Принципы SOLID

- **The Single Responsibility Principle** (SRP) – Принцип единственной ответственности
- **The Open Closed Principle** (OCP) – Принцип открытости/закрытости
- **The Liskov Substitution Principle** (LSP) – Принцип подстановки Барбары Лисков
- **The Interface Segregation Principle** (ISP) – Принцип разделения интерфейса
- **The Dependency Inversion Principle** (DIP) – Принцип инверсии зависимостей

Принцип единственной ответственности (SRP)

- Каждый класс выполняет лишь одну задачу
- ШП Функциональный дизайн
- Антипаттерн «Божественный объект»
- Способы достижимости:
  - использование приёма рефакторинга «выделение класса»;
  - шаблон «фасад»;
  - интерфейсы.
- Примеры:
  - Система управления поливом и клапан (из первого примера ООП);
  - Ведение боя и правила боя (из второго примера ООП).

Принцип открытости/закрытости (OCP)

- «Программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения»
- Специфика:
  - близок к SRP;
  - принцип позволяет избежать пересмотра связанного кода, модульных тестов, текстов самодокументирования и других артефактов ПО;
  - добиться снижения трудозатрат.
- Способы достижимости:
  - те же, что для SRP

Принцип подстановки Барбары Лисков (LSP)

- «Объекты в программе должны быть заменяемыми на экземпляры их подтипов без изменения правильности выполнения программы». Наследующий класс должен дополнять, а не изменять базовый
- Следствия:
  - нельзя усиливать предусловия и ослаблять постусловия методов производных классов;
  - нельзя создавать новых мутаторов свойств, не предусмотренных базовым классом;
  - производные классы не должны бросать исключения, не предусмотренные в базовом классе.
- Примеры:
  - ?

Принцип разделения интерфейса (ISP)

- «Много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения»
  - или «Программные сущности не должны зависеть от методов, которые они не используют»
- Особенности:
  - улучшает адаптивность кода;
  - существенно упрощает рефакторинг.
- Примеры:
  - те же, что для SRP

Принцип инверсии зависимостей (DIP)

- Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.
- Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.
- Примеры:
  - те же, что для SRP.

Предметно-ориентированное проектирование

- **Domain Driven Design (DDD)** - Набор принципов и схем, направленных на создание оптимальных структур объектов, устойчивых к изменениям
- Техника проектирования и разработки, которая хорошо подходит для гибкой разработки
- Хорошо сочетается с микросервисной архитектурой

Преимущества DDD

- Позволяет автоматизировать незнакомые разработчикам предметные области
- Позволяет вести разработку итерационно, постепенно усложняя ПО
- Позволяет значительно ускорить разработку сложного ПО

Недостатки DDD

- Требует лояльности и гибкости заказчика (требование от Agile)
  - готовность заказчика участвовать в разработке
  - гибкость структуры заказчика по работе с контрагентами
- Требует сплочённой и профессиональной команды (требование от Agile)
  - умение использовать современные техники разработки ПО
  - готовность участвовать в изучении предметной области
- Но
  - некоторые приёмы DDD могут с пользой применяться и в неблагоприятных условиях (с осторожностью)

Основные определения

- **Область (Domain)** — предметная область, к которой применяется разрабатываемое программное обеспечение
- **Язык описания** — используется для единого описания модели предметной области
- **Модель (Model)** — описывает конкретную предметную область или её часть, является базой для автоматизации

