1

Kitten: поиск элементов и работа с базовыми характеристиками. Часть 1

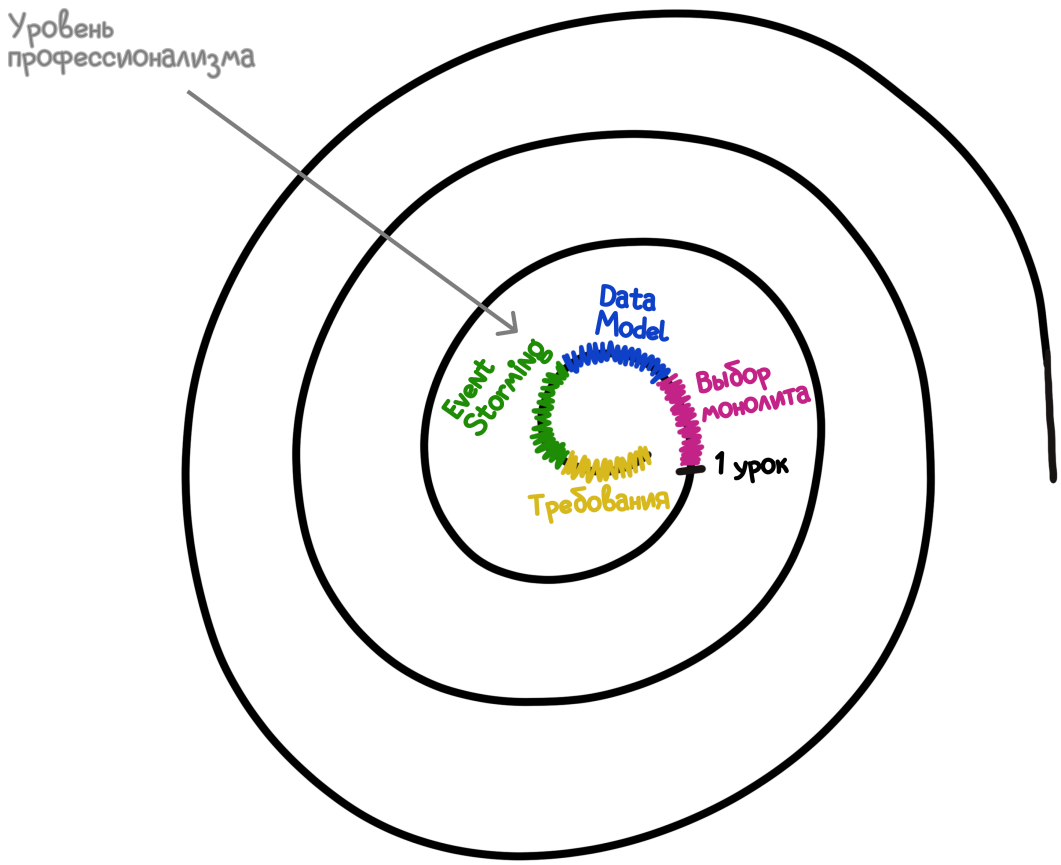
⏳ В первой части примерно 8000 слов — это эквивалент 20 страницам 12 кеглем. Мы ожидаем, что на чтение вам понадобится не менее 40 минут, а скорее 120 — чтобы всё как следует обдумать и перейти к домашке.

Содержание первой части урока:

📌 Привет! Это первый лонгрид курса «Анализ систем». В нём мы сделаем первую итерацию проектирования — разобьём проект на элементы и определим, какие между ними будут связи.

Для этого воспользуемся двумя подходами, которые мы описывали в курсе «Асинхронная архитектура»: **Event Storming** (ES) для анализа поведения системы и **моделью данных** для анализа структуры этой системы. Если не проходили тот курс, ничего страшного — главное мы подсветим. Если же проходили — просто освежите в памяти.

В результате у нас получится монолит, потому что это дёшево и по-другому мы пока не умеем.



Могу сказать, из чего состоит система, и предложить без обоснований

Небольшое напоминание про уровни мастерства. В курсе есть несколько таких уровней, и каждый урок соответствует решениям определённого уровня. Фактически это уровни, на которых Ибрагим умеет решать проблему. На первом круге — решения попроще и больше ошибок, на дальнейших — посложнее и ошибок поменьше.

Знакомство с главным героем и погружение в проблему

Таким Ибрагима видит нейронка. Ну кто мы такие, чтобы не согласиться.

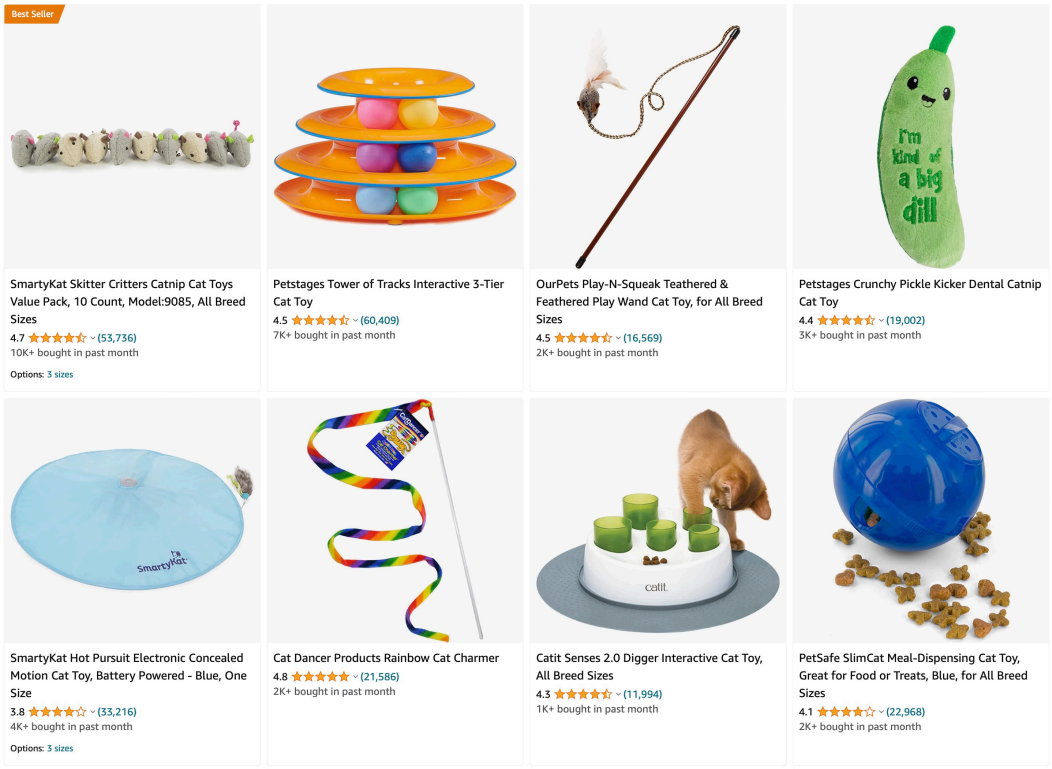
Знакомьтесь — это Ибрагим! Он разработчик, классно пишет код и вообще молодец, потому что любит котов, как и мы. Ибрагим — владелец кота по имени Кот.

Ибрагим хочет расти, чтобы не просто закрывать задачи в Джире, но и проектировать надёжные системы.

У него есть опыт нескольких проектов, которые он сделал с нуля, но этот опыт не особо удачный. Один сервис, который он делал полгода, вообще пришлось выкинуть на свалку из-за вечных ошибок, которые регулярно в нём возникали.

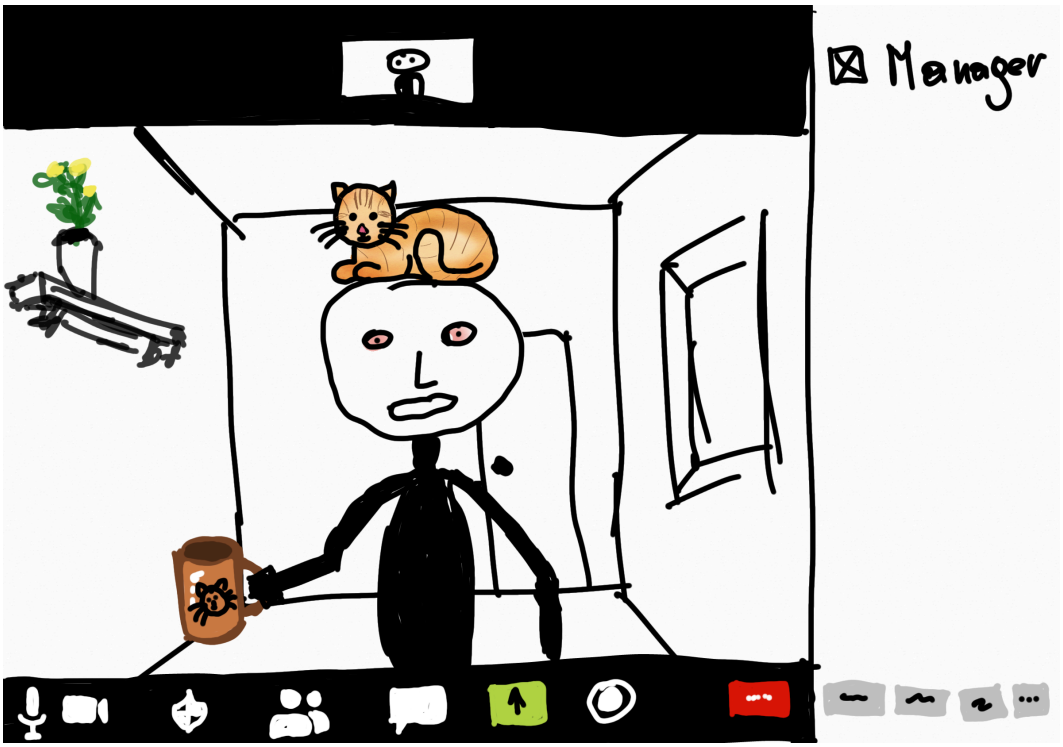
Попытки разделить монолит в другом проекте привели к распределённому монолиту. Из-за этого в системе сформировалась сильная связанность, что приводило к отказу всей системы, если падал один из сервисов. В итоге большая часть проектов Ибрагима либо не запускалась в срок, либо превращалась в кашу.

Внезапно случилось чудо, и Ибрагим устроился на свою первую работу архитектором в высокотехнологичный стартап, который продаёт игрушки для котов. В первый же день стало понятно, что кода нет, технологичностью не пахнет, разработчики появятся с недели на неделю, но пока их тоже нет.



Ибрагим хотел стать архитектором, но жизнь заставила его изучать кошачьи игрушки.

На первом рабочем созвоне менеджер рассказал, что проекту недавно исполнилось два года и внутри всё работает на эксель-таблицах, совместная работа над которыми вызывает проблемы с синхронизацией между коллегами. А ещё кто-нибудь постоянно забывает внести важную информацию, и данные вообще теряются, а коты-тестировщики не попадают по нужным клавишам, и таблицы постоянно расходятся, и их приходится править.



Это менеджер, который созванивался с Ибрагимом.

Стартапу очень нужна автоматизация — веб-приложение, которым сможет пользоваться любой сотрудник, включая кота. Важно, чтобы приложение поддерживало низкий time to market и не выходило за рамки бюджета, потому что бизнесу и так есть на что тратить деньги. Все необходимые требования, примеры таблиц и UX обещали скинуть на следующий день.

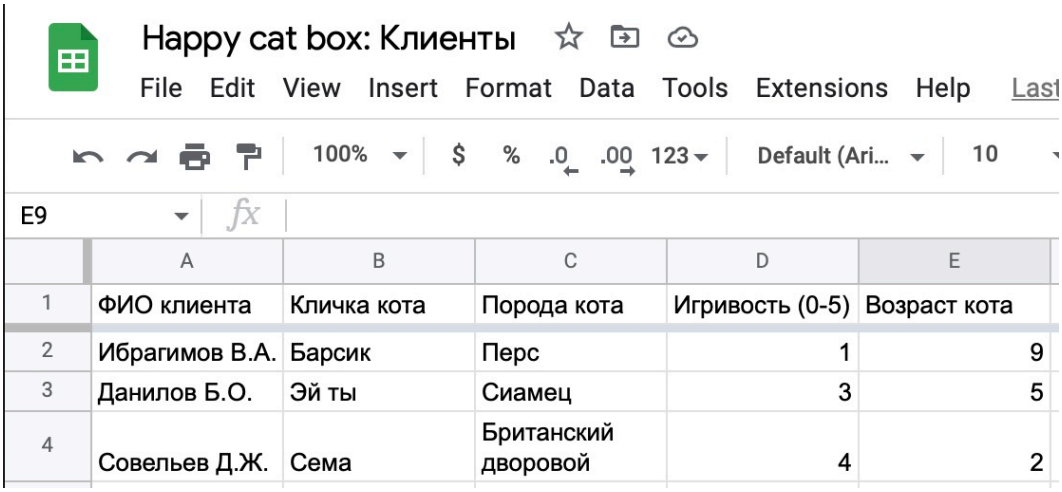
После созвона Ибрагим поймал себя на двух мыслях. Но кот его переубедил.

На следующее утро Ибрагим сварил кофе, открыл почту и обещанное менеджерское письмо.

✉️ from: topmanagment@happycatbox.io
to: ibrahim@happycatbox.io
subject: Обещанная информация по проекту

Привет!
В приложении таблицы, которые мы используем. Больше ничего не нашёл.

С уважением,
Топ-менеджер



	A	B	C	D	E
1	ФИО клиента	Кличка кота	Порода кота	Игривость (0-5)	Возраст кота
2	Ибрагимов В.А.	Барсик	Перс	1	9
3	Данилов Б.О.	Эй ты	Сиамец	3	5
4	Совельев Д.Ж.	Сема	Британский дворовой	4	2

Одна из гугл-таблиц, которыми пользуется компания для работы.

Ясно: кроме пары непонятных эксель-документов, больше ничего нет. Ну ок, работу в любом случае надо начинать:

🗓️ План работы над Happy Cat Box

1. Определить, что от нас хотят, и изучить требования
2. Решить, из каких компонентов будет состоять система
3. Выбрать гипотетическую структуру проекта
4. Подготовить описание решения для разработчиков
5. Передать решение в разработку

Ну что же, Ибрагим, погнали.

I. Определяем, что от нас хотят

Эксель-таблицы — это круто, но для полноценной системы их явно недостаточно. Ибрагим вспомнил, что на прошлой работе у него был разговор с главным бизнес-аналитиком Анатолием Исааковичем. Анатолий Исаакович сетовал, что его подчинённые криво работают с требованиями и поэтому он написал гайд, который в сокращённом виде выглядит так:

- если никогда не работали с требованиями, а разобраться в системе надо — попросите бизнес написать список действий, которые могут делать пользователи или другие системы;
- если бизнес не пишет требования, воспользуйтесь хитростью: напишите хоть какие-нибудь требования сами и дайте их отревьюить тому, от кого нужно получить требования. В таком случае человек найдёт ошибки, поправит и получит толчок, который позволит ему справиться с проблемой чистого листа.

Этот приём работает не только с требованиями. Самый показательный пример — ютуб-блогер и музыкант Ваганыч, который строил изолированную будку для записи вокала у себя в квартире. Он записывал и выкладывал процесс на ютуб, в комментарии к которому набежала куча людей, знающих, как лучше. Ваганыч собрал обратную связь, исправил ошибки и в итоге сделал вменяемую будку за минимальный бюджет.

Если вы не хотите строить звукоизолирующую будку для вокала, можно просто завести два аккаунта на SO.

Гайд, как определить, что от нас хотят

1. **Задавайте вопросы:** чем лучше вы поймёте требования, тем лучше будет итоговый результат. Лучшие друзья — *«Зачем?»* и *«Почему?»* вместо *«Что?»* и *«Как?».*

▼ Чтобы почитать подробнее, раскройте треугольник, там шпаргалка с объяснениями ↓

- **Если у вас появилось хоть какое-то сомнение в понимании — задавайте вопросы.** Это работает не только в software-проектах, но и в редактуре. На эту тему есть хороший пост Ильяхова.
- **Если встречаете неоднозначные формулировки** вроде *«выводить популярные товары на главной»,* **то переписывайте текст с большей детализацией:** *«Взять самые покупаемые товары за неделю и*

показывать их на первом экране сайта в блоке популярных товаров». Если ответственный за требования человек не может объяснить какое-то слово, воспользуйтесь методом 5-whys.

- **Если видите общее и абстрактное утверждение, всегда его уточняйте.** Например, утверждение *«система должна быстро отдавать результат»* для инженеров значит, что *«система должна отдавать результат за 1 наносекунду»,* а для пользователя и за 10 минут уже быстро и предел мечтаний.
- Бизнес живёт в своём контексте и считает многие вещи очевидными. **Если встретите непонятное — просите объяснить и выписывайте объяснение в отдельный словарь.** Например, в сети клиник продакт-менеджеры используют кучу терминов: conditions, «лабораторный справочник», «преаналитика». Если программист не будет знать глоссарий, он не сможет понять ни одной поставленной задачи или сделает совершенно не то, что от него ждёт бизнес.
- Если **не понимаете процесс, просите объяснить, как он работает,** добавить видео с действиями человека или скриншоты интерфейсов и дизайна.
- **Разбирайтесь с тем, как создаются данные,** при каких условиях, кто их создаёт, мутирует и удаляет. Если работаете над CMS — разберите весь процесс создания и публикации контента. Если над магазином — разберите весь процесс от добавления товара в корзину до доставки под дверь. Обратите внимание на систему прав: кто и что делает в системе, у кого какие роли, чем они отличаются, почему это работает именно так.
- **Обращайте внимание на state transition** — то, как работают переходы состояния из одного статуса в другой. Например, в магазине статус заказа перемещается из «не оплачено» в «передано в доставку», в редакции статус материала меняется из «неопубликованное» в «опубликованное». Понимание изменений состояния помогает понимать необходимые валидации и ограничения для бизнес-логики.
- Поведение пользователя может отличаться от того, как его видит бизнес. Сложно предугадать поведение пользователя, но **чем больше**

глупых вопросов с точки зрения бизнеса вы зададите, тем больше вероятность, что у вас получится обезопасить систему.

https://prod-files-secure.s3.us-west-2.amazonaws.com/c09bfca4-7fc3-4a36-9633-210e1b841b6b/dbd57cb9-1a6f-4553-917c-aa0051e466bb/B__mCNyVIAArXOJ.mp4

Классическая гифка с поведением пользователей

- К сожалению, **чем система сложнее, тем проще в ней может пойти что-то не так.** Чтобы это *«что-то не так»* не сломало систему, стоит изучить, что делать в случае разных проблем. Например, пользователь пытается оплатить товар, но на его карте нет денег. Что должно произойти в системе, если не приходят валидные данные? А что будет, если пользователь закажет овер 9000 товаров? А если начнутся проблемы с отправкой писем?

