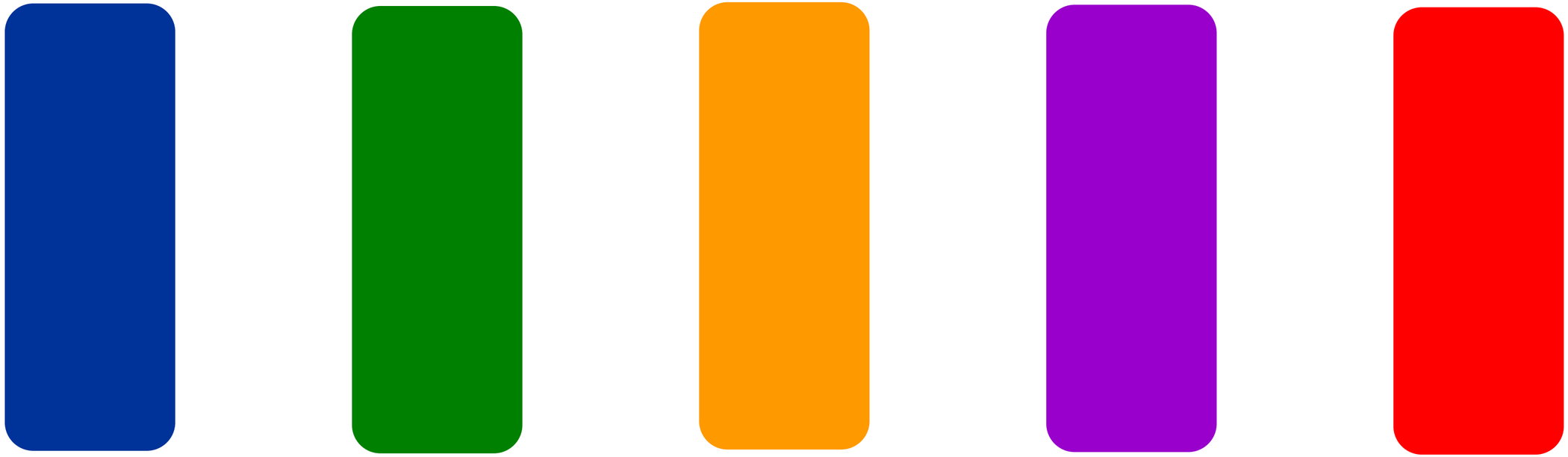
Add Billing Address

Add payment methods process(is it a thing?)