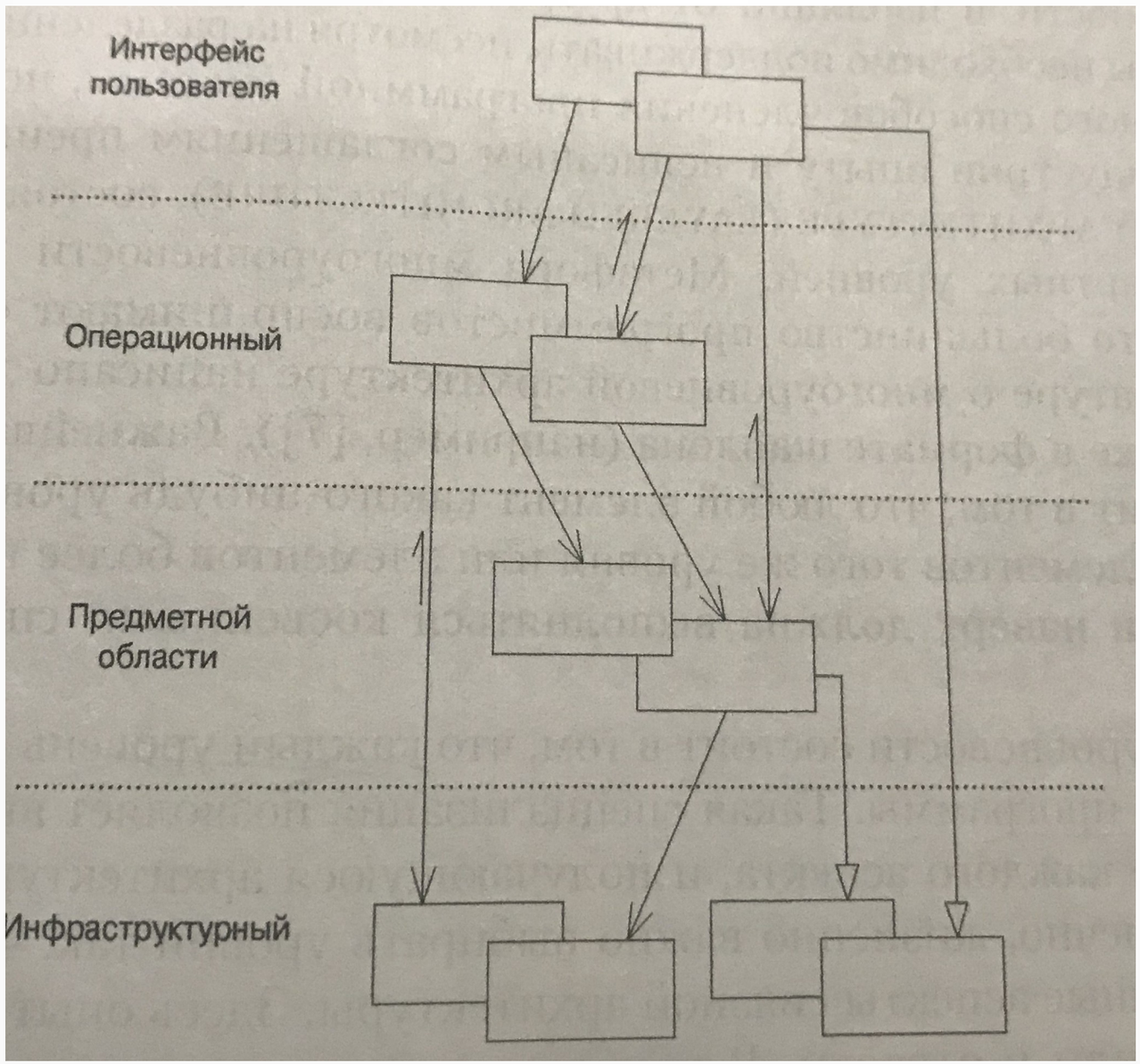
Многоуровневая архитектура
Уровни

- **Интерфейс пользователя** (уровень представления) – UI
- **Операционный уровень** (уровень прикладных операций, уровень приложения) – AL
- **Уровень предметной области** (уровень модели, уровень бизнес-логики) – DL
- **Инфраструктурный уровень** (уровень доступа к данным) – IL

Многоуровневая архитектура
Принцип зависимости уровней

- Каждый уровень:
  - зависит только от нижележащего слоя,
  - может существовать без вышерасположенных слоёв.
- Для связи с верхними уровнями используются:
  - обратные вызовы (callback);
  - ШП Observer;
  - стандартные архитектурные шаблоны:
    - Model-View-Controller (MVC),
    - Model-View-Presenter,
    - Naked objects,
    - и др.