2. **Докапывайтесь до болей.** Бизнес почти всегда предлагает решения, а не говорит о проблемах. Докапывайтесь до проблемы и ищите решение, которое удовлетворит бизнес и не заставит вас переделывать систему целиком.

Например, бизнес видит, что у пользователей возникают проблемы с получением заказа после оплаты в магазине, и решает, что проекту нужен автоответчик на основе AI, который будет помогать пользователям справляться с их проблемами. Но штука в том, что автоответчик не решает проблему пользователей, потому что не ясно, в чём именно у них возникают эти самые проблемы: может быть, курьеры не могут дозвониться, потому что не знают номеров телефонов конечных пользователей, или ввод адреса при покупке настолько неудобный, что покупатели злятся и вбивают туда белиберду.

3. **Если требований нет, поищите в других местах:**

- **Любой документации к проекту.** Даже если она будет неполной, это поможет закрыть часть пробелов.
- **Исходном коде.** Иногда приходится работать в проектах, где надо переделать систему один в один на других технологиях. Например:

бизнес купил продукт и хочет избавиться от старой версии .NET и перевести систему на модный Rust, где разработчиков найти проще. В случае, если бизнес не может составить требования, можно изучить код, чтобы понять его поведение, и на основе этого составить требования, которые в любом случае придётся согласовать с бизнесом. Способ дорогой, но иногда других вариантов просто нет.

- **Аналогичных проектах.** Например, создавая магазин, можно взять требования к абстрактному базовому магазину и дополнить его контекстом компании. Аналогично для CMS, LMS, billing/accounting, бирж, блогов, продажи рекламы и других общих проектов.
- **Работе на месте.** Чтобы собрать ожидаемое поведение системы, мне (Антону) пришлось провести день со службой доставки и сборки заказов в igooods. Первую половину дня я провёл в «Ашане» с выделенным собирателем заказов, помогая выбирать товары и наблюдая процесс взаимодействия с платформой. А вторую — в машине с курьером, которому помогал развозить продукты. Благодаря этому я осознал процесс работы с точки зрения не только кода, но и реальных людей. Оказалось, что день «в полях» может рассказать о проекте больше, чем день изучения требований, кода и разговоров с бизнесом.
- **Мозговом штурме.** Собирайте людей в комнате, закрывайте двери и окна, оставляйте еду, задавайте проблемный вопрос и ждите, что получится. По похожему сценарию работают Storming-воркшопы, например Event Storming, или один из воркшопов по сбору требований.
- **Опросниках.** К примеру, аутсорс-компании, специализирующиеся на телеграм-ботах, имеют целые брифы на эту тему с вопросами *«с какими сервисами хотите интеграцию?», «планируете ли переносить бота в whatsapp business?», «в какой форме общается бот?».*

Требований не было, и Ибрагим пошёл на хитрость, предложенную Анатолием Исааковичем, — набросал требования сам.

Готовые требования Ибрагим отправил менеджеру, и спустя пару часов получил ожидаемый ответ:

> ✉️ from: topmanagment@happycatbox.io
> to: ibrahim@happycatbox.io
> subject: Обещанная информация по проекту [3]
>
> Привет!
>
> Ты неправильно понял, что мы хотим. Я всё переделал и написал, как надо. Подробнее в файле.
>
> Требования к системе
>
> С уважением,
> Топ-менеджер

Открыв ссылку, Ибрагим увидел полотно текста:

📁 **О Happy Cat Box**
Это простой магазин игрушек для котов, в котором коты или их хозяева могут купить ежемесячный набор игрушек по подписке. Игрушки подбираются под породу, пристрастия и характер котов. При этом:

- система мэтчинга игрушек для котов считается суперпрорывной на рынке, так как мы используем оценку вживую + оценку из соцсетей + нейронку;
- для производителей — система оценки тестировщиками считается суперпрорывной, потому что проверяет все возможные свойства (ни один кот не погиб).

Ожидаемое поведение работников

- Менеджеры описывают в админке информацию об игрушках, а также о том, как следить за ними, какие могут возникнуть проблемы.
- Нужно сделать, чтобы было как можно меньше работников, сократились расходы и сошлась юнит-экономика.
- Для тестирования качества игрушек нанимаются коты, которые на своей шкуре проверяют каждую игрушку. Чтобы коты помнили, что они работают, а не тыгыдыкают просто так, мы назвали игрушки *устройствами для игры,* что сохраняет вовлечённость котов в работу.

  И так далее. Полный набор требований вы найдёте тут: Требования к системе.

По вопросам всё оказалось не так просто. Вот выдержка из требований:

> Результатом тестирования устройства для игры будут отмеченные пункты в чек-листах тестирования с подробным описанием каждого пункта, а также заполненная анкета в конце тестирования с общим впечатлением по устройству для игры.

Возникают вопросы:

- непонятно, что такое *устройство для игры.* Если это термин, то стоит выписать его значение, контекст использования и почему этот термин появился;
- что за *чек-листы тестирования.* Из чего они состоят, кто их создаёт, мутирует, удаляет, как они заполняются;
- непонятно, зачем нужна анкета с общим впечатлением. Кто её валидирует, изменяет и где использует.

Основываясь на вопросах, можно дополнить требования и получить новые пункты с пояснениями:

☐ Для тестирования качества игрушек нанимаются коты, которые на своей шкуре проверяют каждую игрушку. Чтобы коты помнили, что они работают, а не тыгыдыкают просто так, игрушки было принято назвать устройством для игры. Благодаря исследованиям и тестированию выявили, что изменение термина увеличивает продуктивность котов-тестировщиков на 21%.

☐ Менеджеры должны добавлять гайд и чек-лист по проверке игрушек, которые проверяют и демонстрируют, как тестируется каждое устройство. Это помогает избегать интеллектуальной работы котов-тестировщиков. Менять гайды и чек-листы может менеджер, при этом ему необходимо отслеживать любые изменения. Если шаги тестирования хочет поправить тестировщик, ему нужно обратиться к менеджеру напрямую.

☐ Коты-тестировщики заполняют анкету с общими впечатлениями об игрушках. Благодаря этому компания может контролировать качество изделий и выбирать лучшего поставщика игрушек, а также анализировать эмоциональные впечатления от использования устройства. Никто, кроме тестировщика, не может менять свой отзыв.

Воспользовавшись советами с прошлой работы, Ибрагим задал вопросы и написал уточнения менеджеру:

✉️ from: ibrahim@happycatbox.io
to: topmanagment@happycatbox.io
subject: Вопросы и уточнения по требованиям
Привет!

Спасибо за присланные требования. У меня появились вопросы по документу с требованиями:

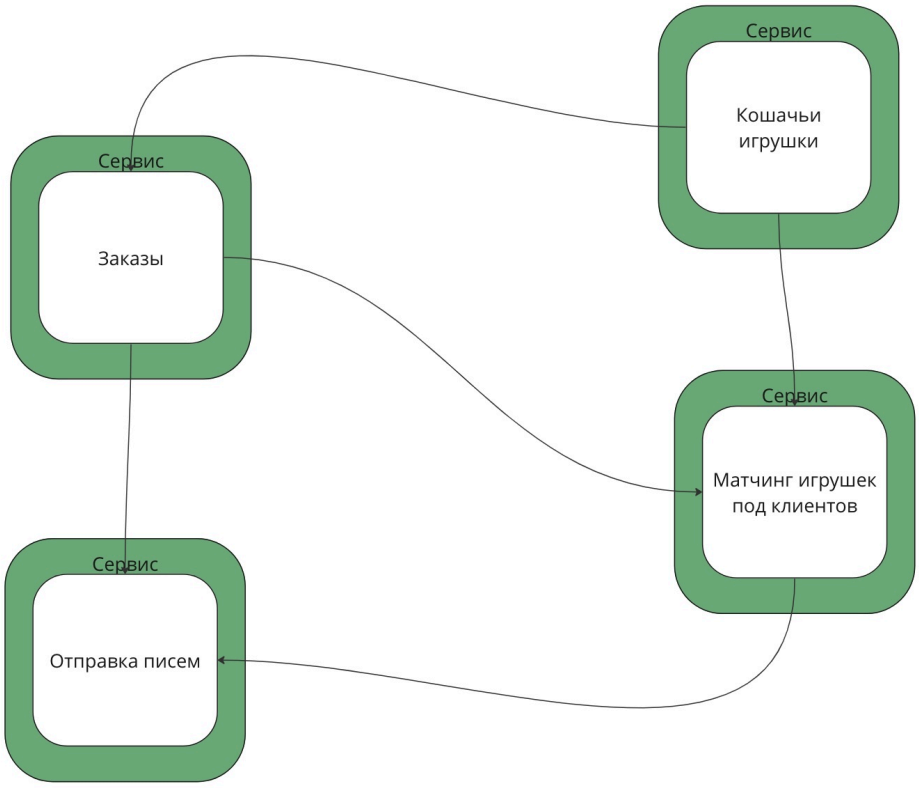
- что происходит, если у сотрудника отрицательный баланс и он получает штраф? Какой максимальный предел отрицательного баланса и что должно произойти, если он его набирает?
- если ежемесячная коробка не собралась или собралась некорректно, как должен происходить процесс повторной сборки и отправки? Сколько раз можно отправлять коробки пользователю повторно и как отслеживать недобросовестных пользователей, которые хотят на халяву получить больше игрушек?
- ...

После обмена письмами Ибрагим наконец дополнил требования и порадовался, что первый пункт в списке задач выполнен.

🗓️ План работы над happy cat box
1. ~~Определить, что от нас хотят, и изучить требования~~
2. Решить, из каких компонентов будет состоять система
3. Выбрать гипотетическую структуру проекта
4. Подготовить описание решения для разработчиков
5. Передать решение в разработку

II. Ищем элементы системы

Ибрагим с ходу решил выделить модули заказов и игрушек, добавить модуль отправки писем и работы с мэтчингом игрушек для пользователей, а потом сделать отдельный сервис под каждый модуль.



Первое, что пришло Ибрагиму в голову, когда он занялся работой над happy cat box.
Посмотреть изображение поближе

Но тут вдруг Кот заговорил:

Ибрагим вспомнил, как вытаскивали два сервиса: сервис цен продуктов и сервис отправки нотификаций. Это выглядело самым очевидным вариантом, но в итоге вышло так, что любые изменения данных в сервисе цен приводили к поломке половины системы, потому что из-за нового отображения цены ломались все запросы клиентов, и потом приходилось долго это разгребать. А нотификации из мелкого сервиса превратились в штуку, без которой не работало даже мобильное приложение. При этом вынос этих сервисов отнял у Ибрагима два года разработки, и в итоге решение никак никого не ускорило.

Потом Ибрагим вспомнил курс по асинхронной архитектуре (AA), который он прошёл в прошлом году. В курсе были два момента, которые могли помочь с его нынешним проектом. Во-первых, это инструмент **Event Storming** для определения бизнес-процессов и связей между элементами системы. Во-вторых — **модель данных**, которая показывает, какие именно данные нужны в каждом из элементов.

Давайте вместе с Ибрагимом воспользуемся обоими этими инструментами.

2.1. Event Storming (ES)

Event Storming придумал Альберто Брандолини в 2012–2014 годах. Это воркшоп, идея которого заключается в том, чтобы запереть всех сотрудников, которые работают над системой, в одной комнате, дать им инструмент и время для выгрузки знаний из головы, то на выходе получится схематичное описание системы, с которым согласен каждый участник данного мероприятия. А ещё разбитую комнату и пару нервных срывов участников, но это уже совсем другая история.

Чтобы провести Event Storming, нужно учитывать несколько условий.

Во-первых — **это не технический воркшоп, в нём не говорят про классы, сервисы, адаптеры и базы данных.** В нём отвечают на три вопроса:

1. как ведёт себя система для разных пользователей?
2. из каких процессов состоит система?
3. как процессы связаны между собой?

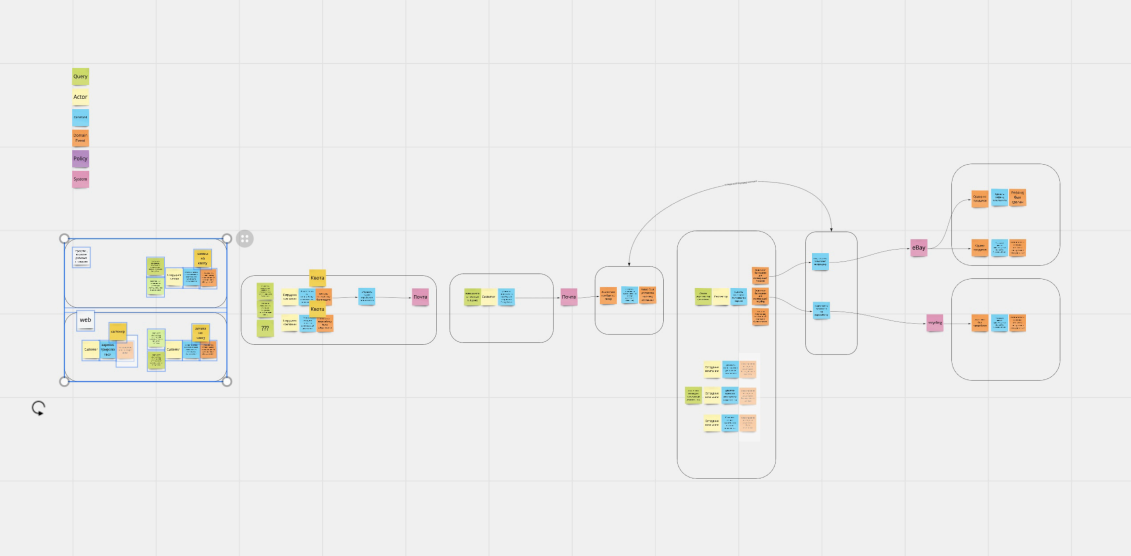
В качестве инструмента для выгрузки знаний можно использовать стикеры разных цветов, огромную белую стену и маркеры.

Те самые стикеры на белой стене

▼ Что делать, чтобы стикеры наполнились смыслом ↓

- Найти все события, которые происходят в системе.
- Для каждого события определить команды (процессы), акторов и условия выполнения команд.
- Сгруппировать команды и найти необходимые данные для работы команд.

В идеальном мире Alberto Brandolini представлял Event Storming офлайн-мероприятием со стикерами на стене. А потом случился ковид.



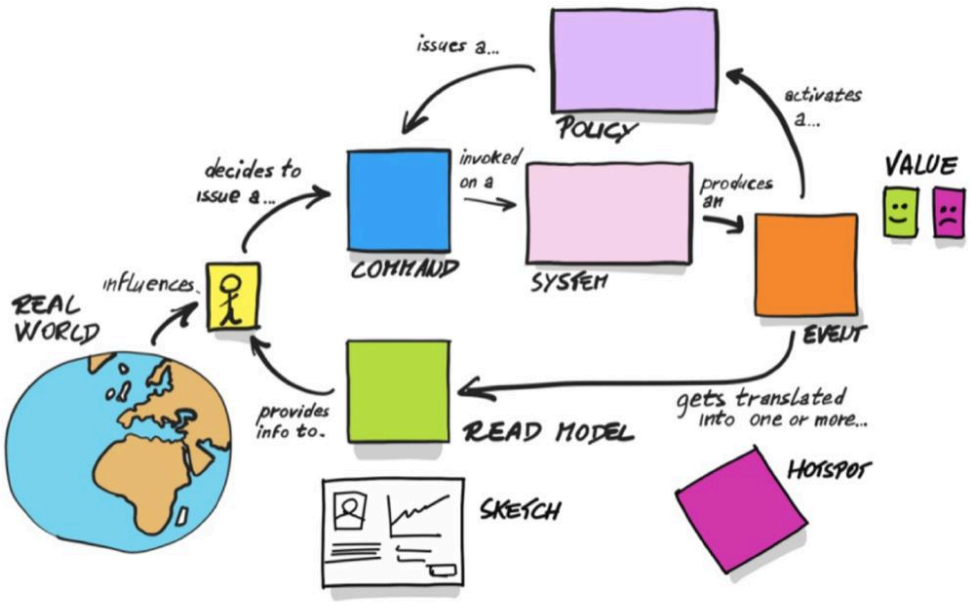
Во времена ковида и распределённых команд ES-сессию успешно перевели в Miro. Эта иллюстрация — результат факультатива курса AA, где разбиралась архитектурная ката. Полный разбор системы: ссылка на факультатив на ютубе и рутубе)

Посмотреть изображение поближе

У каждого цвета своя роль. Существует шесть минимально необходимых типов стикеров:

Цвет стикера	Описание
Жёлтый	Актор, запускающий команду. Может быть живым человеком с определённой ролью администратора, менеджера или клиента, может быть неживым — мобильным приложением или умным чайником.
Голубой	Действие, которое совершает система. Описывается в настоящем времени и говорит о том, *«что»* произойдёт, а не *«как».* Checkout может быть командой, потому что говорит о том *«что»,* а params validator, produce event и store user record — нет, так как говорит, *«как именно»* произойдёт действие.
Оранжевый	Событие (в AA-курсе называлось бизнес-событием), которое является результатом выполнения команды. Всегда описывается в прошедшем времени и говорит о произошедшем результате действия, которое нельзя отменить. То есть account registered — событие, а create user, user maybe created — нет. Событие говорит

Цвет стикера	Описание
	как о положительном, так и об отрицательном результате. Может вызвать другую команду, иногда через policy.
Фиолетовый	Полиси (policy), условие. Иногда для вызова команды через событие необходимо соблюдение какого-то условия. Например, для вызова команды отправки заказа в доставку надо проверить, что заказов собралось больше трёх штук. За подобные условия отвечает policy.
Розовый	Внешние системы (external system). Другие системы, с которыми взаимодействует наша, например эквайринг, который как-то списывает деньги. Здесь неважно, из чего состоит эквайринг, важно указать, просто что он есть. Т.е. внешнюю систему стоит рассматривать как «черный ящик»: мы знаем что приходит на вход и выход, а как именно она работает — не знаем. Бывают ситуации, когда система ничего не получает на вход. Пример: крон говорит, что нужное время наступило, или гитхаб, который говорит о том, что пользователь отправил коммит в нужный репозиторий (через вебхук). Всегда возвращает событие как результат работы.
Зелёный	Модель для чтения (read model). Чтобы выполнить команду, актору надо получить информацию о системе. Например, чтобы сделать чекаут в интернет-магазине, надо знать, что лежит в корзине. Read model говорит о данных, без которых актор не может принять решение, выполнять команду или нет. Собирается либо из событий, либо из локальных данных.