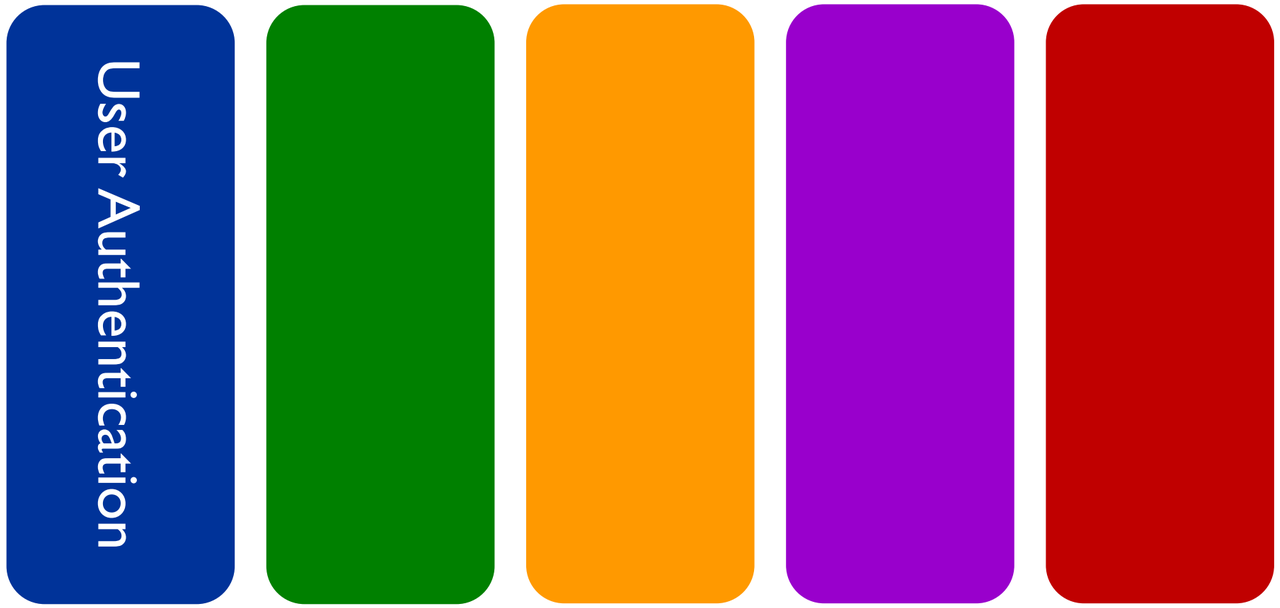
Let's Talk About Process…

It is not going to perfect 😐

DDD and Bounded Context



DDD and Bounded Context



CustomerAccount
- Id
- UserAuthenticationId
- AccountName
- AccountType
- FirstName
- LastName
- AccountActive

UserAuthentication
- Id
- UserName
- Password
- LockedOutDate

CustomerAddress
- Id
- AddressLine1
- AddressLine2
- City
- County
- Country
- Zip

CustomerPhone
- Id
- AreaCode
- PhoneNumber
- PhoneType

CustomerShippingAddress
- Id
- CustomerId
- CustomerAddressId
- Remarks

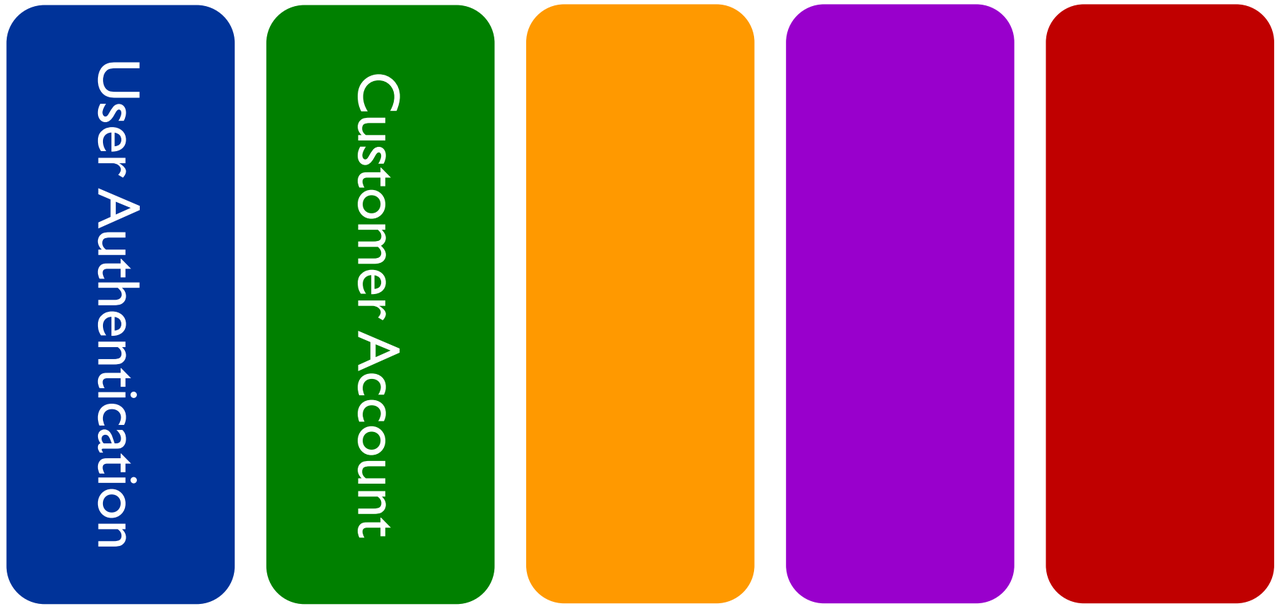
CustomerBillingAddress
- Id
- CustomerId
- CustomerAddressId
- Description