Многоуровневая архитектура
Условная схема



Интерфейс пользователя

- Отвечает за:
  - вывод информации пользователю,
  - интерпретацию команд пользователя.
- Внешним действующим субъектом может быть не человек, а другая программа

Операционный уровень

- Определяет задачи, связанные с конкретным действием в UI (команда пользователя или потребность в информации) и распределяет их между объектами предметной области
- Не хранит состояний объектов предметной области
- Может играть интегрирующую роль - взаимодействовать с операционными уровнями других систем
- Наиболее близкий ШП: Fasade

Уровень предметной области

- Отвечает за:
  - представление понятий прикладной предметной области,
  - рабочие состояния,
  - бизнес-регламенты (поведение модели).
- Этот уровень является главной, алгоритмической частью программы

Инфраструктурный уровень

- Слой технических сервисов
- Обеспечивает техническую поддержку для верхних уровней:
  - передачу сообщений на операционном уровне;
  - непрерывность существования объектов на уровне модели (хранение, транзакционность и т.д.);
  - службы передачи сообщений;
  - почтовые службы;
  - и т.д.

Рефакторинг

- **Рефакторинг** — это такая реструктуризация программы, в результате которой не изменяются её функциональные возможности
- **Рефакторинг** выполняется, если необходимо сделать модель объектов проще, понятнее, легче для чтения
- Важным является то, что после рефакторинга регрессионные тесты должны выполняться без изменений

Углубляющий (уточняющий) рефакторинг

- **Углубляющий рефакторинг** выполняется перед тем, как углублять (дорабатывать) модель, то есть код
- При углубляющем рефакторинге вносятся структурные изменения в код программы, которые готовят его для доработки, выделяя области кода, в которые будут вноситься основные изменения
- Эти области кода изолируются, делаются менее зависимыми, чтобы проще было вносить дальнейшие изменения
- После углубляющего рефакторинга регрессионные тесты должны выполняться без изменений

Особенности техники проектирования по модели (и не только)

- Рефакторинг занимает значительную долю времени разработки
- Для поддержки такого стиля работы приложение должно иметь гибкую архитектуру

Гибкая архитектура
Определение

- **Гибкая архитектура** — набор правил DDD, позволяющих вносить изменения в код модели, при которых сохраняется его читаемость и возможность вносить изменения
- Понятие «гибкая архитектура», в первую очередь, относится к модели, т. е. к коду модели. Иначе говоря, предполагается, что изоляция предметной области уже выполнена

Гибкая архитектура
Правила и приёмы

- Информативные интерфейсы
- Функции без побочных эффектов
- Контрольные утверждения
- Концептуальные контуры
- Изолированные классы
- Замкнутость операций
- Разделение алгоритмов и данных

Гибкая архитектура
Функции без побочных эффектов

- Исходно:
  - В DDD «побочными эффектами» называют такое изменение состояния системы, которое влияет на её будущие операции.
  - Совокупное влияние нескольких правил или комбинации вычислений бывает очень трудно предсказать.
  - Операции, которые возвращают результат, не создавая «побочных эффектов», называются функциями.
  - Однако вносить изменения в состояние системы всё равно необходимо.
- Вывод:
  - Максимально используйте ***чистые функции*** (методы).
  - Максимально разделяйте чистые функции и команды, вносящие изменения в состояние.
  - Максимально используйте ***объекты-значения*** (неизменяемые классы) — лучше пересоздать объект, чем путаться в его состояниях.
  - Изменение состояния должно быть явным: либо очевидным из описанного контекста (см. *контрольные утверждения*), либо явно обозначенным в наименовании операции (см. *информативные интерфейсы*).