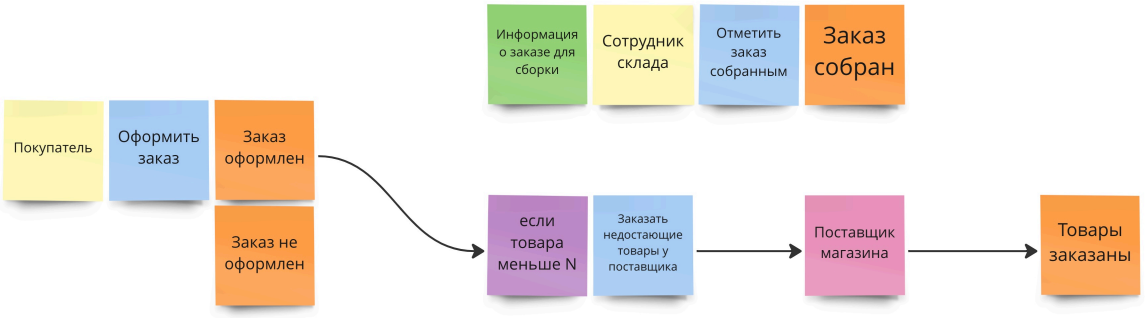


Все стикеры и как они друг друга вызывают

К сожалению, людям сложно без аналогий из реального мира (и это нормально). Если вам сейчас сложно, то воспользуйтесь примером, который поможет в ситуациях, когда надо объяснить ES за 3 минуты.

▼ Смотреть пример чекаута в магазине ↓

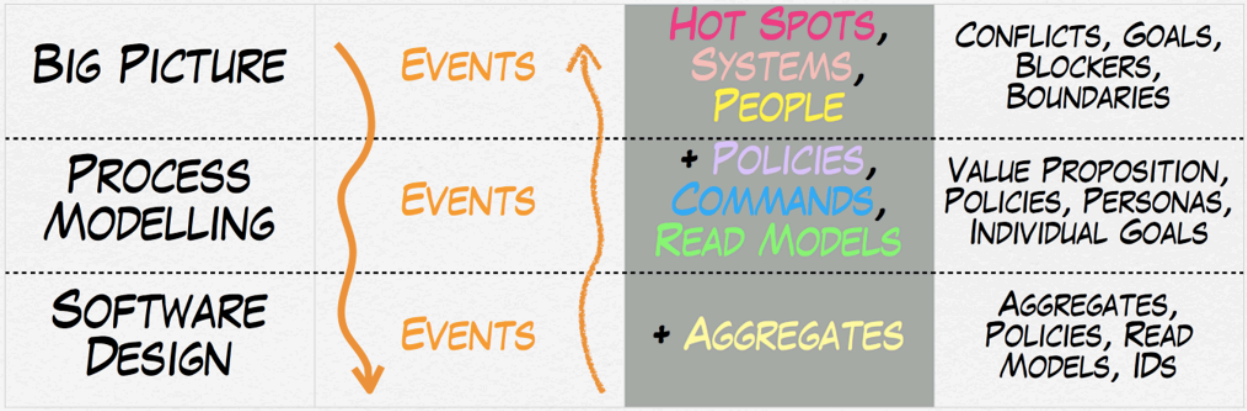
В придуманном чекауте три процесса: оформление заказа в магазине, сборка заказа и пополнение склада от поставщика, если товар начинает заканчиваться. Вот так стикеры описывают роли в системе.



Так выглядит пример из реального мира.
Посмотреть изображение поближе

В воркшопе три уровня, над которыми он позволяет работать:

- **big picture** — разбор всех процессов в системе;
- **process modeling** — разбор отдельного процесса;
- **software design** — мапинг domain model pattern на ES.



Отличия трёх уровней воркшопа. Оригинал

Дополнительно можно использовать собственную версию воркшопа: подключать кастомные стикеры и удалять ненужные базовые. Например, мы не фанаты агрегатов, в которых выполняются команды. Для разбора всей системы мы используем policy, команды и read models. A pain points (иногда встречается название hotspots) считаем оверкиллом.

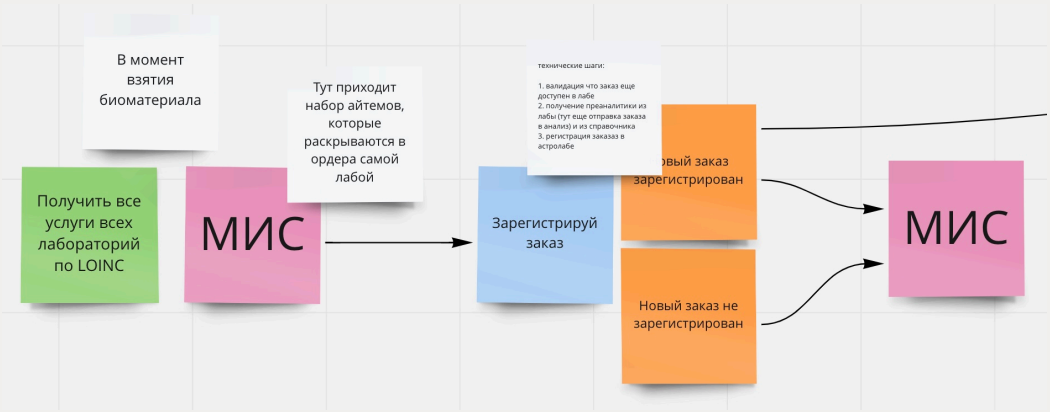
Заумь от Антона о том, какие ещё есть стикеры в Event Storming и почему pain points — оверкил

▼ Необязательный контент для большего погружения в тему
Как я уже говорил, мне хватает шести видов стикеров, но, кроме них, есть ещё два, которые могут вам встретиться.

Цвет стикера	Цель	Для чего
Жёлтый	Агрегат	Это такой контекстный набор данных, вокруг которого выполняются команды. На примере магазина выделим агрегат Order, который выполняет чекаут и рефанд команды и хранит информацию о добавленных айтемах и пользователе. Так как мне ближе функциональный подход, когда данные и поведение разделяются, я опускаю агрегаты. Если вам ближе объектно ориентированный подход — агрегаты могут вам помочь лучше разобраться в системе.
Розовый *(перевёрнутый на 45 градусов)*	Pain points / Hot spots	Такими стикерами отмечают любые опасения вокруг системы. Это могут быть бутылочные горлышки, действия, которые надо вызвать руками и которые требуют автоматизации, отсутствующая документация или domain knowledge. В моём случае я обычно использую стикер с комментариями, так как это выглядит очевиднее плюс уменьшает количество абстракций в итоговой модели.
Белый	Комментарии	Дополнительный стикер, который приходится использовать. Любые заметки, которые содержат дальнейшие шаги, контекст,

Цвет стикера	Цель	Для чего
		объясняющий стикер, или информацию для будущей работы.

Пример из проекта: тут говорится контекст, когда медицинская информационная система вызывает команду, и технические шаги, которые выполняются в команде.



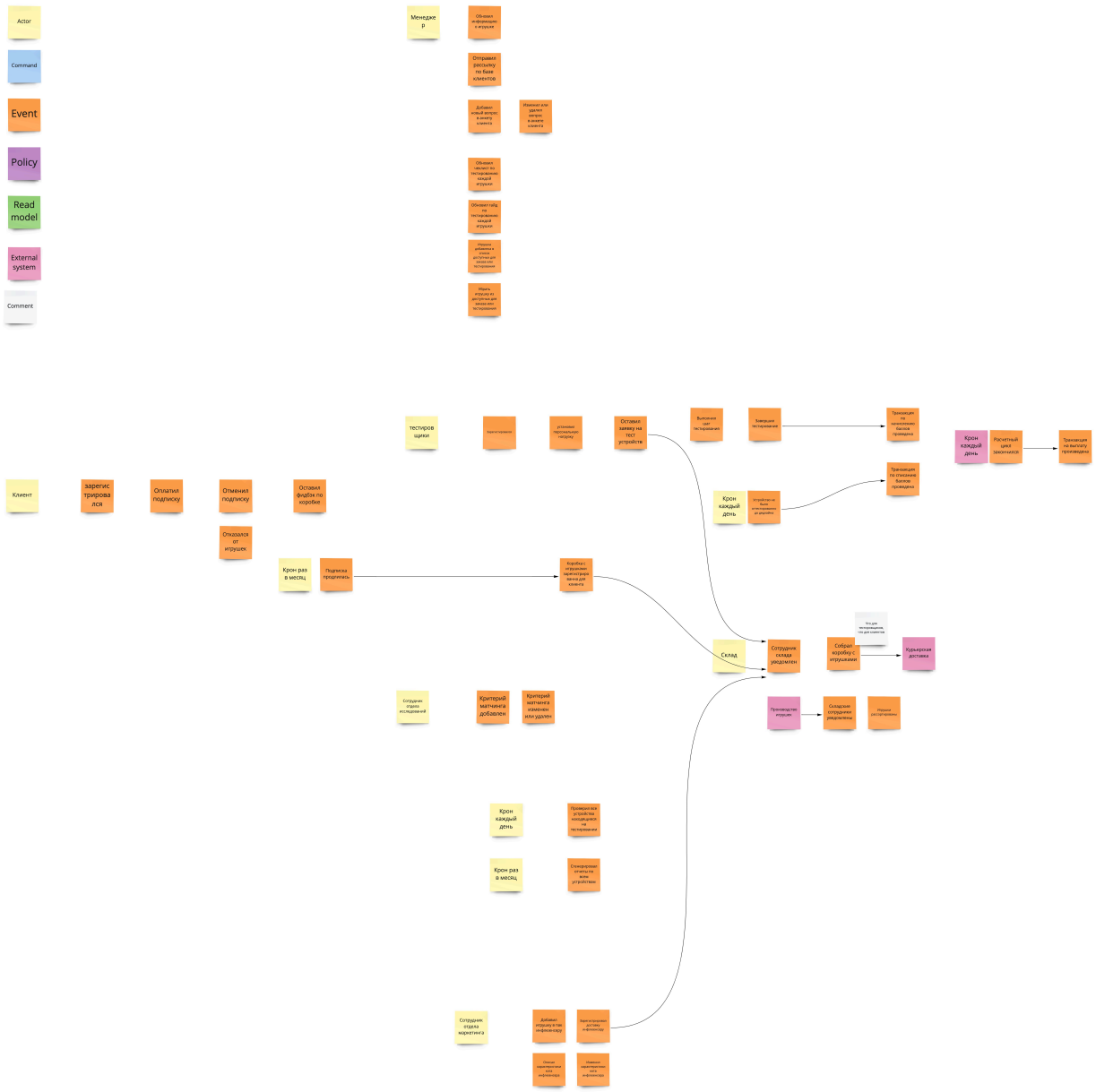
Посмотреть изображение поближе

Давайте посмотрим, как Ибрагим переведёт имеющиеся требования в ES-модель.

- [ ] Шаг 1. Поиск всех событий в системе
- [ ] Шаг 2. Добавление команд, акторов и policy
- [ ] Шаг 3. Поиск контекста и данных для отображения

Шаг 1. Поиск всех событий в системе

Для начала Ибрагим нашёл происходящие события и акторов. Он отобразил события в хронологическом порядке их проявления, а те, которые вызывают другое действие, проиллюстрировал стрелками.



ES. Шаг 1. Поиск всех событий в системе

System Analysis course_ модели для уроков.pdf

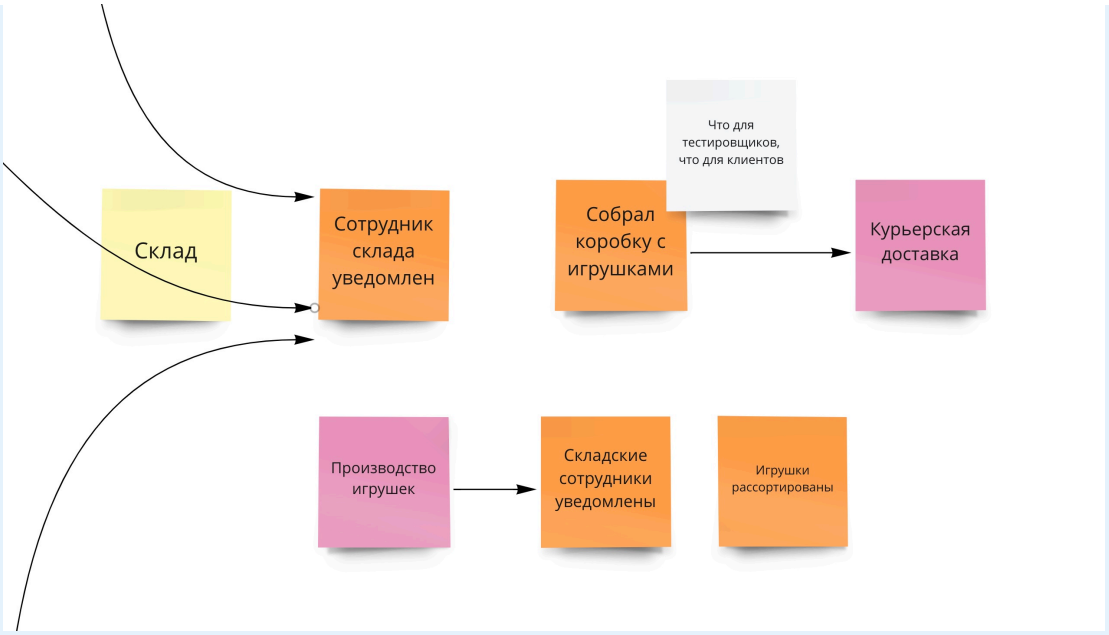
Скачать изображение в ПДФ

📌 **Нюансы**

1. **В EventStorming присутствует временная линия**. Т.е. события, которые происходят с актором раньше, находятся левее, чем события, которые происходят позже. Например: сначала пользователь зарегистрируется, только после этого сможет оформить заказ, следовательно стикер с событием регистрации будет левее, чем стикер с событием оформления заказа.

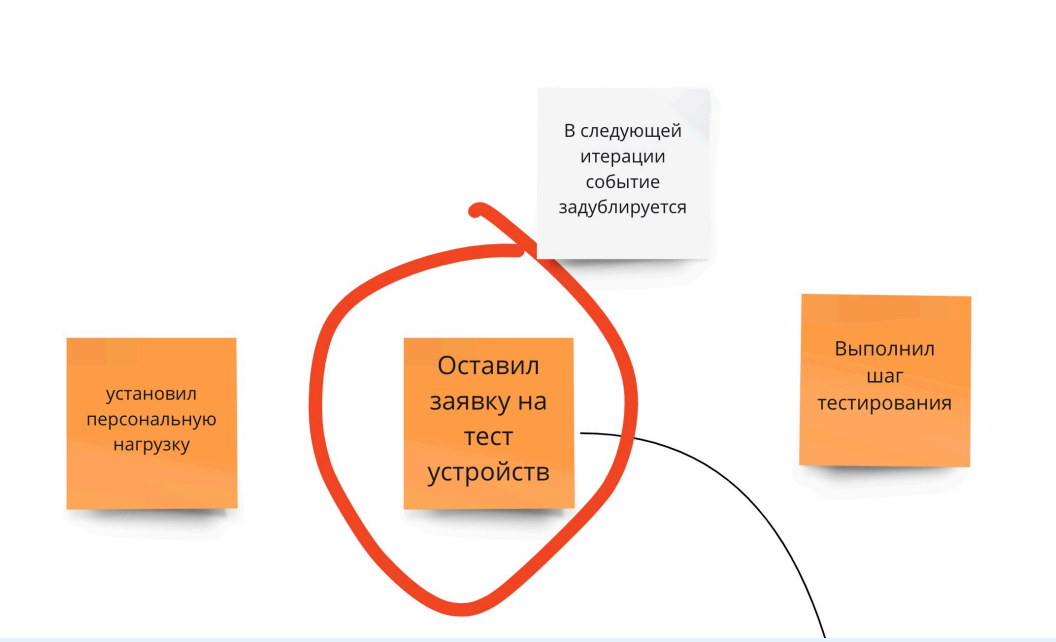
Обратите внимание, что сначала происходит процесс регистрации клиентом, только после этого оплата. А событие отмены подписки возможно только после того, как произойдет событие оплаты подписки.

2. **Если для контекста надо добавить что-то, кроме событий, смело добавляйте.** Это может быть актор, система, которая отправляет события, и комментарии, помогающие лучше понять, что происходит.



