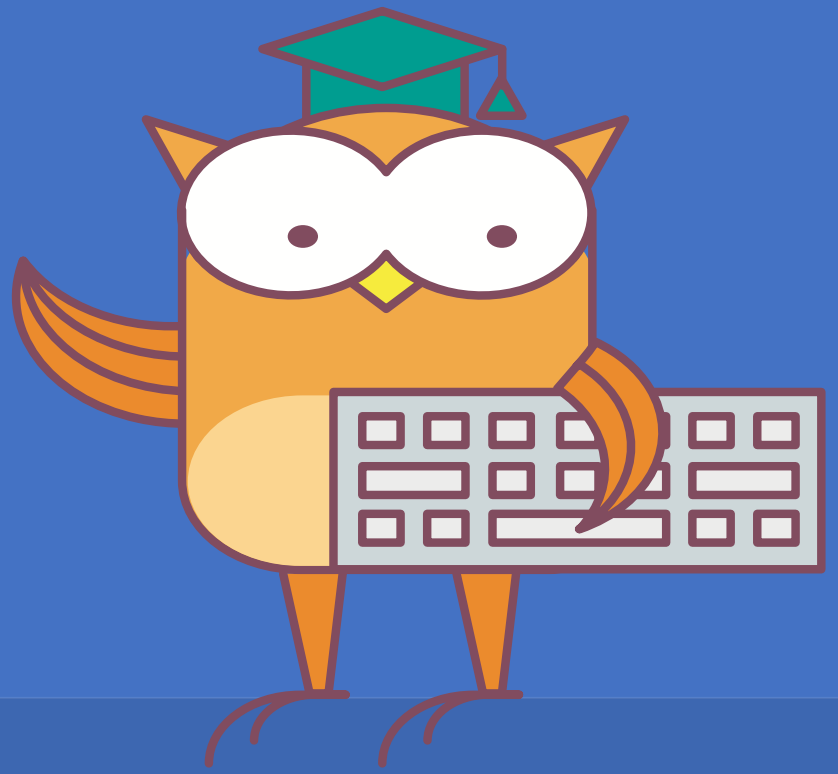
OTUS

Паттерны декомпозиции микросервисов

Архитектор ПО



Щетинников Стас

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

Карта вебинара

- Рассмотрим 2 кейса по декомпозиции сервисов
- На их примере выделим типичные проблемы при декомпозиции сервисов
- Рассмотрим 2 паттерна и 2 антипаттерна

Кейс «Уведомления на почту»

Контекст: есть продукт - стриминговый сервис "Вафликс". Продукт реализован на микросервисной/сервисной архитектуре. Сейчас все сервисы написаны на Java. Сервисов порядка 100, порядка 10 из них надо отсылать уведомления на почту. Для этого используется внешний сервис (MailChimp). Сервисы находятся у разных команд и каждый сервис сам пишет интеграцию с MailChimp.

Задание: оцените архитектурное решение интеграцию каждой системе делать свою? Если для оценки не хватает данных в контексте, можете уточнить детали.

Какие риски вы видите в таком решении?
Опишите сценарии, при которых риски реализуются.
Предложите одно альтернативное архитектурное решение или несколько.

Кейс «Уведомления на почту»

Риск. Изменения логики отправки уведомления потребует изменений во всех сервисах.

- **Сценарий**. Захотели переехать на нового провайдера услуги - придется менять во всех сервисах интеграцию
- **Сценарий**. Захотели для разного типа рассылок иметь разных провайдеров - придется менять во многих сервисах интеграцию
- **Сценарий**. Захотели дублирующего оператора, на случай если один падает - придется менять интеграцию во всех сервисах.

Кейс «Уведомления на почту»

Риски. Отсутствие централизованного управления нагрузкой и распределения запросов на внешний сервис.

Сценарий. Если во всех интеграциях используется один токен к внешнему API, то один сервис может весь лимит выбрать и отвалятся уведомления у всех.

Кейс «Уведомления на почту»

Альтернативные решение: компонентный подход. Логику интеграций хранить в библиотеке.

Какие риски вы видите в таком решении?
Опишите сценарии, при которых риски реализуются.

Кейс «Уведомления на почту»

Альтернативное решение: сервисный подход. Логику интеграции с внешним провайдером хранить в отдельном сервисе.

Какие риски вы видите в таком решении?
Опишите сценарии, при которых риски реализуются.