Event Storming
Определения

- **Event Storming** — это метод, основанный на сессионных встречах (семинарах) представителей заказчика и команды разработки, позволяющий быстро объяснить, что происходит в предметной области для проектирования её модели.
- Отличается лёгкостью в освоении и намеренно не использует компьютерного документирования (впрочем, как и всё в DDD).
- Результат выражается в наклеивании стикеров на длинную стену.

Event Storming
Типичный результат

Event Storming
Особенности

- Бизнес-процесс оформляется в виде серии событий предметной области, которые обозначаются как оранжевые стикеры.
- Event Storming может быть использован в качестве средства для моделирования бизнес-процессов и разработки требований.
- Основная идея состоит в том, чтобы объединить разработчиков программного обеспечения и экспертов в предметной области и учиться друг у друга.
- Чтобы облегчить этот процесс обучения, Event Storming должен быть увлекательным.

Event Storming
Название

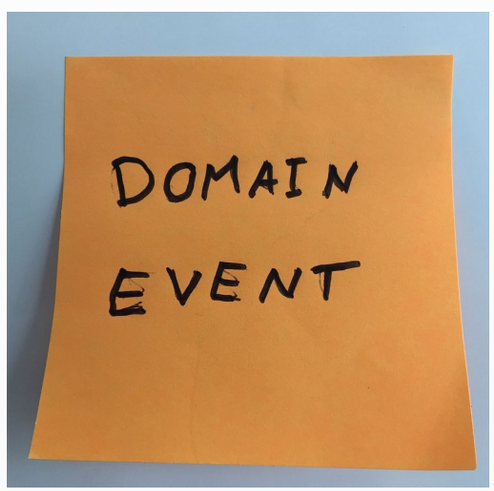
- Название было выбрано, чтобы показать, что основное внимание должно быть уделено событиям предметной области, и метод работает аналогично мозговому штурму или разработке моделей.

Event Storming
Требования

- Для семинара, посвященного Event Storming, важно, чтобы на нем присутствовали нужные люди. Сюда входят люди, которые знают, какие вопросы нужно задавать (обычно разработчики), и те, кто знает ответы (эксперты в предметной области, владельцы продуктов).
- Модель будет размещена на широкой стене с раскатанным на ней рулоном бумаги. Стикеры будут размещены на этой бумаге. Вам потребуется по крайней мере 5 различных цветов для стикеров.