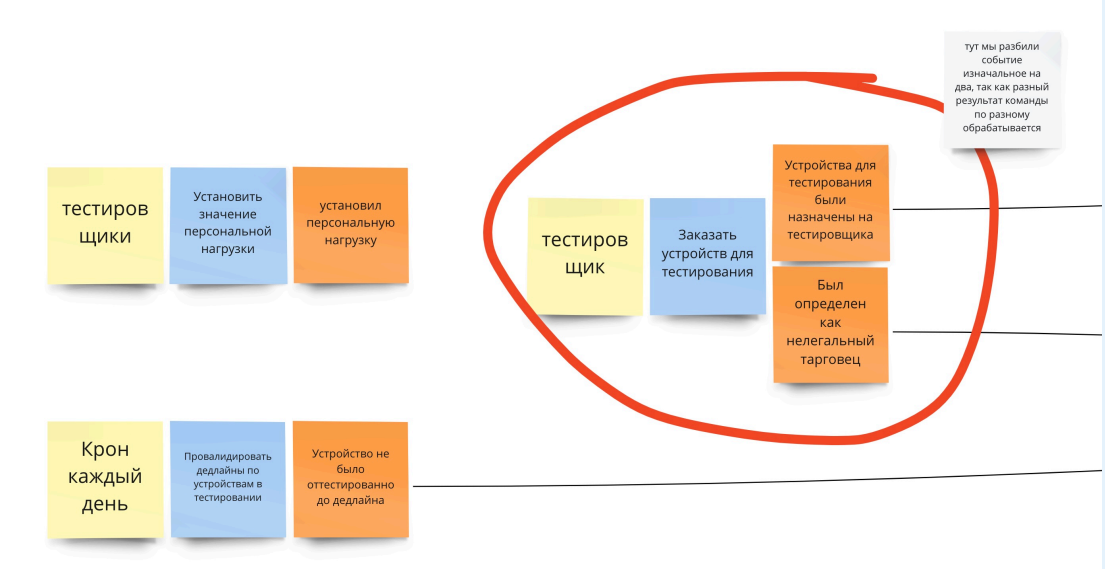
Добавление акторов, комментариев и внешних систем помогает лучше объяснить систему.
Посмотреть изображение поближе

3. **Не нужно выписывать абсолютно все события.** На этом шаге важно найти только базовые, которые запускают следующий шаг. Если пропустите событие на этом шаге, оно добавится на следующем.



Посмотреть изображение поближе

Дальше оно может разделиться на два, три и больше:



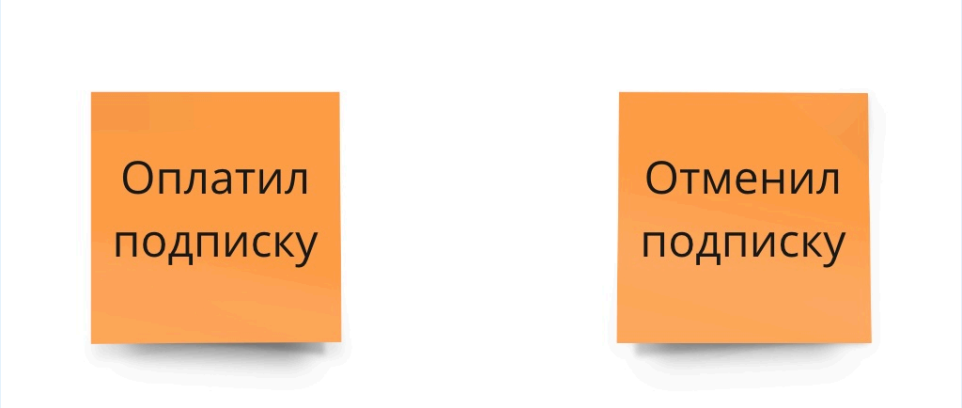
Дальше оно может разделиться на два, три и больше.
Посмотреть изображение поближе

4. **События — это не команды.** События — это случившиеся в прошлом действия. Они необратимы и пишутся в прошедшем времени. «Необратимые» значит, что единственный способ

изменить ситуацию — вызвать другую команду. Если пишете «зарегистрировать пользователя», то, скорее всего, описываете команду, если «пользователь зарегистрирован» — событие.

5. **События, которые вызывают друг друга, отличаются от событий, которые происходят последовательно во времени. Вызывают друг друга.** В этом случае произошедшее событие — это триггер для следующего события. Например, тестировщик отчитывается о завершении тестирования игрушки, а нам надо зачислить ему за это деньги. Первое событие запускает логику, результатом которой будет совершённая транзакция на зачисление денег.

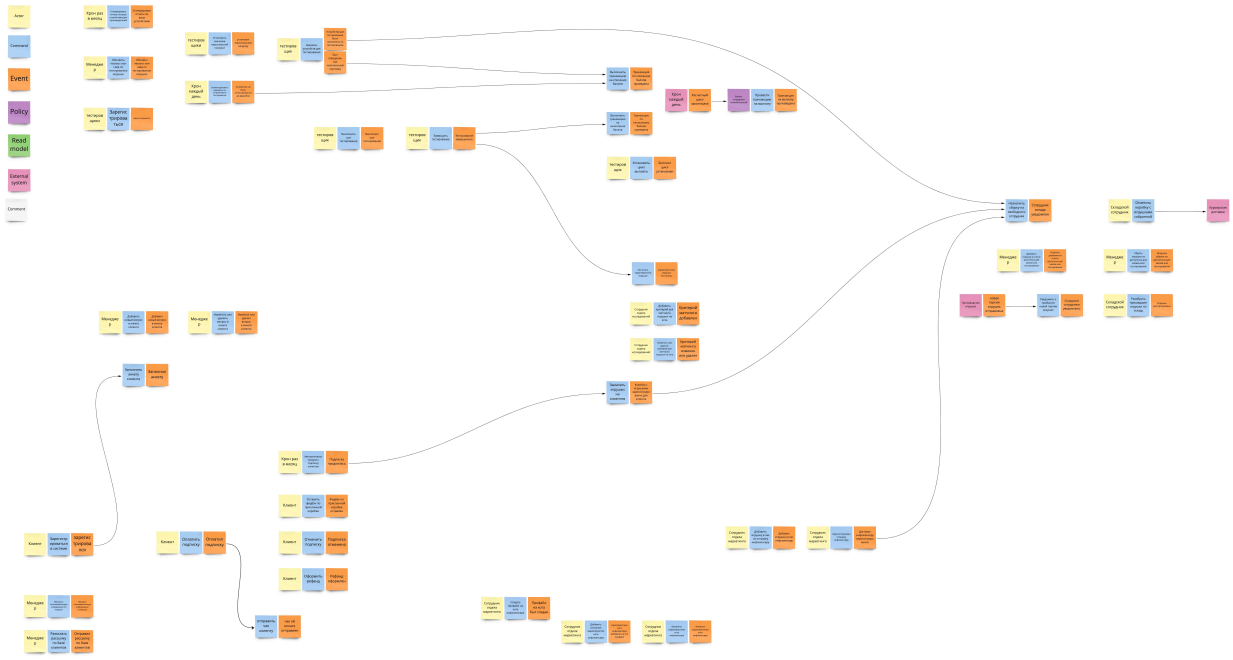
Происходят во времени. В этом случае событие «отменил подписку» может произойти только после того, как совершено событие «оплатил подписку». Но оплата подписки не приводит к автоматической отмене подписки.



- [x] ~~Шаг 1. Поиск всех событий в системе~~
- [ ] Шаг 2. Добавление команд, акторов и policy
- [ ] Шаг 3. Поиск контекста и данных для отображения

Шаг 2. Добавление команд, акторов и полиси

После того, как Ибрагим нашёл все базовые события, ему нужно их сгруппировать по командам, добавить к командам актора или указать, какое из событий вызывает команду. А ещё добавить стикеры с полиси во всех местах, где они нужны. Группировать в команды нужно по действиям, которые они совершают.



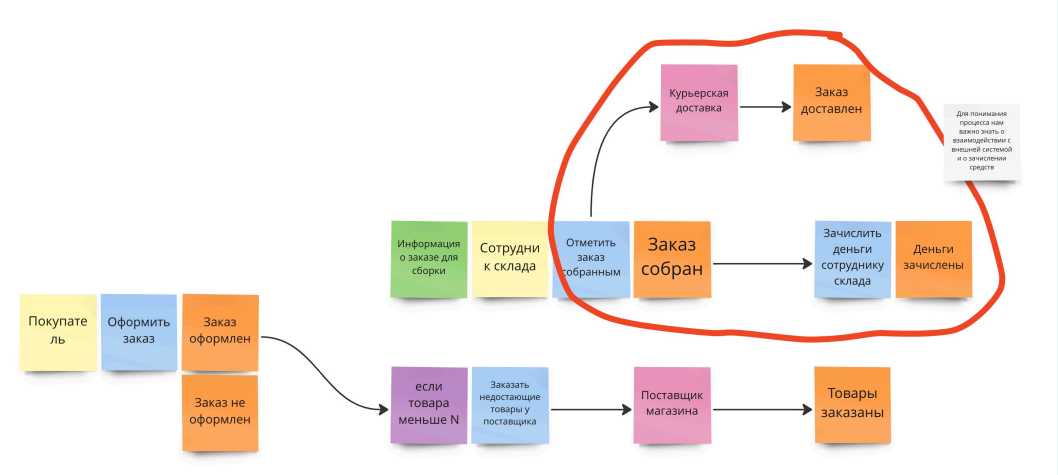
ES. Шаг 2. Добавление команд, акторов и policy

System Analysis course_ модели для уроков.pdf

Скачать изображение в ПДФ

📌 **Нюансы**

1. **Полиси (policy) стикер нужен для лучшего понимания контекста процесса.** Это похоже на комментарии в коде, которые не имеют смысла, если описывают очевидное. В ES так же: полиси *«если событие получено»* не даёт нужного контекста, но увеличивает когнитивную нагрузку на человека, который будет изучать ES-модель.
2. **Аналогичная ситуация с полиси «данные валидны».** Вместо этого сообщения добавьте описание контекста, в котором событие запускает команду: *«если* ***на складе осталось меньше N товара нужной марки,*** *то запустить команду на заказ недостающих товаров у поставщика магазина».* Жирным выделен кусок с бизнес-контекстом, в котором запускается команда, и это то, что нужно для ES-модели.
3. **В полиси-стикерах важно избегать технического описания.** В контексте ES-модели нам всё равно, откуда получено событие, проведена ли валидация данных и есть ли нужный id в системе. Потому что всё это — техническое описание работы алгоритма, а нам важно описать бизнесовый контекст, в котором работает система.
4. **Внешняя система всегда возвращает событие, а вызывается из команды, а не события.** Иногда возникают ситуации, когда важно и событие, и вызов внешней системы, — в Happy Cat Box такой ситуации нет. Для примера вернёмся к процессу покупки товара в магазине: если магазину нужно отправить заказ в логистическую компанию и заплатить за это, то придётся указать и событие, и вызов внешней системы из команды.

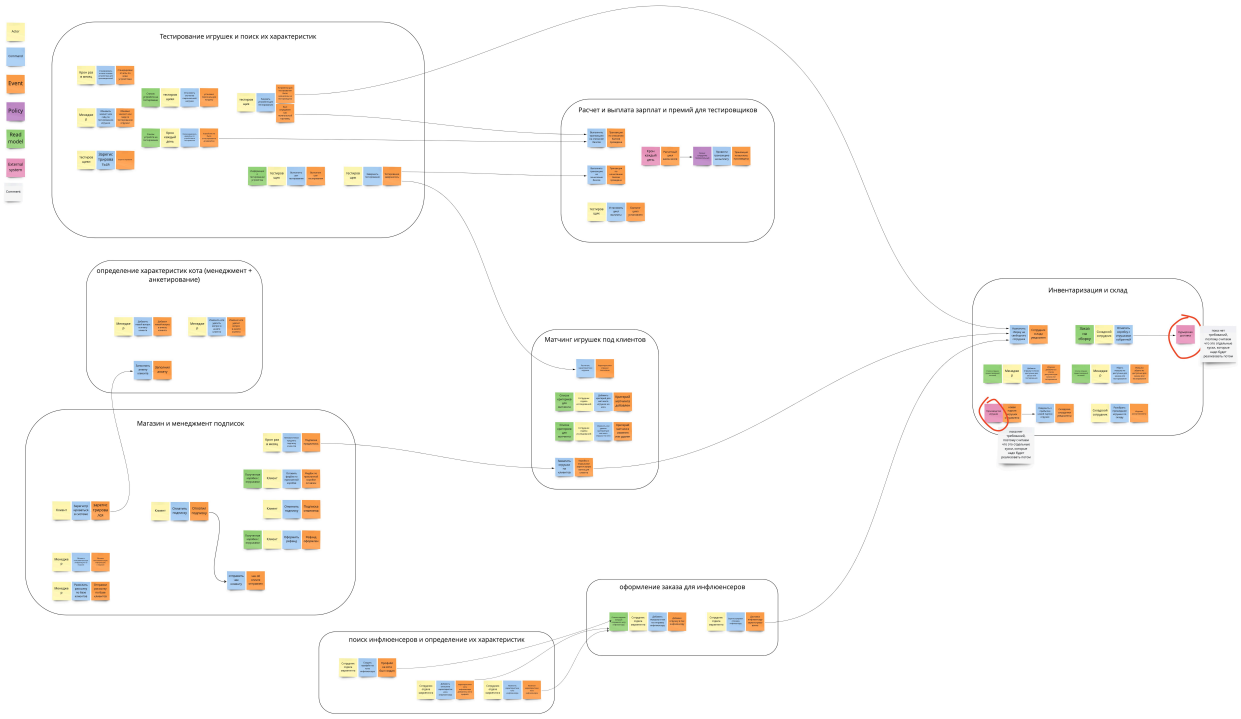


Команда на отмечание заказа собранным не только генерирует бизнес-событие, но и взаимодействует с внешней системой курьерской доставки.
Посмотреть изображение поближе

- [x] ~~Шаг 1. Поиск всех событий в системе~~
- [x] ~~Шаг 2. Добавление команд, акторов и policy~~
- [ ] Шаг 3. Поиск контекста и данных для отображения

Шаг 3. Поиск контекста и данных для отображения

На этом шаге Ибрагим предположил, как собрать команды и события в группы стикеров с общим контекстом выполнения — например, группу команд, которые связаны тестированием игрушек. Помимо этого, Ибрагим определился с моделями на чтение — необходимым набором информации, без которого акторы не смогут принять решение вызова команды.



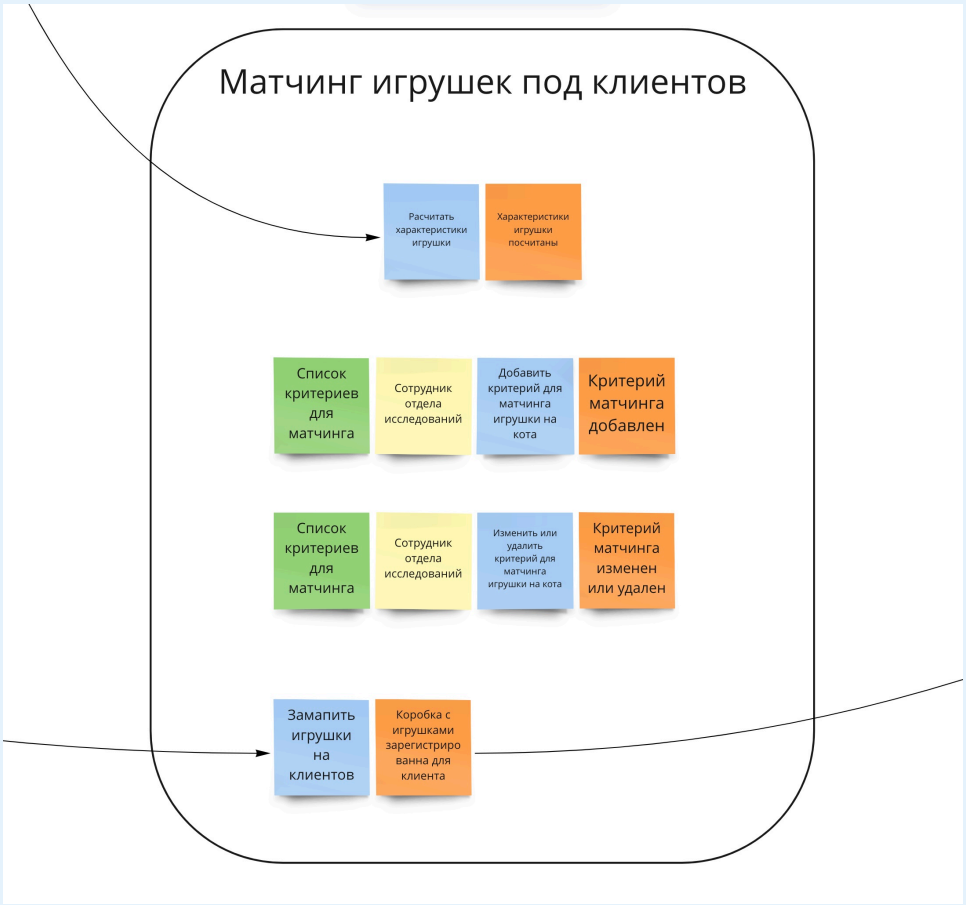
ES. Шаг 3. Поиск контекста и данных для отображения

System Analysis course_ модели для уроков.pdf

Скачать изображение в ПДФ

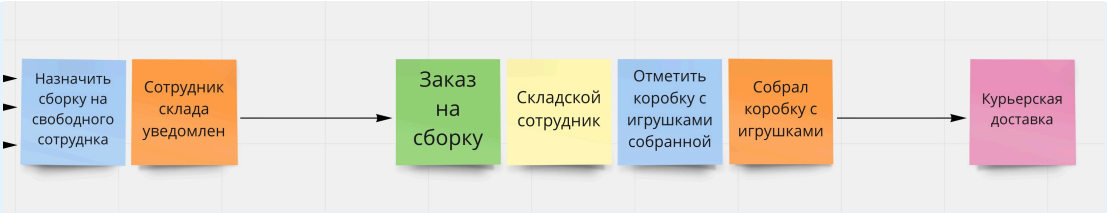
📌 **Нюансы**

1. **Нет правильного способа группировки команд и событий по контекстам.** Например, мы стараемся группировать по логике вокруг процесса. То есть для успешного мэтчинга игрушек и клиентов нужно добавить критерии и рассчитать характеристики игрушки после тестирования — и это будет один контекст. Возможно, подобная группировка окажется некорректной, но узнаем мы об этом уже в следующих уроках.