Кейс «Уведомления на почту»

Контекст: есть продукт - стриминговый сервис "Ива". Продукт планируется реализовать на микросервисной/сервисной архитектуре. Проект только стартует. В наличии только Java разработчики. По бизнес-процессу надо отсылать уведомления на почту, пока только в паре случаев при регистрации. Для этого планируется использовать внешний сервис (MailChimp).

Задание: оцените архитектурное решение интеграцию каждой системе делать свою? Если для оценки не хватает данных в контексте, можете уточнить детали.

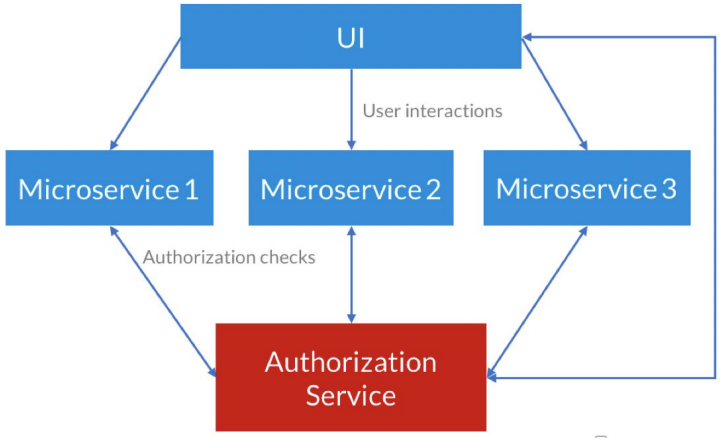
Какие риски вы видите в таком решении?
Опишите сценарии, при которых риски реализуются.
Предложите одно альтернативное архитектурное решение или несколько.

Кейс «Права доступа»

Контекст: есть сервис размещения объявлений "Завито". В сервисе есть логика проверки прав доступа к объявлению внутри системы (также как и с другими объектами). Клиент может заводить объявления, добавлять фотографии и отправлять на модерацию и снимать с показа свои объявления. Модератор может модерировать объявления в рамках своего блока, но не может их менять. Администраторы могут редактировать и модерировать вообще любые объявления.

При проектировании было предложено в рамках микросервисной архитектуры логику проверки прав инкапсулировать в отдельном сервис "Проверка прав". Если пользователь (клиент, модератора, администратор) хочет произвести какое-то действие, то сервис ходит в сервис авторизации и делает пр[illegible]

Задание: оцените архитектурное решение. Если для оценки не хватает данных в контексте, можете уточнить детали. Какие риски вы видите в таком решении?
Опишите сценарии, при которых риски реализуются.



Основные риски при декомпозиции сервисов

- **Риск**. Вероятность согласованных изменений в (большом) количестве сервисов

Сценарий. Для реализации фичи необходимо сделать несколько согласованных правок в разных сервисах.

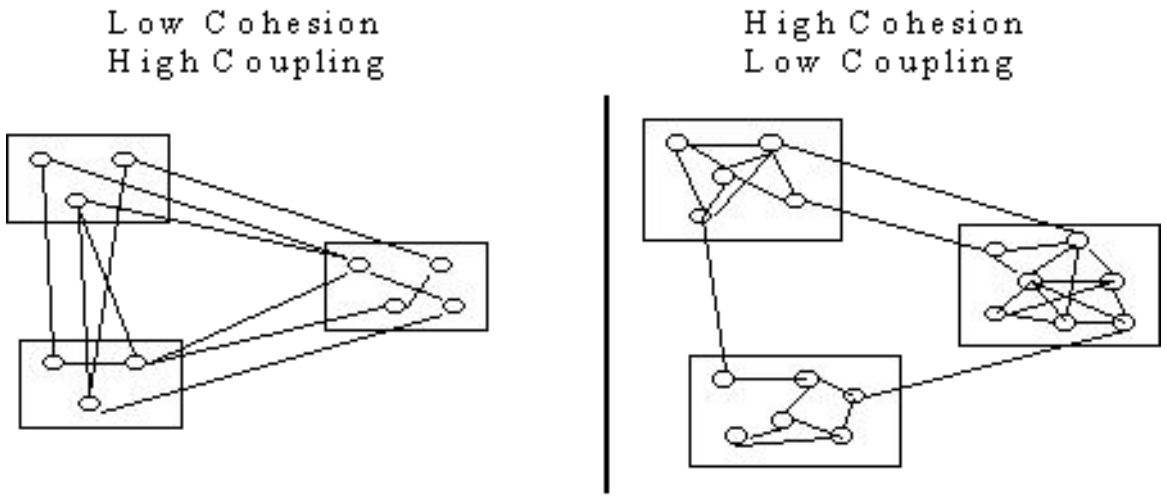
- **Риск**. Излишняя связность с другими сервисами может привести к уменьшению отказоустойчивости

Сценарий. Падение сервиса уводит с собой другие сервисы, т.к. сервис связан со многими другими.