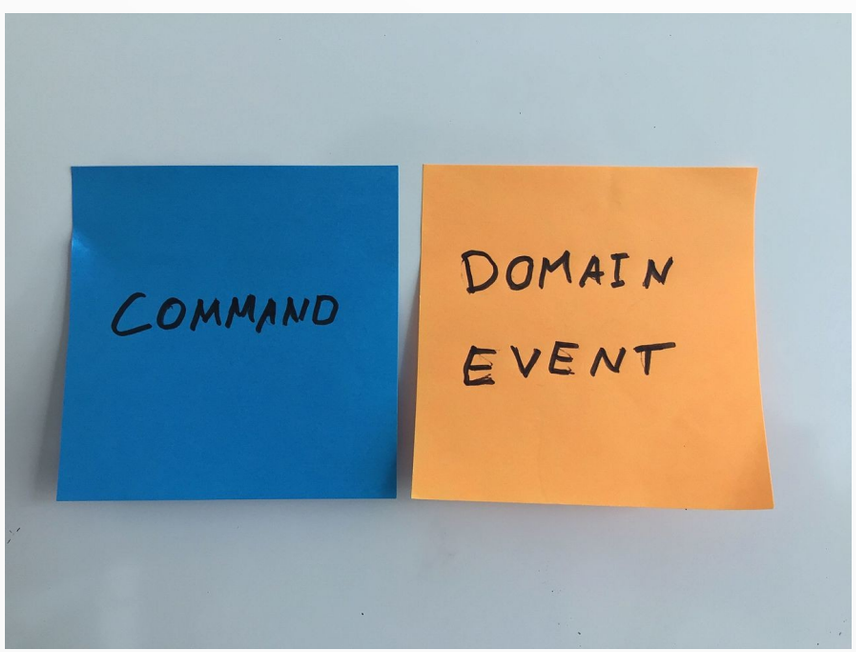
Event Storming
Шаг 1

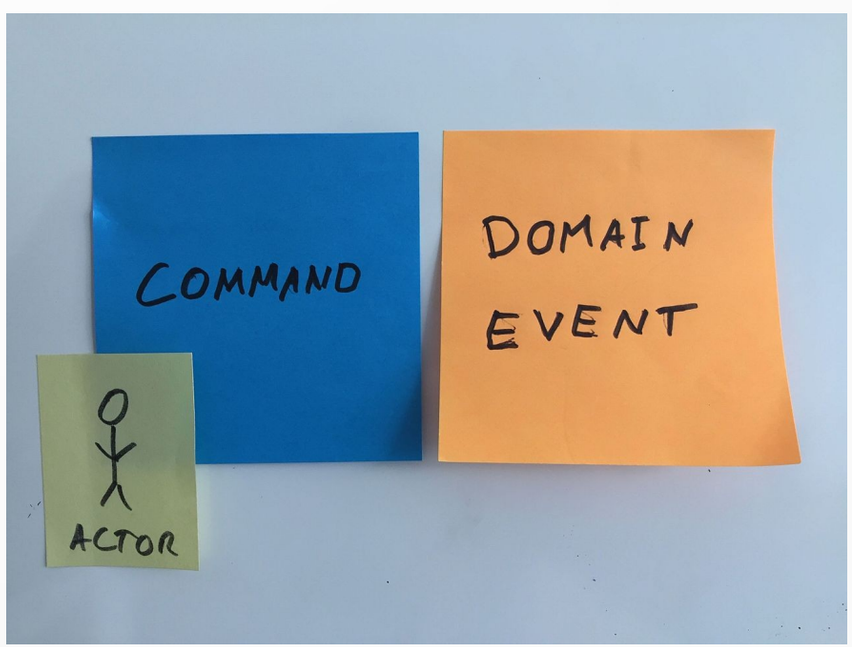
- Выбрать интересуемые события предметной области и записать их на оранжевых стикерах.



Event Storming
Шаг 2

- Когда все события предметной области выявлены, вторым шагом является определение команды, которая вызвала каждое из событий. Команды записываются на синих заметках и помещаются непосредственно перед соответствующим событием.





Event Storming
Шаг 3

- На третьем этапе определяются агрегаты, в пределах которых выполняются команды и где происходят события. Агрегаты написаны желтыми стикерами.

Event Storming
Категории собранной информации

- Информация о предметной области, собранная во время сессии Event Storming, делится на несколько категорий, каждая из которых имеет свой собственный цвет стикера:

Событие домена

Событие, которое происходит в бизнес - процессе. Написано в прошедшем времени.

Пользователь

Человек, который выполняет команду через представление.

Бизнес-процесс

Обрабатывает команду в соответствии с бизнес - правилами и логикой. Создает одно или несколько событий домена.

Командование

Команда, выполняемая пользователем через представление агрегата, которая приводит к созданию события домена.

Агрегат

Кластер доменных объектов, которые можно рассматривать как единое целое.

Внешняя система

Сторонний поставщик услуг, такой как платежный шлюз или судоходная компания.