Посмотреть изображение поближе

2. Иногда помогает **указать стрелкой, какое именно событие станет read model для актора.** Это хорошо работает, когда модель на чтение состоит из большого количества событий и важно не потерять все эти события из виду.

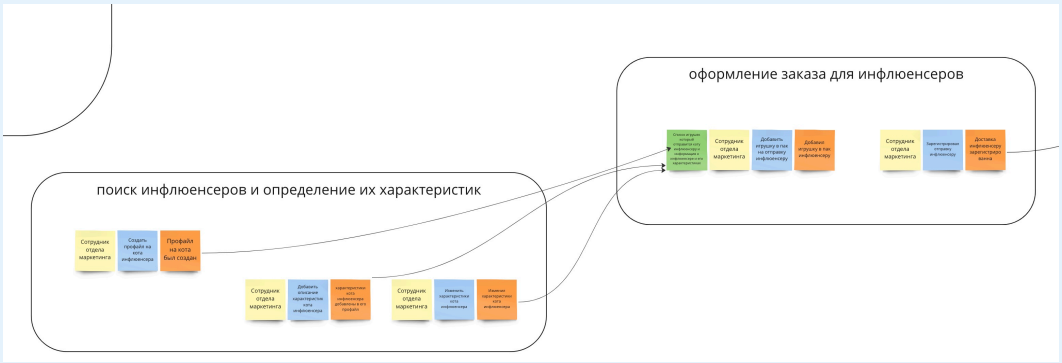


В этом примере событие «сотрудник склада уведомлён» содержит информацию, без которой сотрудник склада не сможет отметить коробку с игрушками, собрана или нет.

Посмотреть изображение поближе

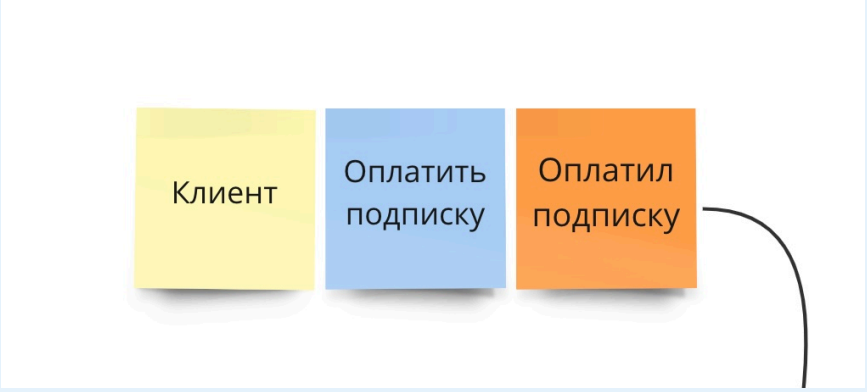
3. Чтобы не перегружать модель, можно **пропускать очевидные read models, потому что они не говорят ничего неочевидного, а только усложняют модель.** И указывать read models только там, где они неочевидны и собираются из событий, связанных с другим контекстом.

Например, для корректной работы оформления заказа нужно знать информацию о регистрации кота в системе и его характеристиках.



Тут важно указать, из каких событий состоит список игрушек для кота-инфлюенсера на отправку. А благодаря этой информации можно собрать игрушки в пак коту.

Посмотреть изображение поближе



А тут данные не так критичны, чтобы ими захламлять модель.
Посмотреть изображение поближе

- [x] ~~Шаг 1. Поиск всех событий в системе~~
- [x] ~~Шаг 2. Добавление команд, акторов и policy~~
- [x] ~~Шаг 3. Поиск контекста и данных для отображения~~

Общие советы по всему процессу

1. **Ищите связь между бизнес-командами.** Факт наличия связи важнее вида этой связи. Например, ES помогает проиллюстрировать, что происходит после продления подписки: система связывает игрушки из базы с предпочтениями клиентского кота и получает коробку с этой игрушкой. При этом с технической точки зрения не важно, как именно это происходит: через HTTP API, rpc-вызов или отправку сообщения через брокер.

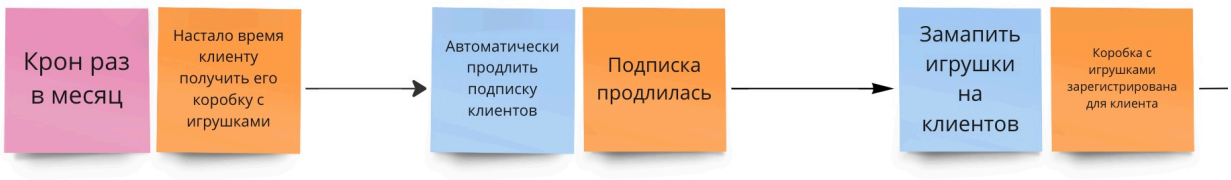
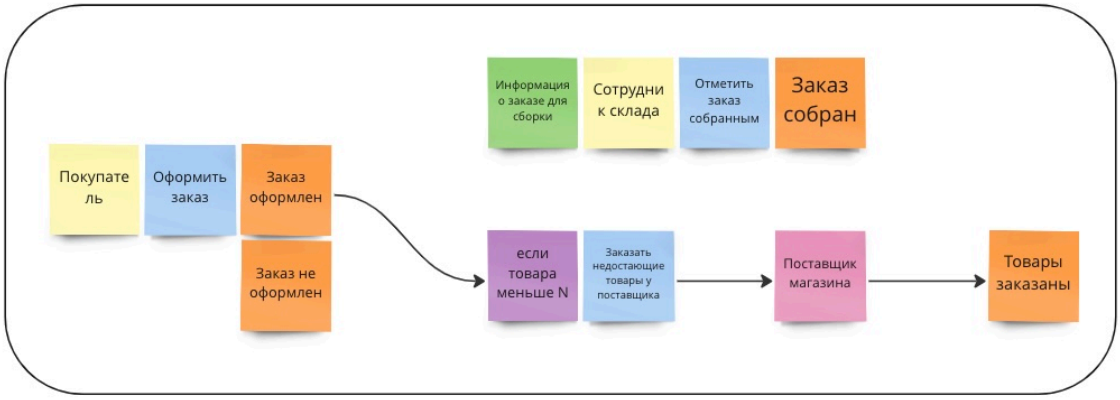


Иллюстрация того, что происходит в системе после продления подписки.
Посмотреть изображение поближе

2. **Не закапывайтесь в техническую реализацию.** Если заметили, что в момент шторминга начали обсуждать, как писать код, прекратите этим

заниматься, запишите контекст обсуждения и договоритесь на специальную встречу после.

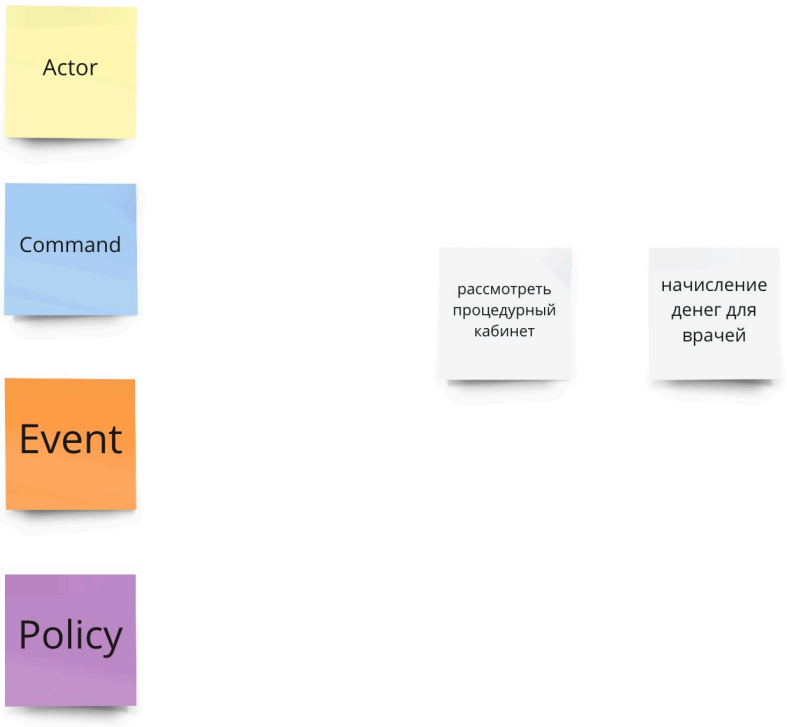
3. **Сделайте и держите под рукой легенду,** которая будет объяснять, что значит зелёный стикер, и поможет не отвлекаться от процесса.



Базовая легенда, которую я всегда добавляю к любой ES-модели.
Посмотреть изображение поближе

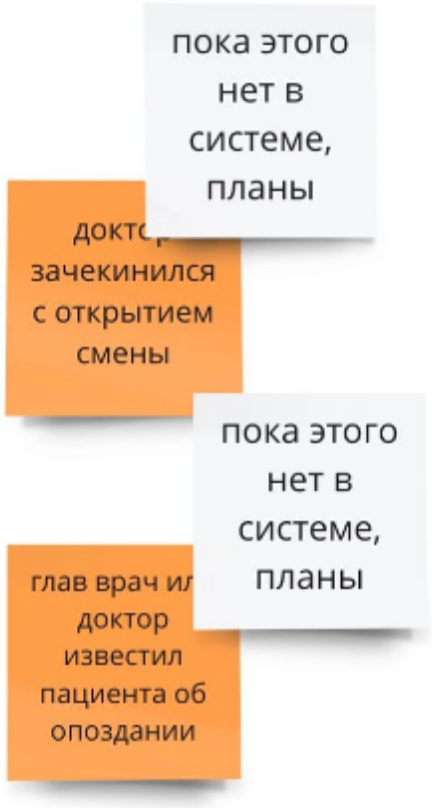
4. **Оставляйте комментарии.** Комментарии важны, когда вы ставите задачи на будущее и когда объясняете контекст.

- **Задачи на будущее.** Это любая часть работы, которая не нужна прямо сейчас, но важна в будущем. Иногда это места, которые пока непонятны и которые надо будет разобрать в будущем.



Пример TODO-стикеров из реальной жизни.
Посмотреть изображение поближе

- **Объяснение контекста.** Это полезно для людей и команд, которые пока не существуют, но появятся в ближайшем будущем.



Пока этих команд нет, но менеджмент хочет видеть логику в будущем.
Посмотреть изображение поближе

5. **Сохраняйте полученный артефакт**. Если вы проводите ES в реальной жизни, сфотографируйте полученную модель, чтобы было видно подпись на каждом стикере, загрузите её с временной меткой о проведении (и списком присутствующих) и дайте доступ всем в компании. Если работали в Miro, убедитесь, что ссылка на модель находится в документации, а доступ к модели может получить любой сотрудник компании.

Напомним, что в курсе по асинхронной архитектуре (AA) был онлайн-факультатив, где мы разбирали систему на ES. Посмотрите, если хотите разобраться подробнее, как это происходит.

2.2. Моделируем данные

Система работает не только с поведением, но и с данными. Если это интернет-магазин, то его данные — это товары и их количество. Если блог — посты, комментарии и пользователи. При создании структуры системы важно учитывать, из каких данных состоит проект и как эти данные между собой связаны.

Причем, выделяют три перспективы моделирования данных: концептуальную, логическую и техническую. Нас интересует концептуальная

перспектива, которая говорит о том, какие данные используются системой и как данные связаны между собой. Т.е. не то, как выглядит схема базы данных (это логическая перспектива), а то, что необходимо бизнесу из данных для работы, без привязки к конкретной реализации хранения данных в конкретной бд.

Для моделирования можно взять диаграмму из AA-курса, structure diagrams из UML или самый примитивный вариант из квадратов и стрелок. Ибрагим решил не заморачиваться и взять модель из курса.

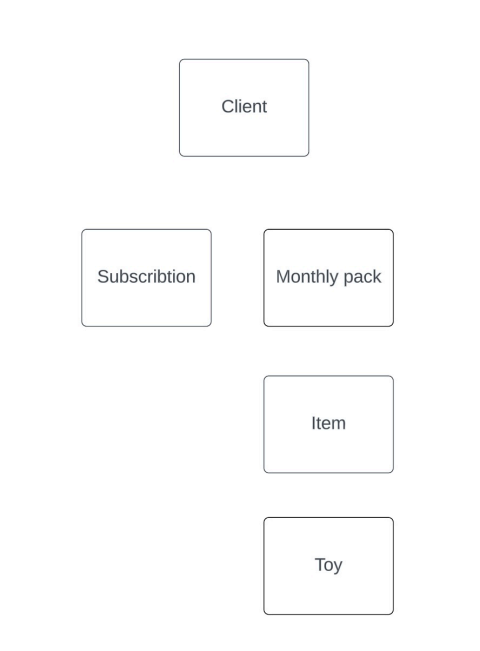
📎 **Data model 101**

101 в названии — референс на интродакшен-курсы в MIT или Стэнфорде.

▼ **Основные принципы и подходы**

Давайте разберёмся, как создаётся модель данных, на основе базовой логики магазина и менеджмента подписок Happy Cat Box.

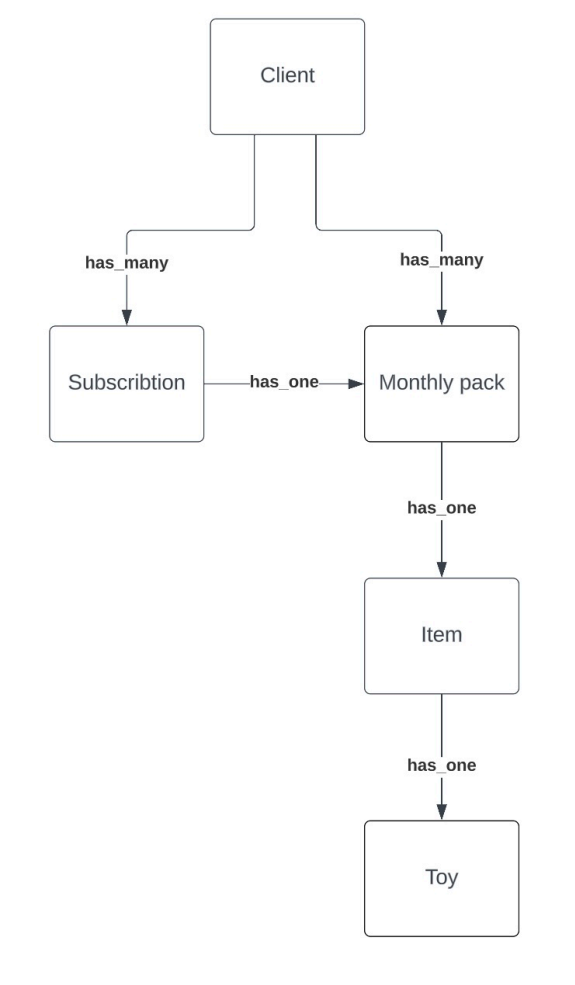
Начнём с определения объектов, с которыми работает система. Я говорю *«объекты»,* хотя на деле это могут быть сущности, энтити, структуры или даже целые агрегаты. Название вторично, главное — определиться, из каких основных элементов всё состоит. Базовая логика выглядит так ↓



Посмотреть изображение поближе

Клиент пользуется магазином (Client). Он подписывается на услугу (Subscription), которая ежемесячно создаёт набор игрушек (Monthly pack), состоящий из айтемов (Item), каждый из которых — игрушка (Toy). Так мы получили пять квадратов с основными объектами.

Объекты чем-то похожи на набор моделей из MVC-паттерна, но отличаются тем, что это просто набор высокоуровневых данных, которые говорят о бизнес-логике, а не о структуре БД. Едем дальше.



Посмотреть изображение поближе

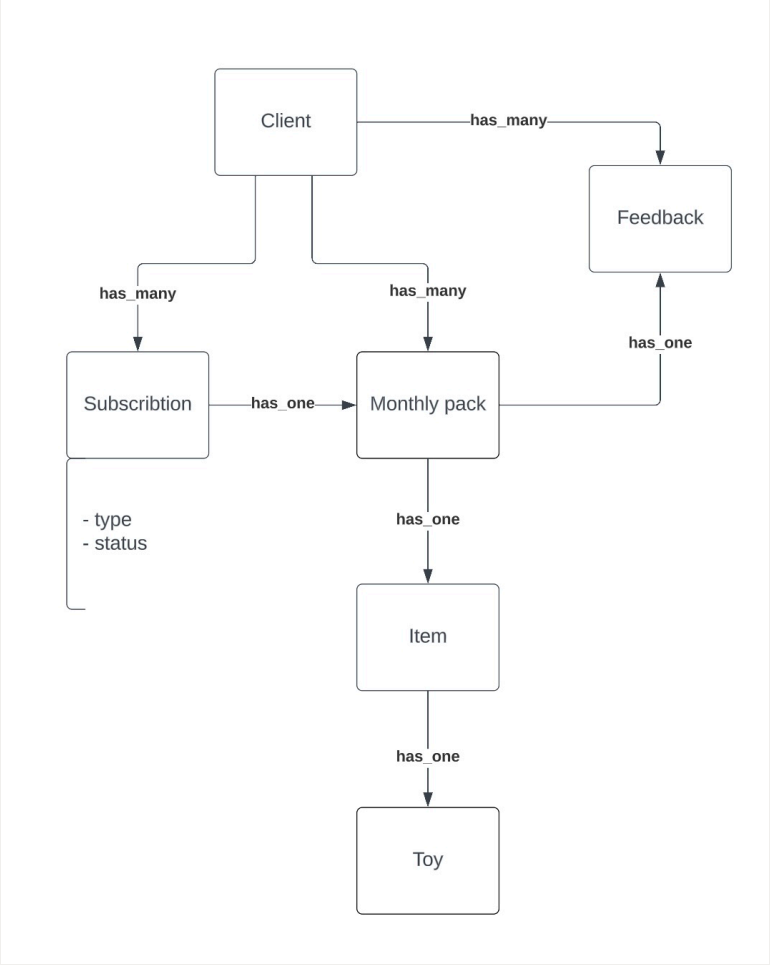
После определения основных блоков мы можем разобраться со связями между ними.
Для этого я использую стрелки, направления которых показывают, какой из объектов чем владеет. Например, у клиента есть несколько подписок, а не у подписок есть клиенты. Клиенты владеют подписками.
Также я явно указываю вид связи: **has_one** — это обозначение одного к одному, **has_many** — одного ко многим. Например, в

магазине я могу взять один товар, а могу десять, которые могу нести в руках, а могу положить в корзину. В этом случае корзина с товарами — has_many: одна корзина и много товаров внутри. Благодаря этому на диаграмме видно, что подписка связывается с одним ежемесячным паком игрушек, который состоит из набора айтемов, которые по сути игрушки.