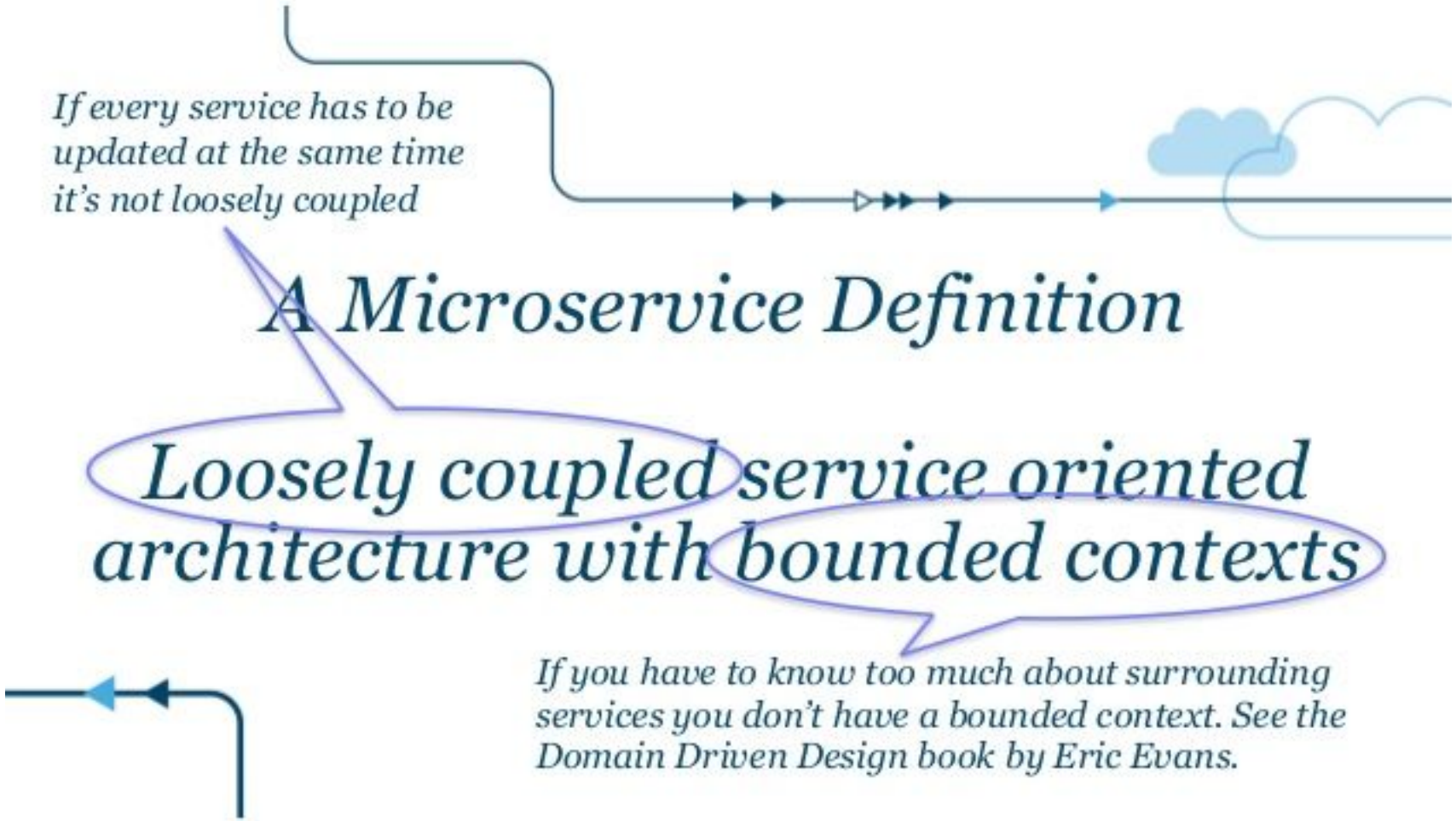
- **Риск**. Излишняя связность с другими сервисами может привести к бОльшей нагрузке на сервисы.

Сценарии. Увеличение нагрузки на несколько сервисов, также увеличивает нагрузку и на связанные с ним.