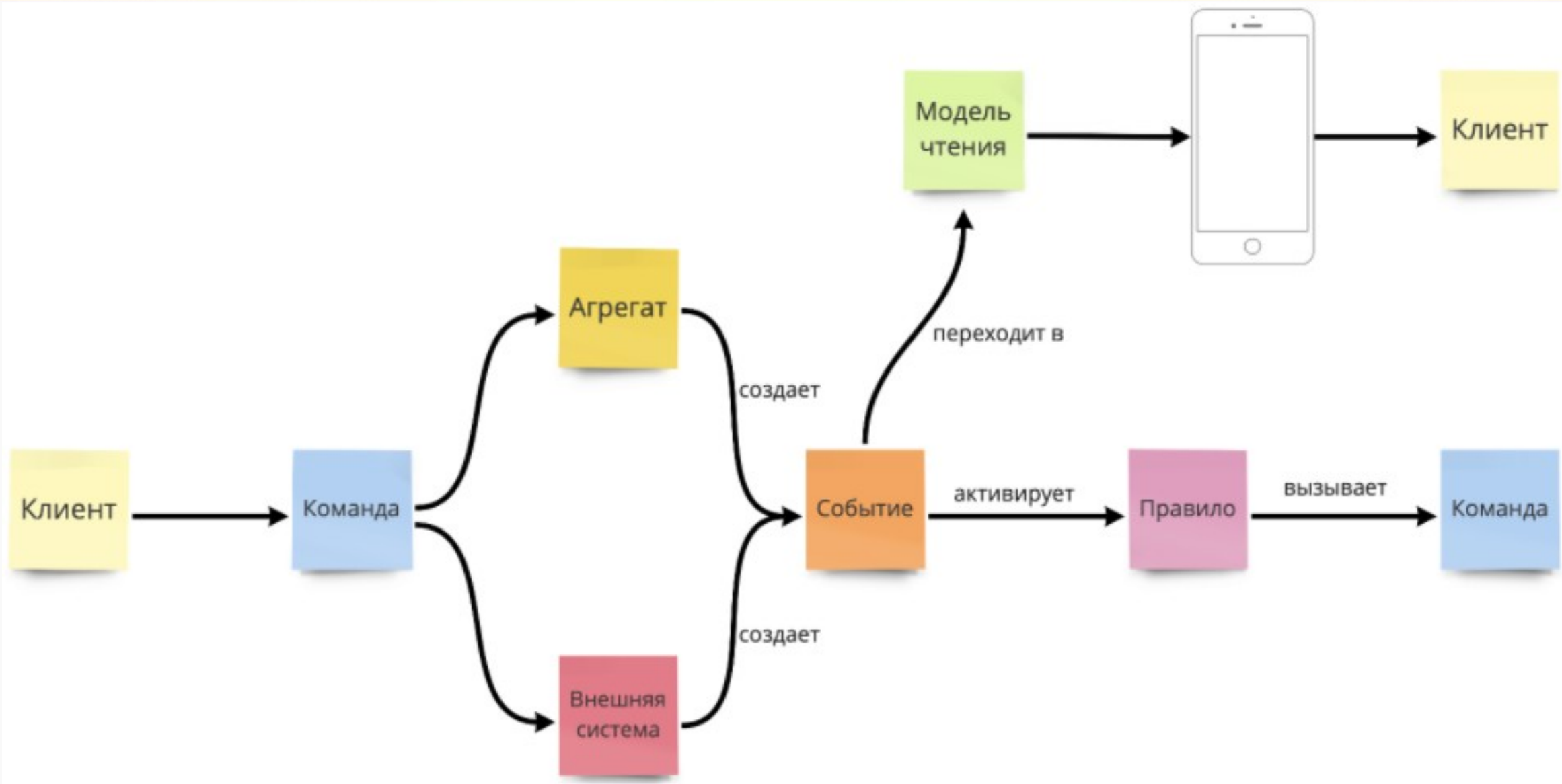
Вид

Представление, с которым пользователи взаимодействуют для выполнения задачи в системе.