Так как объект — это набор полей, то важные для бизнес-логики поля я показываю отдельно. Для этого использую два способа, которые зависят от важности поля: **отдельный квадрат** и **список необходимых полей.**

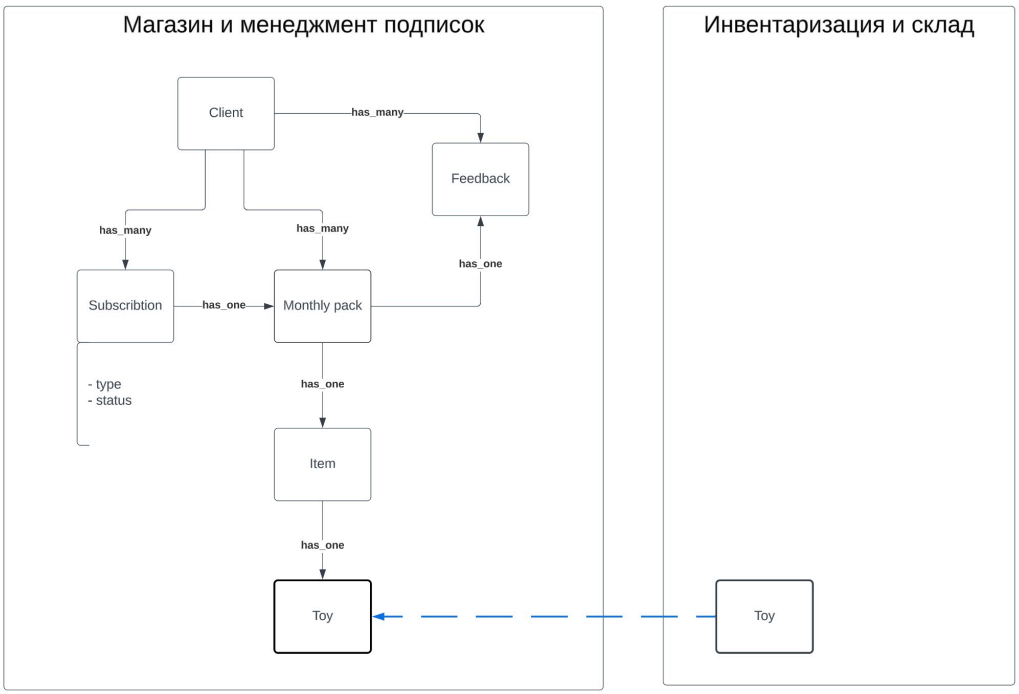
Отдельные квадраты я использую для обозначения статусов, аудита логов, тегов и другой информации, которая в теории может оказаться в одной таблице с основным объектом, но его стоит выделить отдельно из-за связанности с несколькими объектами.

Список необходимых полей я использую для общего описания структуры объекта, если нужно указать наличие полей. В примере с магазином фидбэк по ежемесячному паку может быть как отдельным объектом, так и частью объекта Monthly pack.



Статус и тип подписки — важные поля для Subscription объекта, которые я отобразил списком. А фидбэк, который может оказаться просто text-полем в таблице Monthly pack, вынес как отдельный квадрат.
Посмотреть изображение поближе

При выборе любого варианта важно, чтобы фидбэк был указан как относящийся к ежемесячному паку. Это важная часть данных, необходимая для бизнес-логики, поэтому я вынес его в отдельный квадрат. В случае с подпиской тип и статус подписки — это важная часть реализации логики подписки, но связанной только с подпиской. Поэтому в этом случае я использовал список необходимых полей.



Посмотреть изображение поближе

Иногда в разных частях проекта нужны одни и те же объекты, но они могут содержать разные поля. Такие объекты стоит делать заметнее, например увеличить размер окантовки или отметить их другим цветом.
Пример. Я отправляю Ибрагиму бумажное письмо. Письмо — объект, но для каждого участника переписки в нём важны какие-то свои поля или значения. Для меня важно, чтобы бумага была приятной на ощупь, а для Ибрагима — читаемость и сохранность письма. В примере курса аналогичная логика. Для склада *toy* — это объект реальности, который лежит на полке и имеет свои поля и срок хранения. Для магазина этот же объект *toy* важен как часть продажи — со своим описанием, цветом и картинкой.

Когда создаёте модель, используйте толстую стрелку, направленность которой говорит о том, где мутируются данные, а где они только читаются.

Возвращаясь к примеру с магазином и складом: нам нужна информация об игрушках, которая находится в части

инвентаризации и склада. При этом склад — оунер информации об игрушке, потому что он создаёт и мутирует данные. Следовательно, стрелка идёт от склада к магазину.
В итоге мы получаем модель, в которой есть вся информация о данных, необходимых для работы проекта, связях между ними и том, как каждый из кусков проекта взаимодействует с данными.

Чтобы сделать модель данных из квадратов и стрелок (на самом деле любую), Ибрагиму нужно пройти три шага:

- [ ] Шаг 1. Перенести элементы из ES
- [ ] Шаг 2. Определить данные для каждого из элементов
- [ ] Шаг 3. Найти связи с владельцами этих данных

Заумь от Антона о настоящем шедевре Фаулера, о котором все молчат

▼ Необязательный контент для большего погружения в тему

Мартин Фаулер в 1996 году пытался решить проблему описания структуры данных для различных «популярных» доменов, например биржи, аккаунтинга и так далее. Для этого он написал книгу Analysis Patterns, Reusable Object Models, в которой описал возможные варианты реализации доменов на своём собственном modelling language.

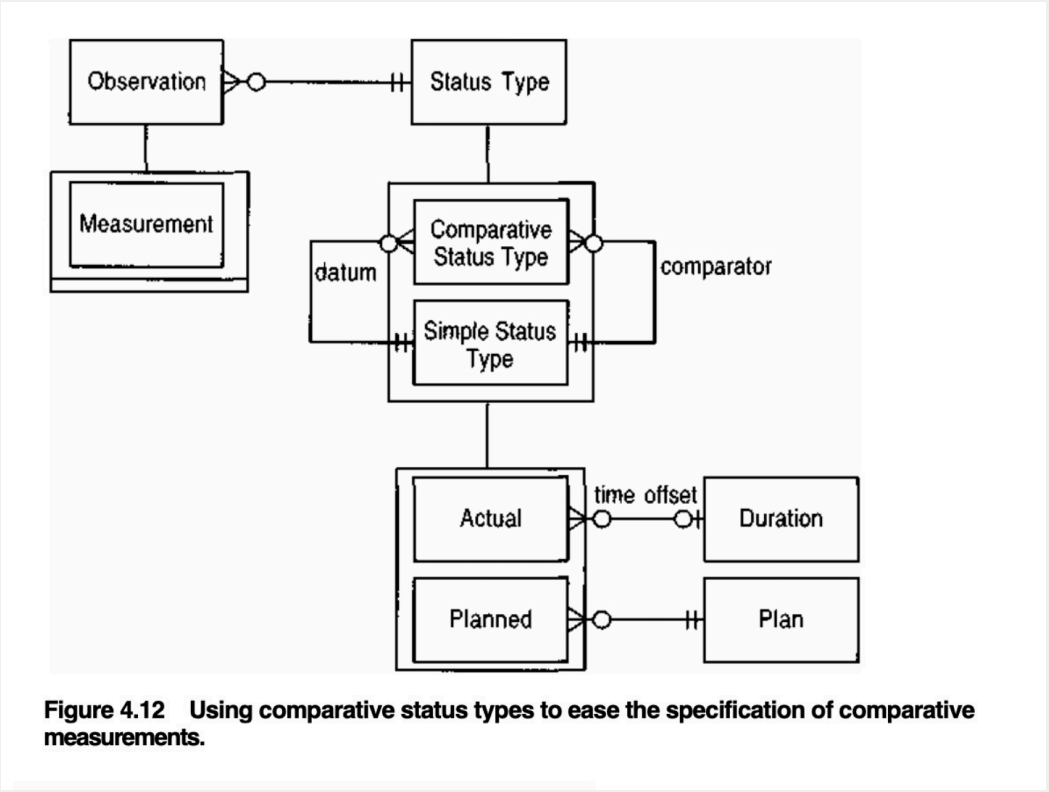


Figure 4.12 Using comparative status types to ease the specification of comparative measurements.

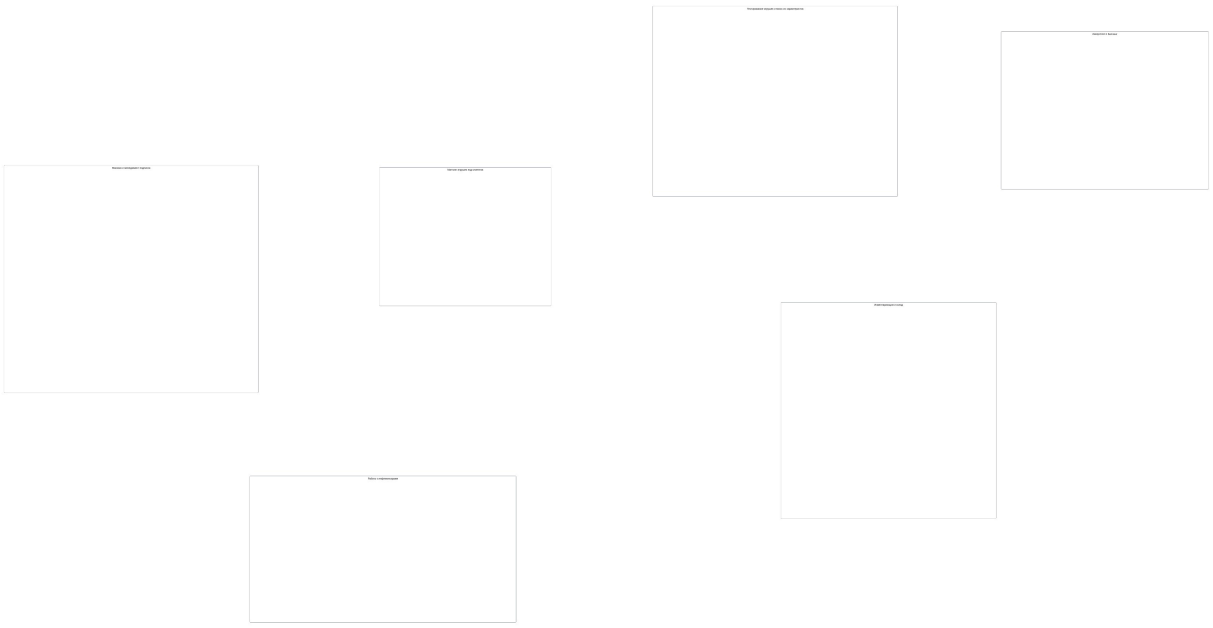
Пример modelling language, который разработал Фаулер для своей книги.
Посмотреть изображение поближе

К сожалению, о книге мало кто знает, но она помогла мне два раза. В первый раз — когда я столкнулся с реализацией биржи, а второй — когда надо было выбрать оптимальную структуру для аккаунтинга.

Шаг 1. Перенос элементов из ES-модели

Для этого нужно взять все определённые в процессе штурма элементы и добавить их в черновик будущей модели данных. Здесь нет строгого инструментария: выбирайте любой инструмент, который вам подходит. Например, Lucid Chart, Diagrams.net, mermaid, лист бумаги, Miro или обычный вайтборд на стене.

Вот что получилось у Ибрагима.



Берём контексты из ES и просто делаем их пустыми квадратами, которые начнём заполнять в следующем шаге.
Посмотреть изображение поближе

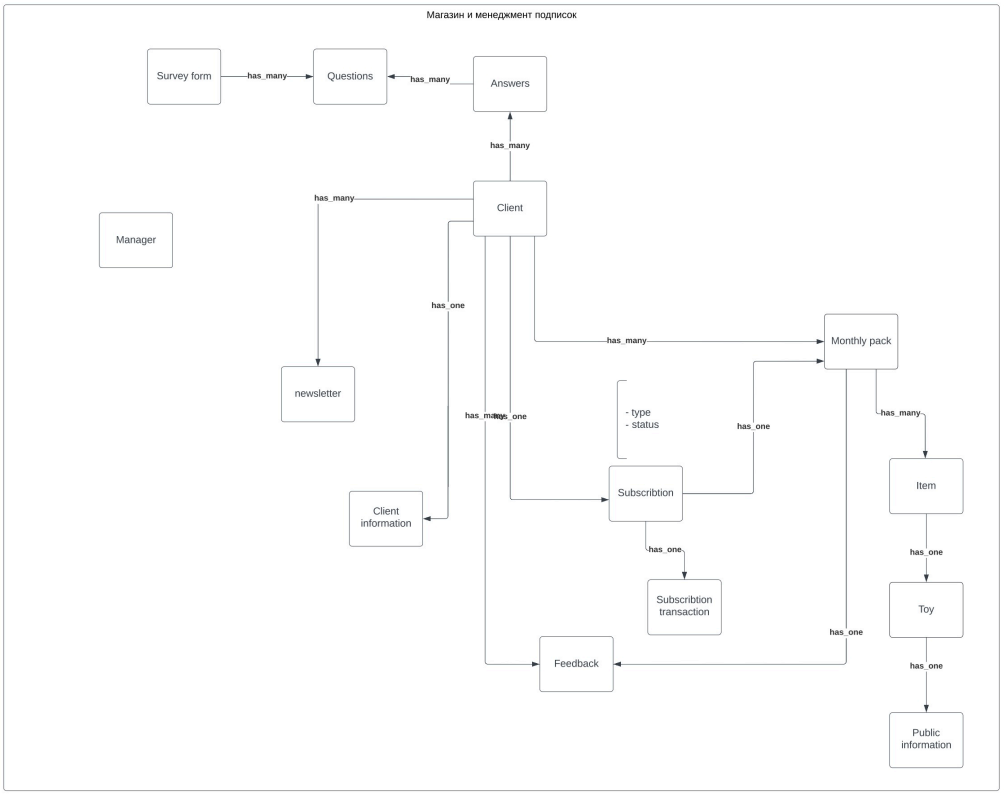
📌 **Нюансы**
В этом шаге нужно переносить только сгруппированные области. Если переносить связи, которые мы получили в процессе ES, то получим перегруженную модель, с которой будет невозможно работать.

- [x] ~~Шаг 1. Перенести элементы из ES~~
- [ ] Шаг 2. Определить данные для каждого из элементов
- [ ] Шаг 3. Найти связи с владельцами этих данных

Шаг 2. Определение данных для каждого элемента

Для каждого элемента нужно указать необходимые данные. Если в разных контекстах используются одинаковые данные (клиенты или игрушки), то их нужно продублировать.

Ниже модель данных для контекста магазина, которую сделал Ибрагим.



Для остальных контекстов из ES он сделал аналогичную работу.
Посмотреть изображение поближе

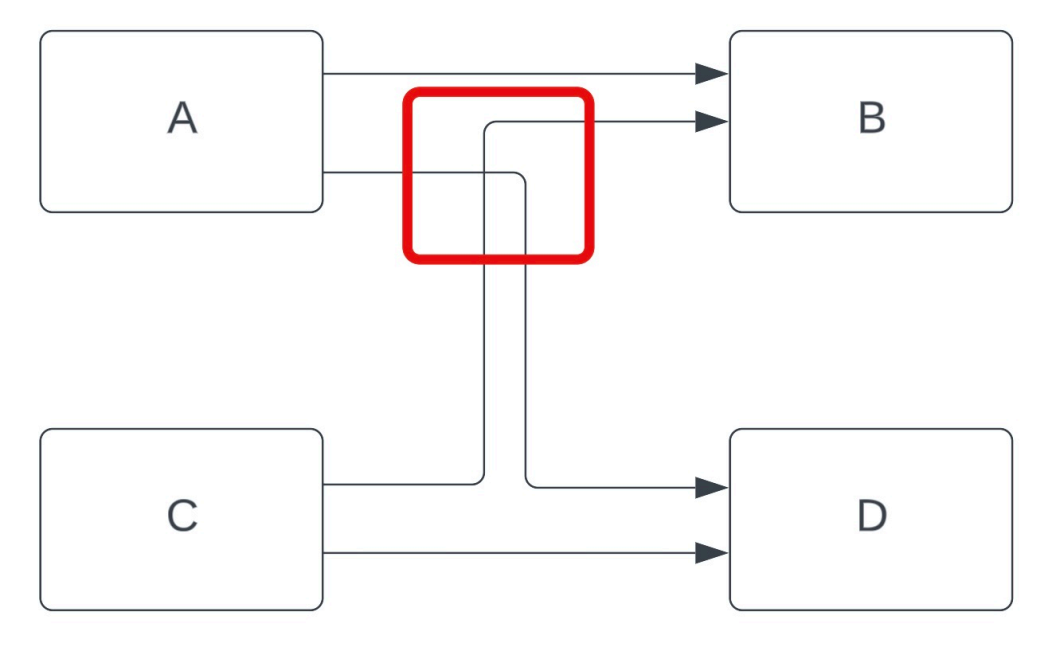
Обратите внимание — в магазине нужны менеджеры, которые также необходимы для определения характеристик кота.