CustomerContactDetails
- Id
- CustomerId
- CustomerPhoneNumberId
- CustomerAddressId
- ContactEmail

CustomerPaymentLogs
- Id
- CustomerId
- TransactionId
- OrderId
- Ammount

CustomerPaymentMethods
- Id
- CustomerId
- PaymentMethodType
- PaymentDetails
- IsActive
- ValidateDate

DDD and Bounded Context



CustomerAccount

Id
UserAuthenticationId
AccountName
AccountType
FirstName
LastName
AccountActive

UserAuthentication

Id
UserName
Password
LockedOutDate

CustomerAddress

Id
AddressLine1
AddressLine2
City
County
Country
Zip

CustomerPhone

Id
AreaCode
PhoneNumber
PhoneType

CustomerShippingAddress

Id
CustomerId
CustomerAddressId
Remarks

CustomerBillingAddress

Id
CustomerId
CustomerAddressId
Description

CustomerContactDetails

Id
CustomerId
CustomerPhoneNumberId
CustomerAddressId
ContactEmail

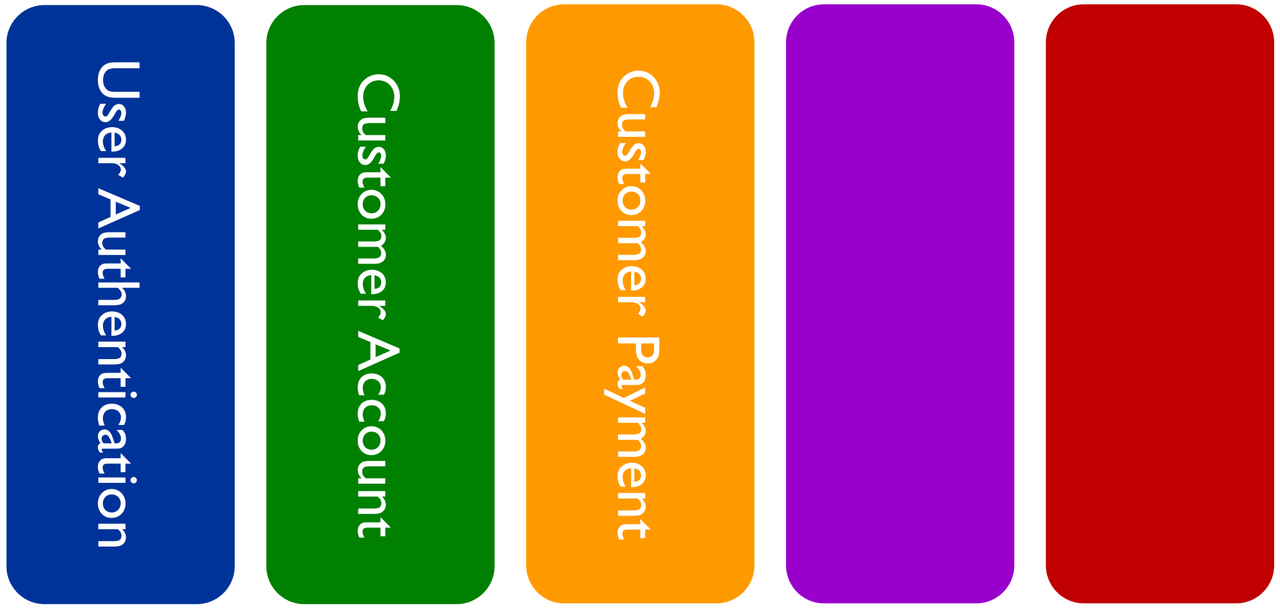
CustomerPaymentLogs

Id
CustomerId
TransactionId
OrderId
Ammount

CustomerPaymentMethods

Id
CustomerId
PaymentMethodType
PaymentDetails
IsActive
ValidateDate

DDD and Bounded Context



CustomerAccount
- Id
- UserAuthenticationId
- AccountName
- AccountType
- FirstName
- LastName
- AccountActive