Loose coupling & high cohesion принцип



- Внутри сервиса находится связанная бизнес-логика
- Сервисы максимально независимы друг от друга


If every service has to be updated at the same time it's not loosely coupled
A Microservice Definition
Loosely coupled service oriented architecture with bounded contexts
If you have to know too much about surrounding services you don't have a bounded context. See the Domain Driven Design book by Eric Evans.

Loose coupling & high cohesion

Loose coupling & high cohesion – это лишь признак правильно спроектированных систем. Но он не говорит, как именно достигать сильной связности и слабой зависимости между модулями и сервисами.

На вопрос, как можно достигнуть слабой зависимости и сильной связности, отвечают паттерны и антипаттерны.

Декомпозиция по предметной области
(decomposition by subdomain)

Какую проблему решает наше ПО? Какова его предметная область?

Поговорить с представителями заказчиков и послушать их.

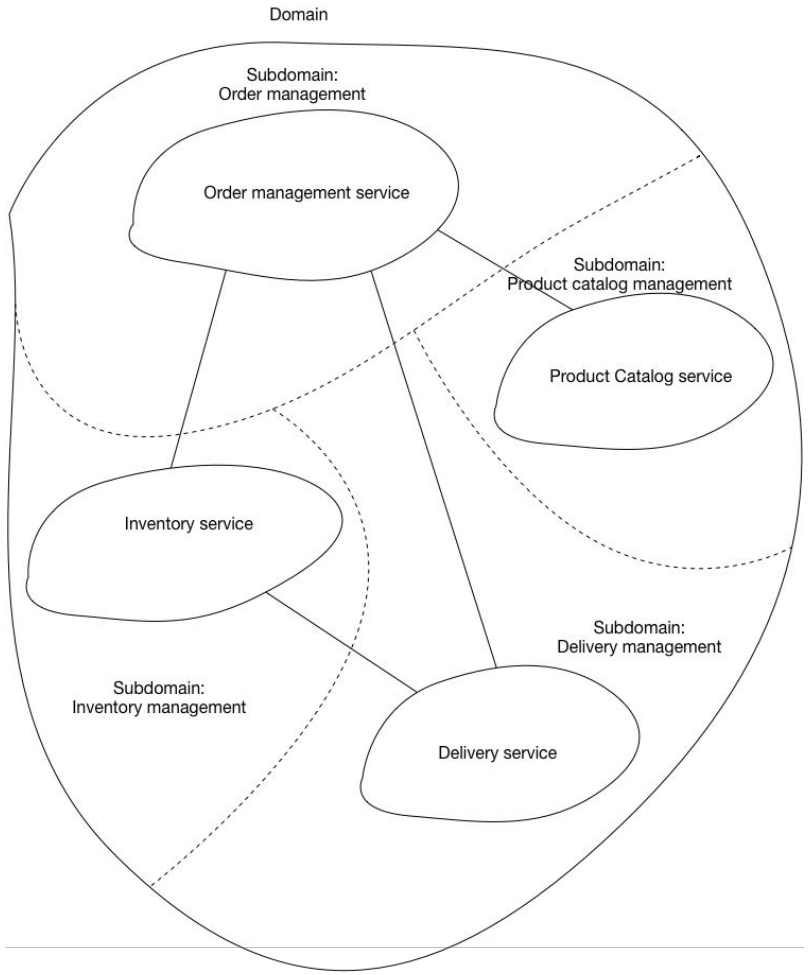
Проблемные области, в рамках которых эксперты предметной области говорят на одном языке, с использованием одного набора понятий и связанные бизнес-логикой между собой, скорее всего являются хорошими для того, чтобы строить вокруг них сервисы. (bounded context DDD)

Декомпозиция по предметной области
(decomposition by subdomain)

Примеры проблемных областей (или ограниченных контекстов)
для интернет-магазина:

- Поиск товара
- Оплата заказа
- Формирование заказа
- Доставка товара

Декомпозиция по предметной области (decomposition by subdomain)