📌 **Нюансы**

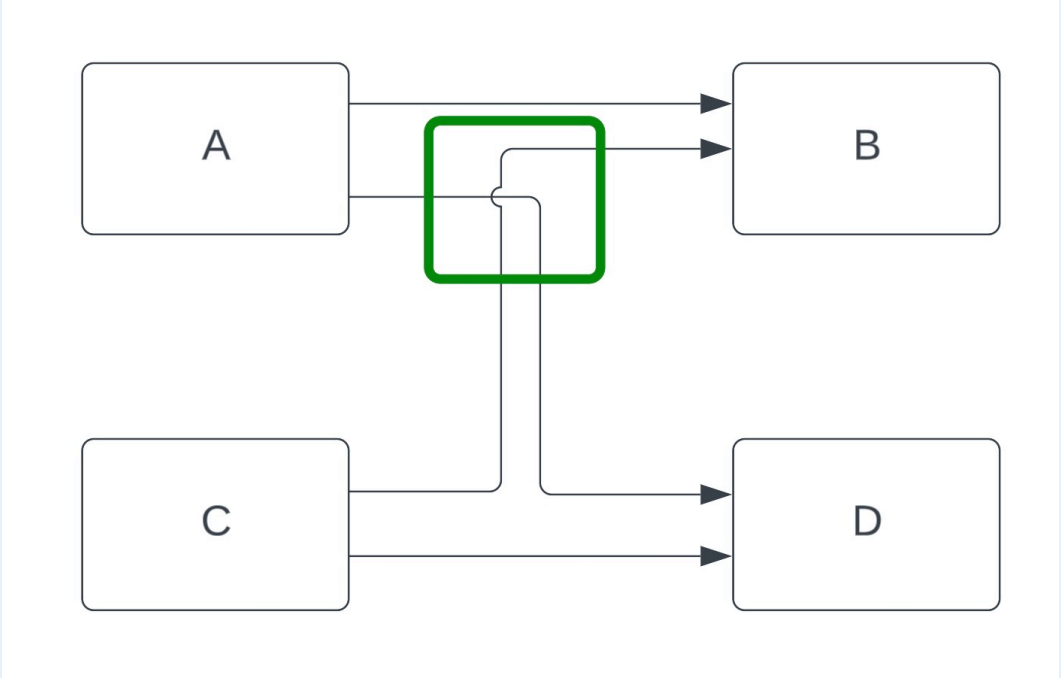
1. **Дублирование квадратов понятнее нагромождения стрелок.** Если у вас есть условный item в трёх контекстах, то сделайте нужное количество квадратов с item, потому что в «спагетти» из стрелок можно потерять важную связь данных между контекстами.

2. Важно отметить, что ордер на складе, ордер в магазине и гифт-бокс в маркетинге — это разные ордера и они обозначают разные вещи. То есть **мы не можем позволить себе использовать ордер как название для всего, потому что обозначает разное и использует свой контекст.** При этом эти ордера не связаны между собой на уровне данных, они связаны тем, что доменное событие создаёт ордер в другом месте.

3. **Не пересекайте линии между собой, это усложняет чтение модели.** Если без пересечений не обойтись, сделайте явное наложение линий.



Плохо: линии не явно пересекаются, на большом количестве пересечений будет сложно понять, куда каждая из линий ведёт.



Хорошо: явно видно, что линии пересекаются перпендикулярно между собой. Направление линии считывается явно, это уменьшает когнитивную нагрузку на изучение модели.

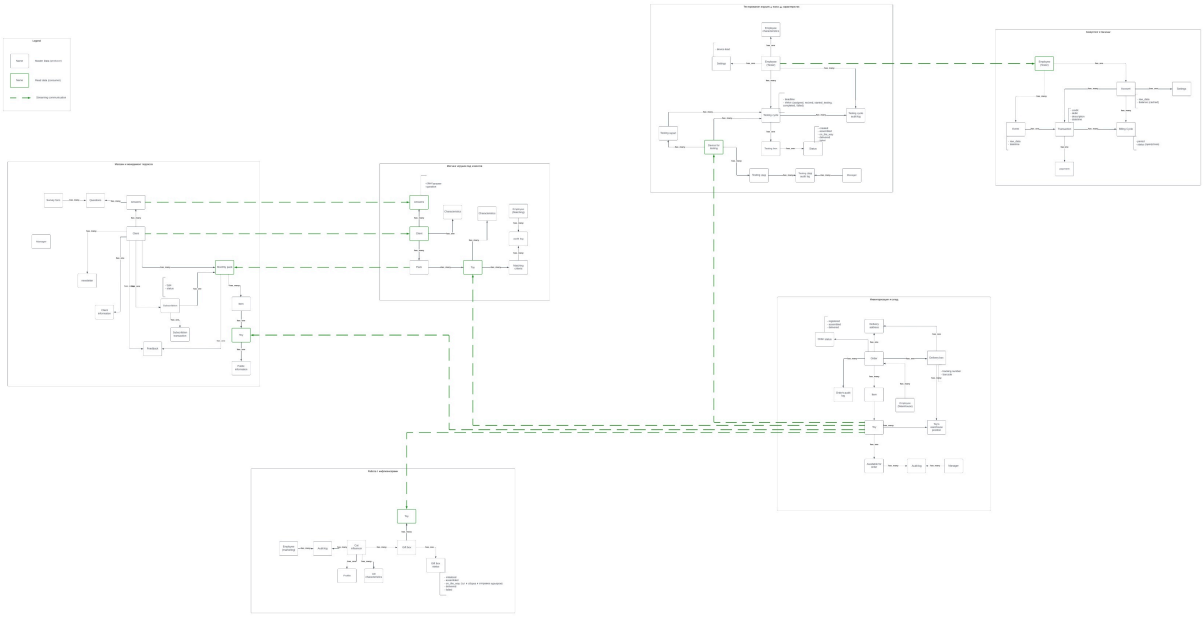
- [x] ~~Шаг 1. Перенести элементы из ES~~
- [x] ~~Шаг 2. Определить данные для каждого из элементов~~
- [ ] Шаг 3. Найти связи с владельцами этих данных

Шаг 3. Определение связей по данным между элементами и поиск владельцев данных

Контексты из первого шага зависят от одинаковых данных, расположенных в других контекстах. Например, информация о кошачьих игрушках нужна в сборке и в четырёх других контекстах. Красьте подобные данные в отдельный цвет, изучайте, кто создаёт и мутирует данные (это владелец данных), а кому данные нужны только для чтения (это потребитель). После чего покажите стрелками, кто и откуда получает эти данные (owner → consumer).

Владелец данных — это элемент, который создаёт, редактирует или удаляет данные. Владелец определяется по командам, описанным в ES-модели.

Ниже финальный результат модели данных Ибрагима ↓



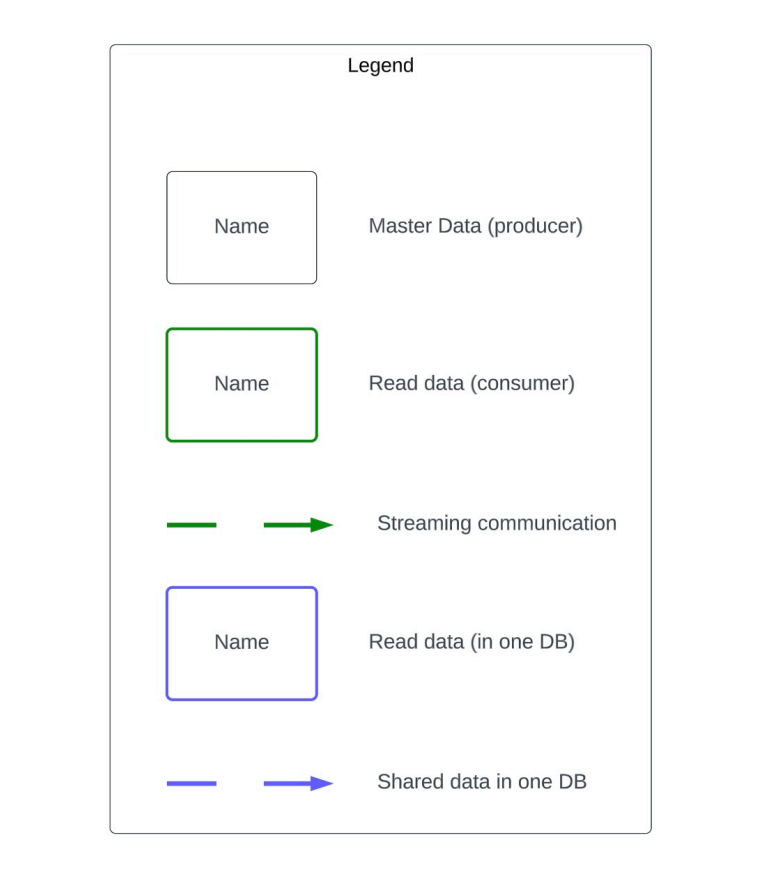
Итоговая data model для happy cat box.

System Analysis course_ data model.pdf

Скачать изображение в ПДФ

📌 **Нюансы**

1. **Если пользуетесь разными видами стрелок — опишите их виды в легенде.** Если указываете синхронные и асинхронные коммуникации, задействуйте сплошную для синхронных и пунктирную для асинхронных коммуникаций.



Пример легенды для модели данных. Без неё через месяц будет трудно понять, что именно обозначает каждый из квадратов или стрелок.
Посмотреть изображение поближе

2. **Если у вас есть user и один контекст владеет основными данными** (имя, пароль, логин), **а другой — специфичными** (локация, биллинг-информация и так далее), **то это нужно указать отдельно.** Это нормальная ситуация, которая технически решается двумя стриминг-событиями вокруг одного агрегата с разными полями.

3. Чтобы не переусложнить систему, **не тащите в модель абсолютно**

все данные. Используйте только те, на которые завязаны бизнес-процессы. Если необходимо отобразить техническую информацию — сделайте отдельный слой или модель, если выбранный инструмент не поддерживает слои.

- [x] ~~Шаг 1. Перенести элементы из ES~~
- [x] ~~Шаг 2. Определить данные для каждого из элементов~~
- [x] ~~Шаг 3. Найти связи с владельцами этих данных~~

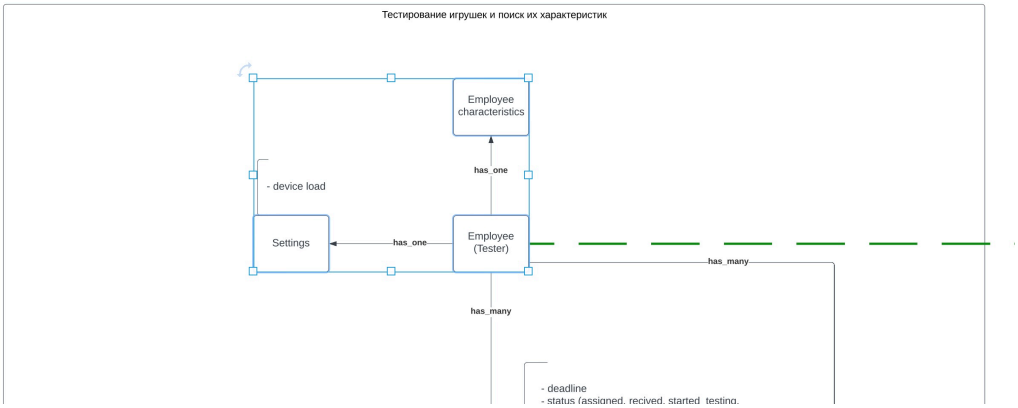
Общие советы по всему процессу

1. **Используйте обратный подход в создании модели.** Это когда вначале собираются все данные и только потом делятся по контекстам. Такой вариант работает, когда вначале собирается модель данных, а после ES-модель. У подхода есть и минус — когда в системе много абстракций, связанных между собой, данные сложно разделить.
2. Если **обнаруживаете между двумя условными контекстами много связей**, то подумайте, точно ли контексты были выбраны правильным образом.
3. **Явно что-то идёт не так, когда существует несколько оунеров данных.** Например, в случае Happy Cat Box несколько оунеров — это магазин и склад, которые потенциально могут добавлять игрушки в систему. Подобные ситуации могут привести к проблемам, связанным с транзакционностью и консистентностью, особенно в случае, если сервисы не зависят друг от друга. Если появляются вопросы в духе *«как мне сделать сагу для двух сервисов»,* это точно оно.

   Возможных решений два:

   - **сделать оунером один элемент** (склад в нашем случае) и бить по рукам каждому менеджеру, который захочет добавлять или менять информацию об игрушках в магазине;
   - если оставить одного оунера данных нельзя, то придётся либо жить с проблемами, либо **возвращаться к ES-модели и переделывать группы процессов, объединяя их или выделяя новые.**

4. **Введите легенду с обозначениями каждой стрелки и цвета,** чтобы не запутаться среди всего этого разнообразия.
5. Если вам **досталась легаси-система без людей,** которые могут помочь разобраться в деталях проекта, то придётся самостоятельно разбираться со схемой базы данных, UI и прочими штуками**.** Если так случилось, то вот источники, откуда можно достать информацию о необходимых данных:
    - **требования**
      Пример из Требований к системе: *«Любой кот-тестировщик может зарегистрироваться в системе, для чего он заполняет информацию о себе, о поведении и способе выплаты его гонорара».*
      Из требования выше можно определить, что у нас есть кот-тестировщик *(Employee (Tester))*, информация о его поведении *(Employee characteristics)* и предпочтительном способе выплаты гонорара, которая уйдёт в другой элемент вместе с информацией о коте.

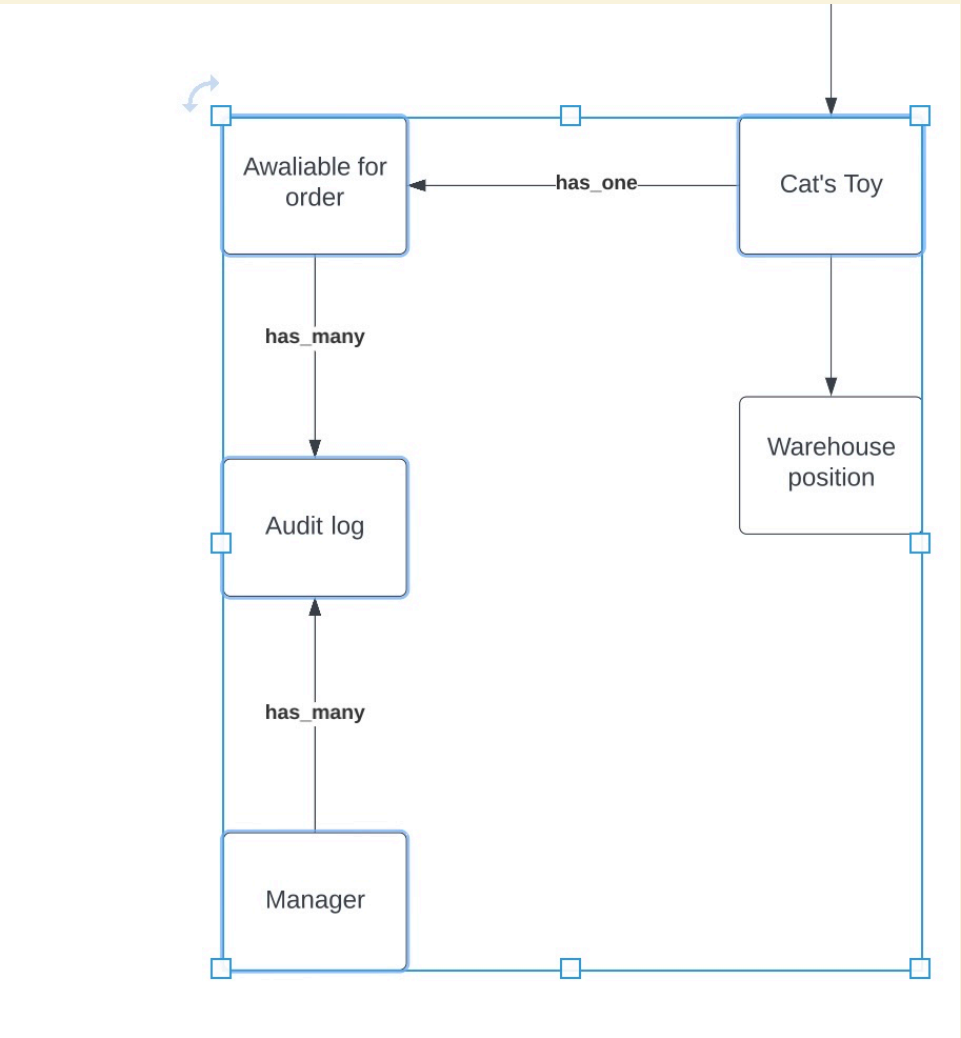


Посмотреть изображение поближе

- **UX/UI.** Тут информация о том, что надо показывать пользователю. Аналогично помогут все найденные read models из Event Storming.
- **существующее приложение.** Откройте исходный код, желательно схему БД. После этого попробуйте разобраться в структуре и переведите её в модель данных. Из минусов: здесь могут быть проблемы, когда одно приложение использует две отличающиеся схемы данных.

- **доменные концепции.** Обратитесь к стандартам. Например, если вы работаете с медициной — поможет fhir, с аккаунтингом — investopedia.