UserAuthentication
- Id
- UserName
- Password
- LockedOutDate

CustomerAddress
- Id
- AddressLine1
- AddressLine2
- City
- County
- Country
- Zip

CustomerPhone
- Id
- AreaCode
- PhoneNumber
- PhoneType

CustomerShippingAddress
- Id
- CustomerId
- CustomerAddressId
- Remarks

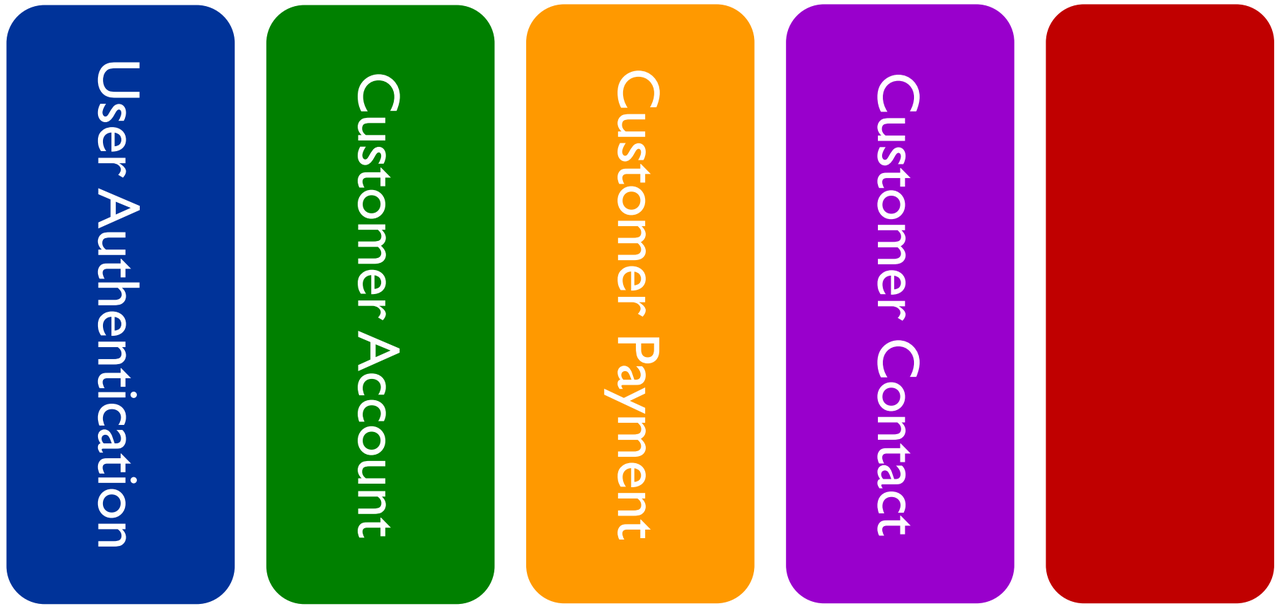
CustomerBillingAddress
- Id
- CustomerId
- CustomerAddressId
- Description

CustomerContactDetails
- Id
- CustomerId
- CustomerPhoneNumberId
- CustomerAddressId
- ContactEmail

CustomerPaymentLogs
- Id
- CustomerId
- TransactionId
- OrderId
- Ammount

CustomerPaymentMethods
- Id
- CustomerId
- PaymentMethodType
- PaymentDetails
- IsActive
- ValidateDate

DDD and Bounded Context



CustomerAccount
- Id
- UserAuthenticationId
- AccountName
- AccountType
- FirstName
- LastName
- AccountActive