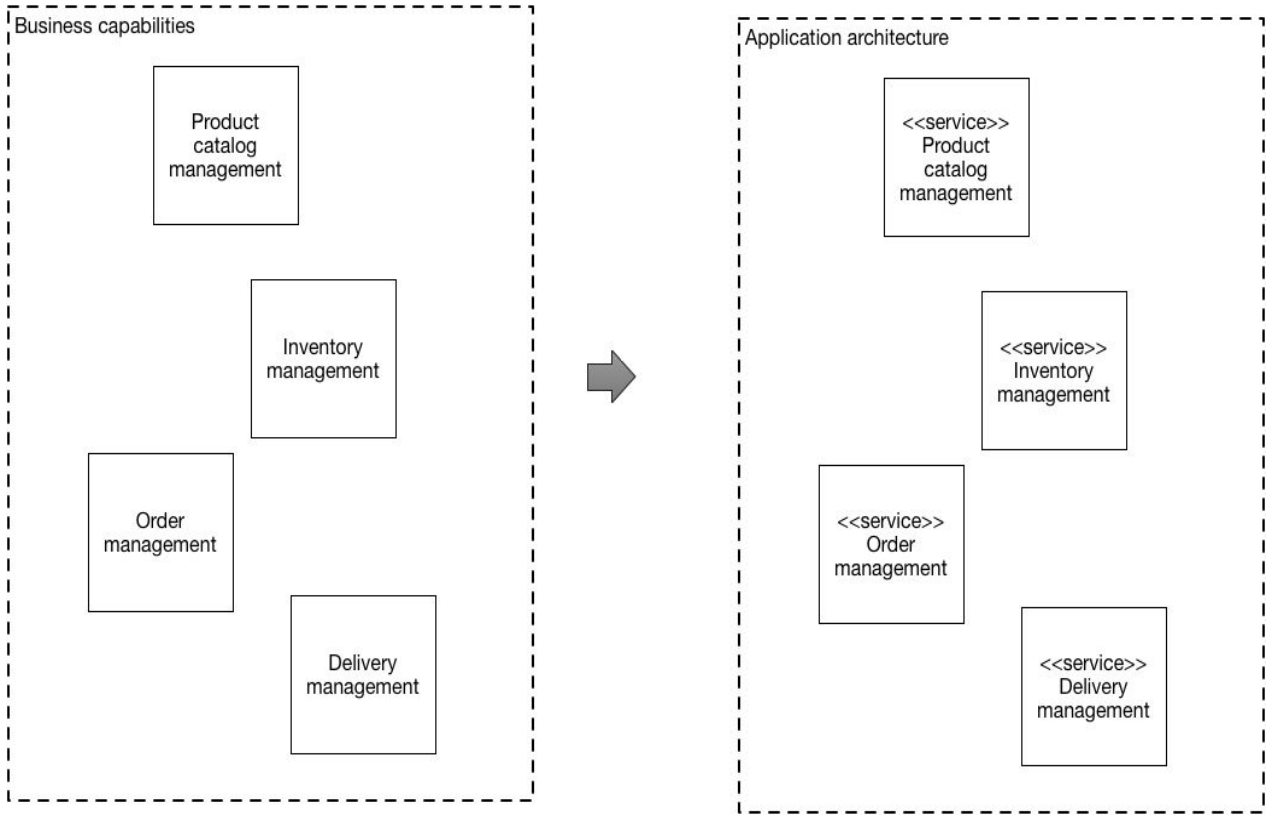


- Берем предметную область и выбираем проблемные области (поддомены)
- Вокруг поддоменов организуем сервисы

Минусы:

- Не всегда «правильная декомпозиция» проблемной области совпадает с орг-структурой и бизнес-процессами

Декомпозиция по бизнес-процессам
(decomposition by business capability)