Event Storming
Пример

- Источник:
  Моделирование микросервисов с помощью Event storming

Event Storming
Структура



Event Storming
Структура

- Клиент инициирует выполнение команды. Команда всегда формулируется в будущем времени: это запрос на выполнение действия. Оно еще не выполнено и может быть отменено.
- Допустим, речь идет о том, чтобы добавить товар в корзину. Команда приходит в агрегат (термин из DDD), либо уходит во внешнюю систему. Если команда может быть выполнена без нарушения инварианта агрегата, создается событие. Событие — это факт и действие, произошедшие в прошлом. В отличие от команды, событие не может быть отменено («Товар добавлен в корзину»).
- На основе событий, например, обновляется модель чтения, цель которой — помочь пользователю в принятии решений («Купи еще на 1000 рублей и получи скидку 10%») и выполнении последующих команд. Или же событие может активировать некое бизнес-правило. Которое, в свою очередь, вызовет команду, и цикл повторится.

Event Storming
Процесс. Шаг 1

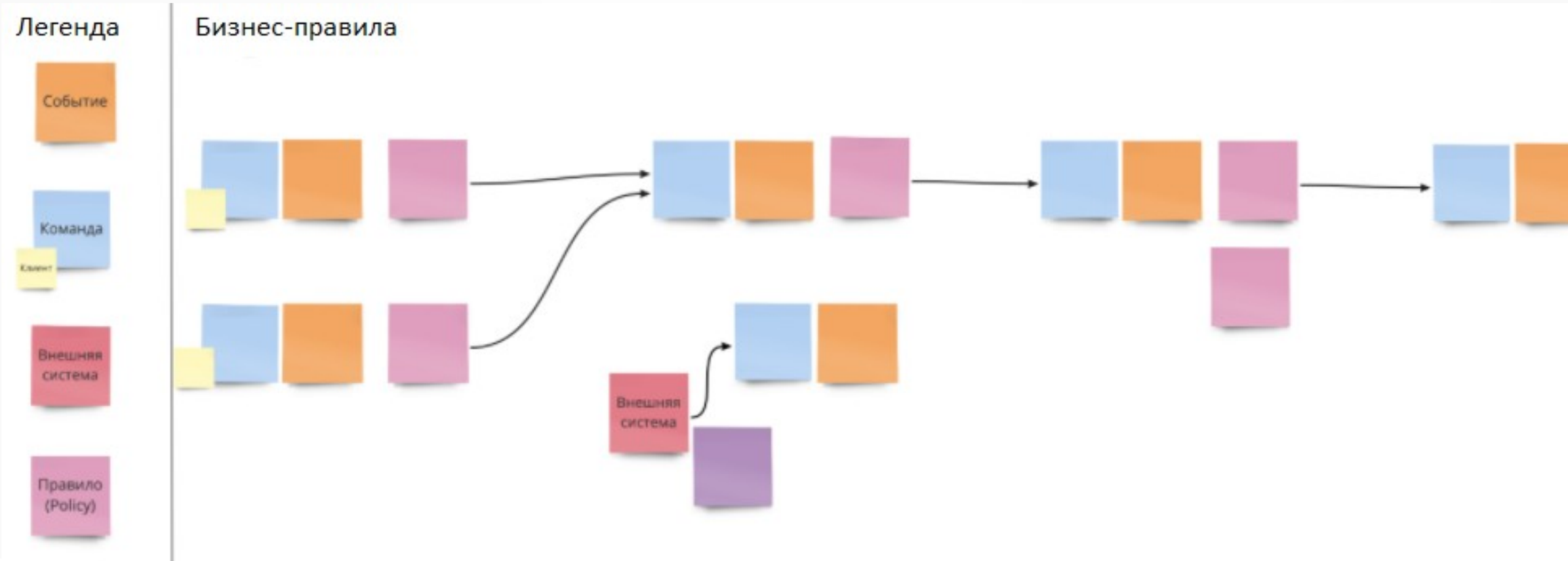
- Определяются абсолютно все события, происходящие в предметной области. Их могут быть сотни, и даже тысячи.
- Примеры событий: товар добавлен в корзину; товар оплачен; доставка оформлена.