UserAuthentication
- Id
- UserName
- Password
- LockedOutDate

CustomerAddress
- Id
- AddressLine1
- AddressLine2
- City
- County
- Country
- Zip

CustomerPhone
- Id
- AreaCode
- PhoneNumber
- PhoneType

CustomerShippingAddress
- Id
- CustomerId
- CustomerAddressId
- Remarks

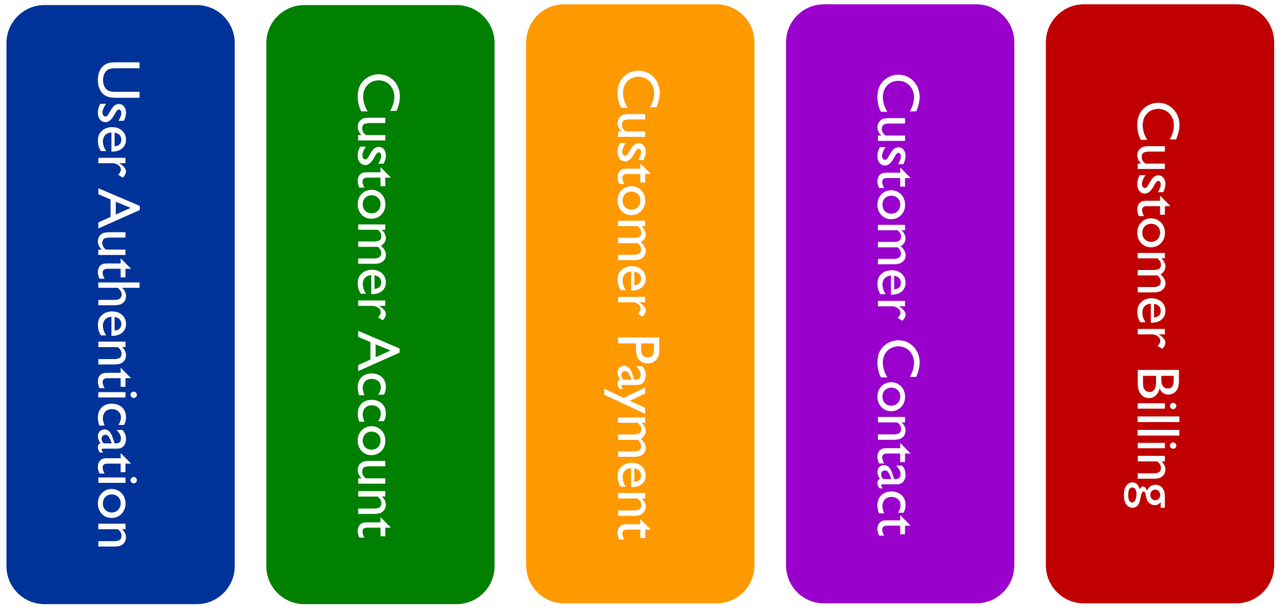
CustomerBillingAddress
- Id
- CustomerId
- CustomerAddressId
- Description