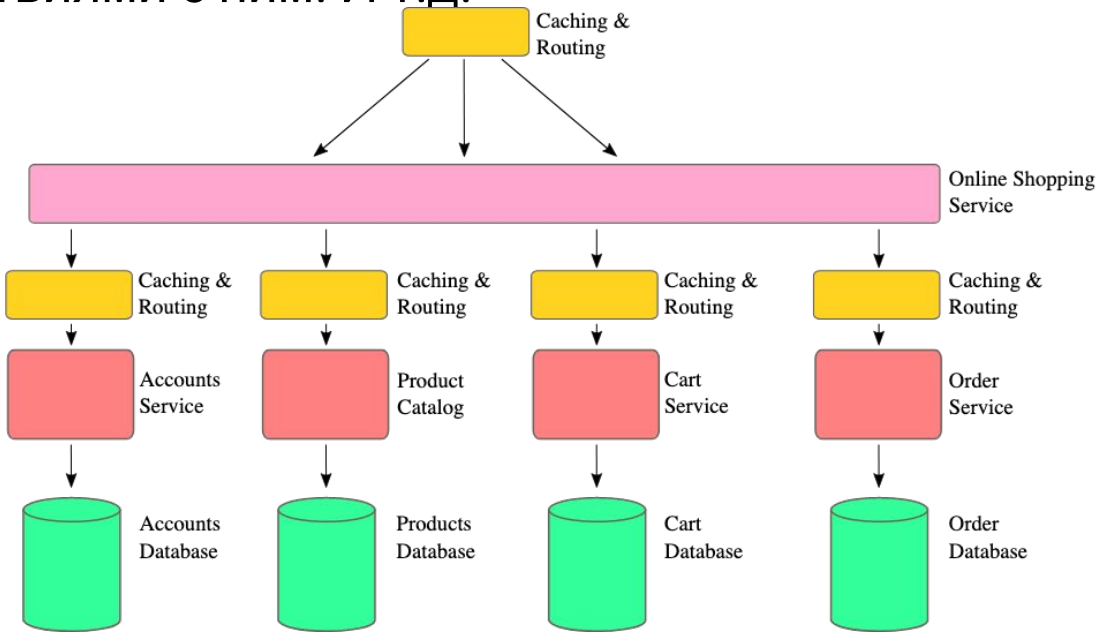


- Берем бизнес-процессы
- Вокруг бизнес-процессов организуем сервисы

Минусы:
бизнес-процессы не всегда оптимальны и мы эту неоптимальность можем утащить в архитектуру

Сервис-сущность (service-entity)

Есть интернет магазин "Вазон".
Архитектор ПО решил выделить (микро)сервисы "Продукт", "Заказ", "Клиент", "Корзина".
В сервис "Продукт" складывается вся бизнес-логика, связанная с продуктом и действиями с ним. В сервис "Заказ" складывается вся бизнес-логика, связанная с продуктом и действиями с ним. И т.д.



Сервис-сущность (service-entity)

Риск согласованных изменений. Чаще всего бизнес-логика, затрагивает несколько сущностей.
Какие изменения и в какие сервисы надо сделать, чтобы добавить доставку на дом?

- в сервисе «заказ» добавить адрес доставки, желательное время и доставщика
- в сервисе «клиент» добавить список избранных адресов доставки для клиента
- в сервисе «товар» добавить сущность список товаров

Или какие изменения и в какие сервисы надо сделать, чтобы добавить скидки по промокоду?
Как минимум надо:

- в сервис «заказ» добавить промокод
- в сервисе «товар» добавить действует ли скидки по промокоду на этот товар
- в сервисе «клиент» добавить список промокодов, которые выдали клиенту