👉 **Допущения в модели данных**
Для бизнес-логики важен статус, поэтому его можно сделать отдельным квадратом. Это покажет связь с аудитлогом и важность для бизнес-логики. Хотя в реализации всё будет в одной таблице ордеров, просто как энам.



Посмотреть изображение поближе

2.3. Собираем всё вместе

Итак, у Ибрагима теперь есть две модели: **Event Storming** и **Модель данных.** Одна описывает структуру, другая — поведение.

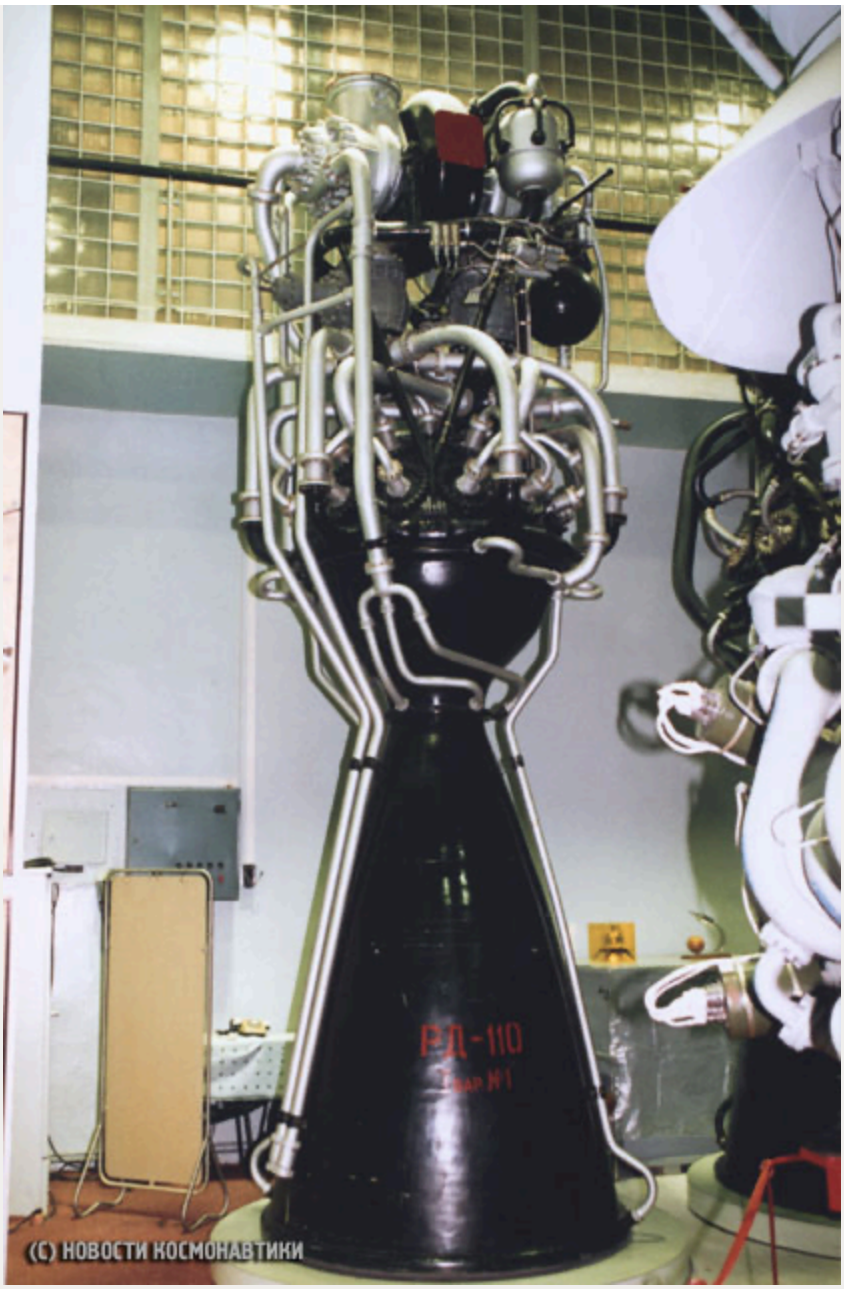
Чтобы получить общую систему, нужно соединить эти две системы.

Заумь от Антона, в которой рассказывается о системной инженерии и при чём тут функция и форма системы

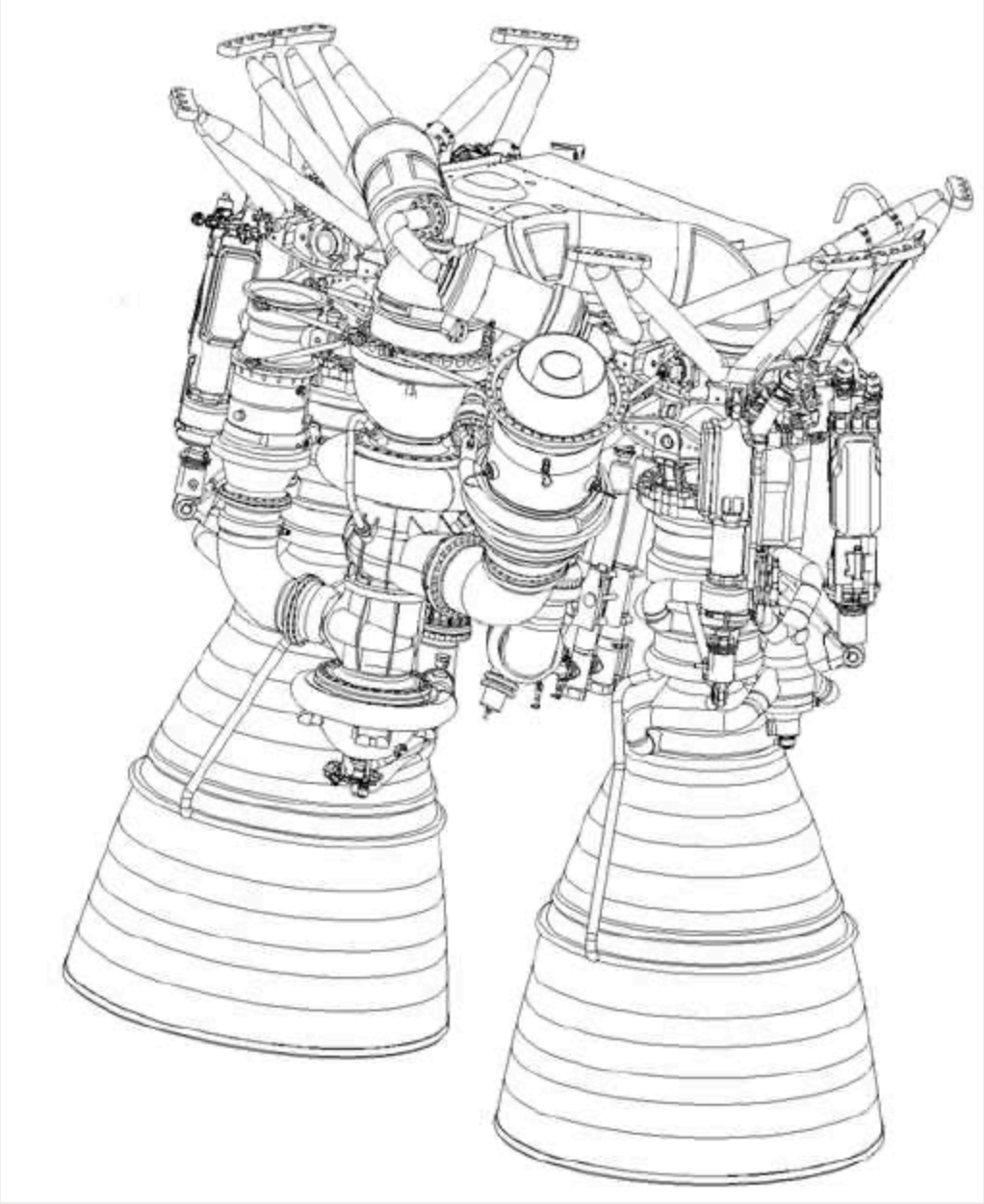
▼ Необязательный контент для большего погружения в тему

Всю историю люди создают системы. Это могут быть как организации людей, так и новые инструменты, софт, здания и так далее. С каждым годом системы становятся всё сложнее и сложнее.

Мне нравится пример про жидкостные двигатели для космических ракет. Для сравнения — есть два двигателя: РД-110 и РД-180. Каждый из двигателей — система, которая за 50 лет производства усложнилась в разы. В качестве примера можно посмотреть на картинки или почитать описание двигателей в источниках.



РД-110. Использовался для ракеты Р-3, разрабатывался в 1947 году.
Источник
Посмотреть изображение поближе



РД-180, используется в ракетах «Атлас», разрабатывался в середине 90-х.
Источник
Посмотреть изображение поближе

Такие системы не то что сложно уместить в голове, их даже спроектировать сложно. Аналогичная ситуация работает для проектирования автомобилей, зданий и любых других систем, где задействованы сотни и тысячи людей. Но мы, люди, научились производить сложные системы с помощью специальных подходов. Для этого человечество придумало

system engineering, цель которого — помочь инженерам в работе со сложными системами.
В system engineering вводится понятие системы и рассматриваются способы изучения и работы с системой.

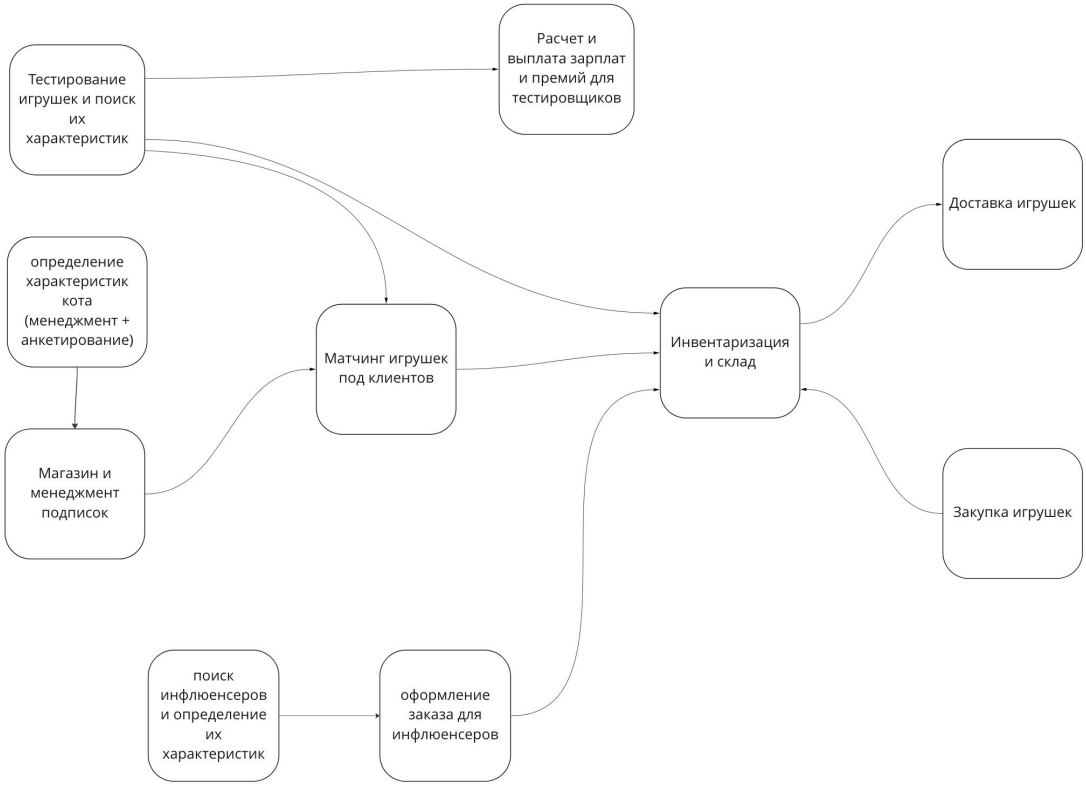
> **Система** — это набор элементов (entity) и их взаимосвязей. Функциональность системы превышает сумму отдельных энтити. Система имеет форму (System Form) и функцию (System Function) и состоит из свойств (system properties). Форма — это то, чем система является, а функция — то, что она делает.

Если говорить о software-системах, то под элементами подразумеваются модули, сервисы, приложения, классы и функции. Форма — набор данных, структура кода, описание интерфейсов. Функция — поведение системы или отдельных её частей, а свойства — характеристики, например перформанс.

Связи между ними — это каплинг, о котором мы будем говорить во втором уроке. При этом связи между элементами могут быть как формальные (общие данные между), так и функциональные (когда один процесс запускает другой).

То есть, чтобы найти все связи между элементами, нам надо найти оба вида связей: формальные и функциональные. Поиск функциональных связей и элементов системы перекладывается на event storming в нашем случае, а поиск формальных — на модель данных. А объединив две модели в одну новую, получим модель системы.

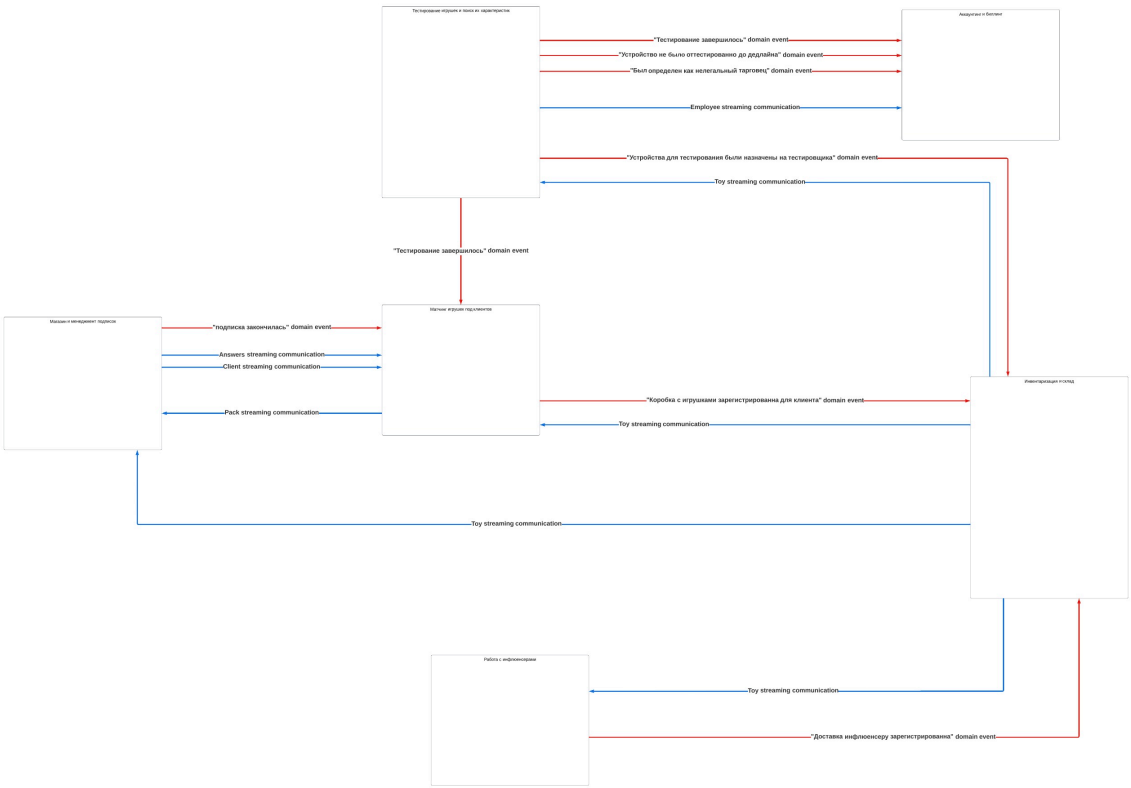
Если упростить ES-модель и оставить только контексты и связи между контекстами, основанными на поведении, то получим что-то такое:



Выкинув все команды и оставив только группировку по функционалу со связями, мы получим структуру, которая описывает, как связаны разные части поведения Happy Cat Box.

Посмотреть изображение поближе

Теперь нам осталось добавить связи между контекстами, основанными на модели данных. После этого — собрать все связи и проанализировать полученную структуру системы. Для этого Ибрагим сделал новую модель, состоящую из контекстов, полученных в ES-модели, добавил в неё бизнес-события из ES и связи по данным из модели данных. Каждую стрелку покрасил своим цветом и подписал, какие события и какие данные связывают полученные контексты.



Итоговая модель элементов и всех видов связанности между ними. Бизнес-события — красные, связи по данным — синие.

System Analysis course_ context view.pdf

Скачать изображение в ПДФ

С помощью этой модели Ибрагим сможет проанализировать, из каких абстрактных элементов состоит сервис Happy Cat Box и какие связи проходят между этими элементами.

> 🎉🎉🎉 **Второй пункт готов!** 🎉🎉🎉
>
> Настал лучший момент отдохнуть и разгрузить голову. Например, мне помогают записи из кабины поездов. Посмотрите, как прекрасна Япония весной.

🗓️ План работы над happy cat box

1. ~~Определить, что от нас хотят, и изучить требования~~
2. ~~Решить, из каких компонентов будет состоять система~~
3. Выбрать гипотетическую структуру проекта
4. Подготовить описание решения для разработчиков
5. Передать решение в разработку

На этом месте мы решили сделать паузу, чтобы дозировать большой рассказ. Отдохните, переварите прочитанное и потом возвращайтесь ко второй части истории.

🛋️ **На этом ознакомительный фрагмент заканчивается**

Если хотите присоединиться к курсу — купить доступ можно здесь →

❓ Если не нашли ответа на свой вопрос или запутались, пишите на почту support@tough-dev.school

---

Правила распространения контента

По умолчанию контентом курса — презентациями, конспектами и видео — делиться нельзя. Не отправляйте, пожалуйста, ссылки на видео и презентации коллегам. Вы можете пересказывать идеи и модели, ссылаясь на Школу.

Берите отдельные куски, не более 10% от конспекта или видео. Можете делать скриншоты уроков и давать ссылку на курс.