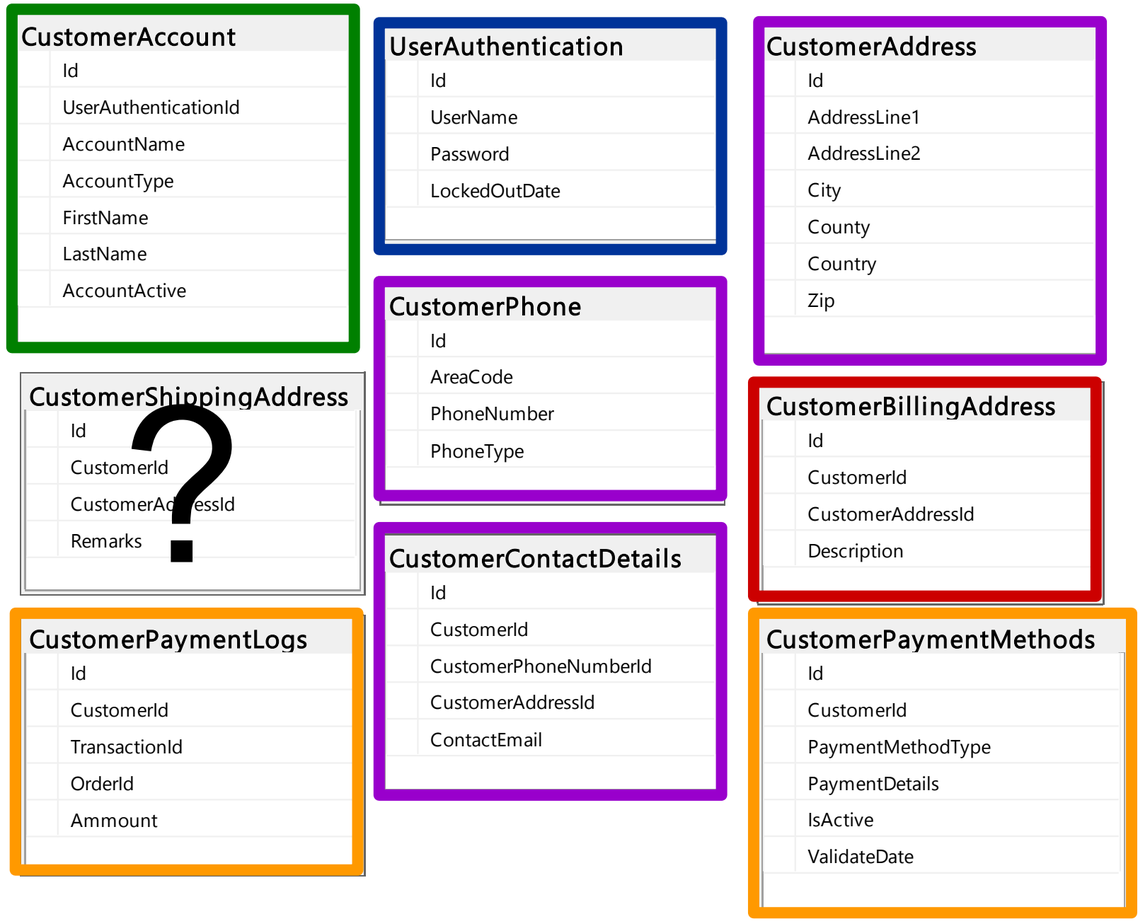
CustomerContactDetails
- Id
- CustomerId
- CustomerPhoneNumberId
- CustomerAddressId
- ContactEmail

CustomerPaymentLogs
- Id
- CustomerId
- TransactionId
- OrderId
- Ammount

CustomerPaymentMethods
- Id
- CustomerId
- PaymentMethodType
- PaymentDetails
- IsActive
- ValidateDate

DDD and Bounded Context





Work Your Way Up

Start in the edge of the system

Use APIs where possible

Include the data in every iteration

Work Your Way Up

Find the bounded Context

Reduce it to the smallest model you can

Find the “Service” Boundaries and it’s processes

Work Your Way Up

Nothing left outside

ITOps – outside service bouderies

Domain Driven Design - Bounded Context

Loose coupling outside the boundary

Tight coupling and cohesion inside the boundary

Respect single data ownership

Composing

UI composition

API composition

Data composition – Read View Models

Summary

Use bounded context to slice your verticals

Include the data in your vertical slicing

Let go of entities, use fields instead

Summary

Keep your boundaries

Reduce coupling between bounded contexts

Use asynchronous communication (for state changes)

Publish Subscribe between bounded contexts

Summary

CQS, SRP, Clean Code

Align teams, repositories, business segments

Don’t forget your stakeholders

It won't be perfect, accept it ☺

One more thing

Get free access to Advanced Distributed Systems Design course recording with Udi Dahan

http://go.particular.net/dotnextru19

Thank You!

Sean Farmar

@farmar

Particular.net

Q&A