https://www.michaelnygard.com/blog/2017/12/the-entity-service-antipattern/

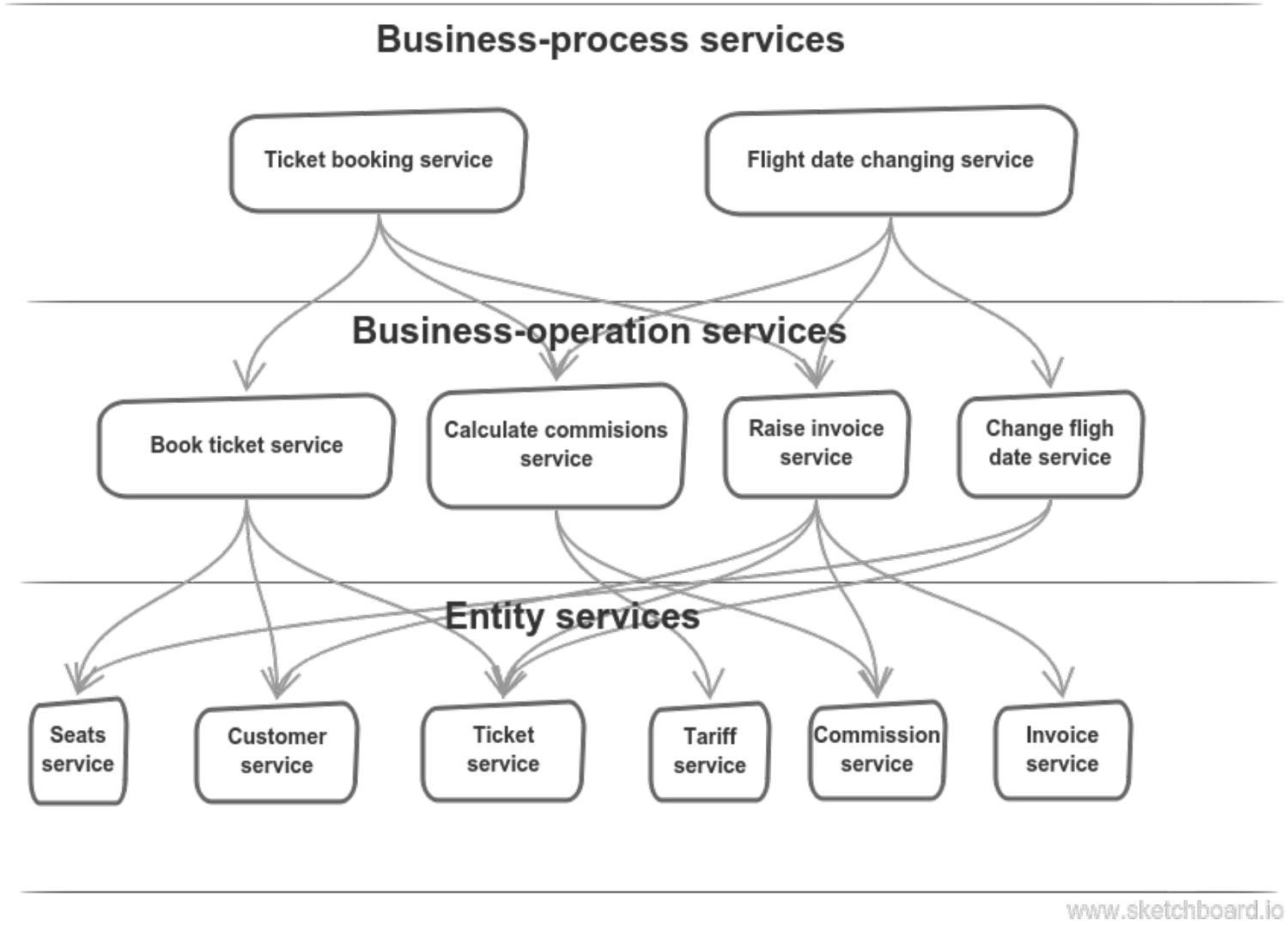
Сервис-сущность (service-entity)

Сервис-сущность - организация сервиса вокруг сущности, которая участвует в разных бизнес-процессах и может относится к разным предметным подобластям.

Минусы:

- В таком сервисе появляется бизнес-логика, которая друг с другом не связана
- Сервис правится разными командами, что мешает разработке.
- Сервис подтвергается бОльшей нагрузке
- Сервис становится излишне критичным: падение такого сервиса часто кладет, все связанные с ним

Сервис-хранилище



https://medium.com/hackernoon/wrong-ways-of-defining-service-boundaries-d9e313007bcc

Сервис-хранилище

Сервис-хранилище - сервис как прокси к хранилищу, в котором нет никакой бизнес-логики, кроме хранения. Вся бизнес-логика хранится на более высоком уровне

Минусы:

- Теряем в производительности, потому что отдаем данные не из базы напрямую, а через сервис прослойку
- Теряем в гибкости, потому что API сервиса обычно намного менее гибок, чем SQL или любой другой язык запросов
- Сервис подтвергается большой нагрузке (rps)
- Сервис становится излишне критичным: падение такого сервиса кладет всю систему

https://martinfowler.com/bliki/AnemicDomainModel.html

Когда декомпозируете на сервисы:

- Обращайте внимание на проблемную область
- Тестируйте сервисы на соответствие loose coupling & high cohesion принципу
- Пробуйте декомпозировать по предметной области (by subdomain)
- Пробуйте декомпозировать по бизнес-процессам (by business capability)
- Относитесь с осторожностью к сервисам-сущностям
- Относитесь с осторожностью к сервисам-хранилищам
- Декомпозируйте сервисы итеративно

Домашнее задание

Разделите ваш сервис на несколько микросервисов с учетом будущих изменений.

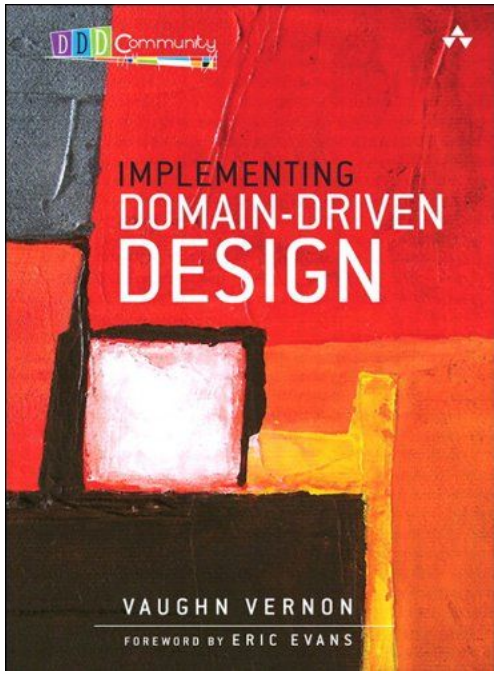
Попробуйте сделать несколько вариантов разбиений и попробуйте их оценить. Выберите вариант, который вы будете реализовывать.

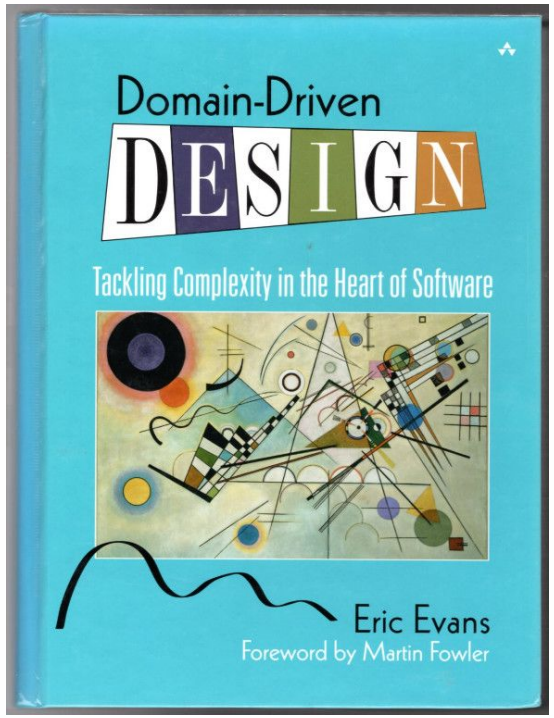
На выходе вы должны предоставить общую схему взаимодействия сервисов. Для каждого сервиса опишите назначение сервиса и его зону ответственности. Опишите контракты взаимодействия сервисов друг с другом.

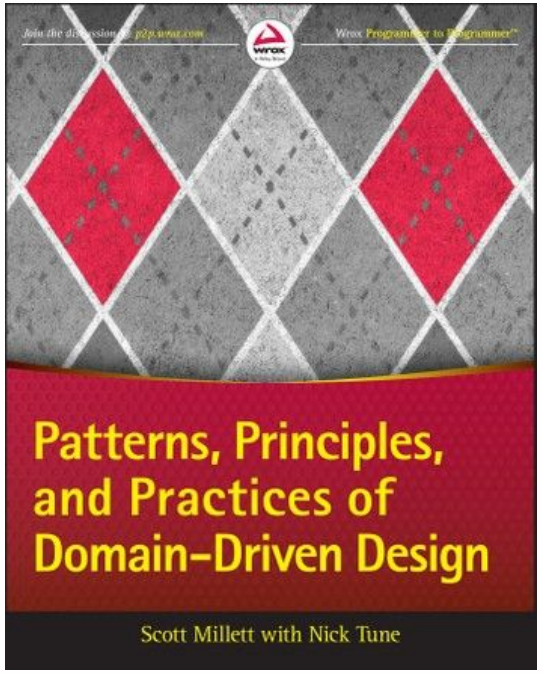
По желанию:

- В одном из вариантов попробуйте декомпозировать сервисы по предметной области или по бизнес-процессам.
- В одном из вариантов попробуйте декомпозировать «неправильно» и использовать подход сервисов-хранилищ или сервисов-сущностей и посмотрите, что получится.

Книги


DDD Community
IMPLEMENTING
DOMAIN-DRIVEN
DESIGN
VAUGHN VERNON
FOREWORD BY ERIC EVANS


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

Ссылки

- https://www.michaelnygard.com/blog/2017/12/the-entity-service-antipattern/ - статья про антипаттерн сервис-сущность
- https://martinfowler.com/bliki/AnemicDomainModel.html - статья Вадима Самохина про антипаттерны декомпозиции сервисов
- https://martinfowler.com/bliki/AnemicDomainModel.html - статья Фаулера про анемичную модель
- https://microservices.io/patterns/decomposition/decompose-by-business-capability.html - статья про декомпозицию по бизнес-процессам
- https://microservices.io/patterns/decomposition/decompose-by-subdomain.html – статья про декомпозицию по предметной области

Опрос

https://otus.ru/polls/6403/

Спасибо
за внимание!