Event Storming
Процесс. Шаг 2

- К этим событиям приписываем команды. Все очень просто. Товар добавлен в корзину. Команда, которая относится к этому событию: «Добавить товар в корзину».

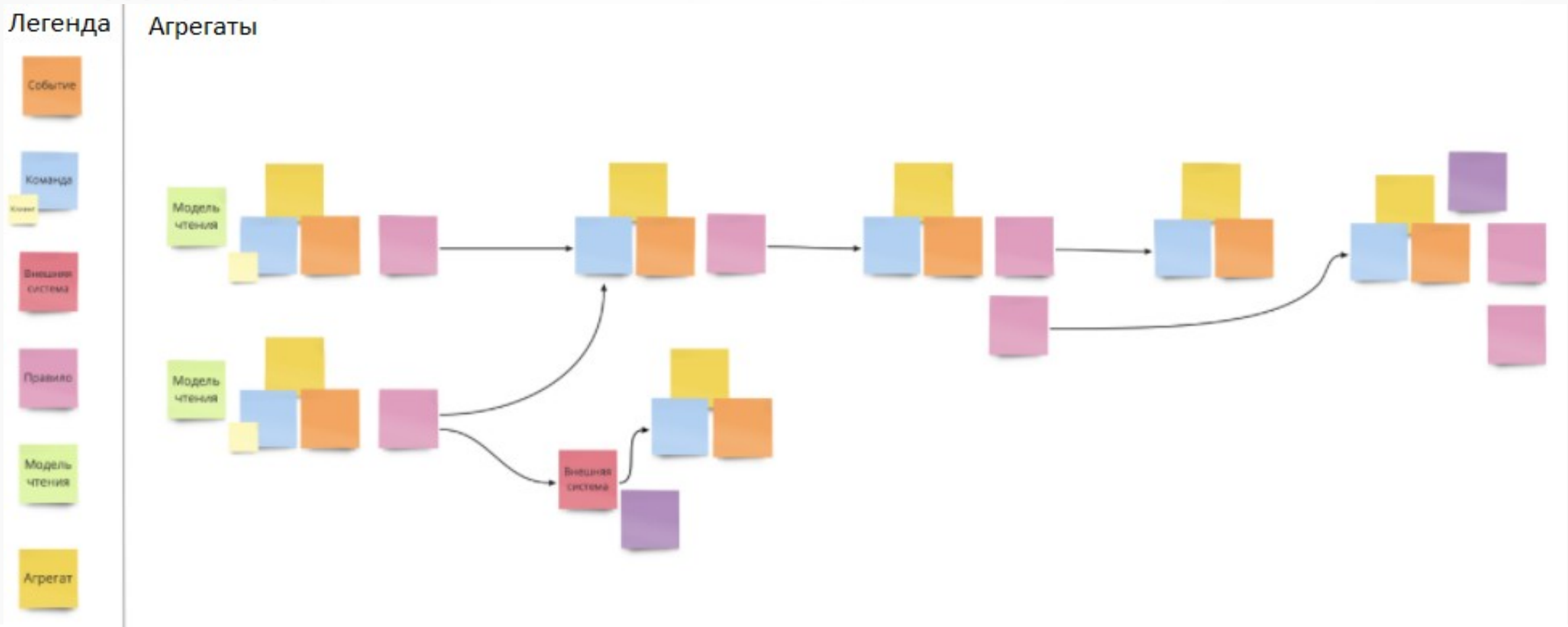
Event Storming
Процесс. Шаг 3

- Постепенно идет усложнение и появляются определенные бизнес-правила. Здесь очень хорошо помогают именно представители бизнеса, потому что программисты, по сути, переводят эти бизнес-правила в код. И важно здесь не то, как технически это выглядит, а то, как продукт работает в реальном бизнесе.



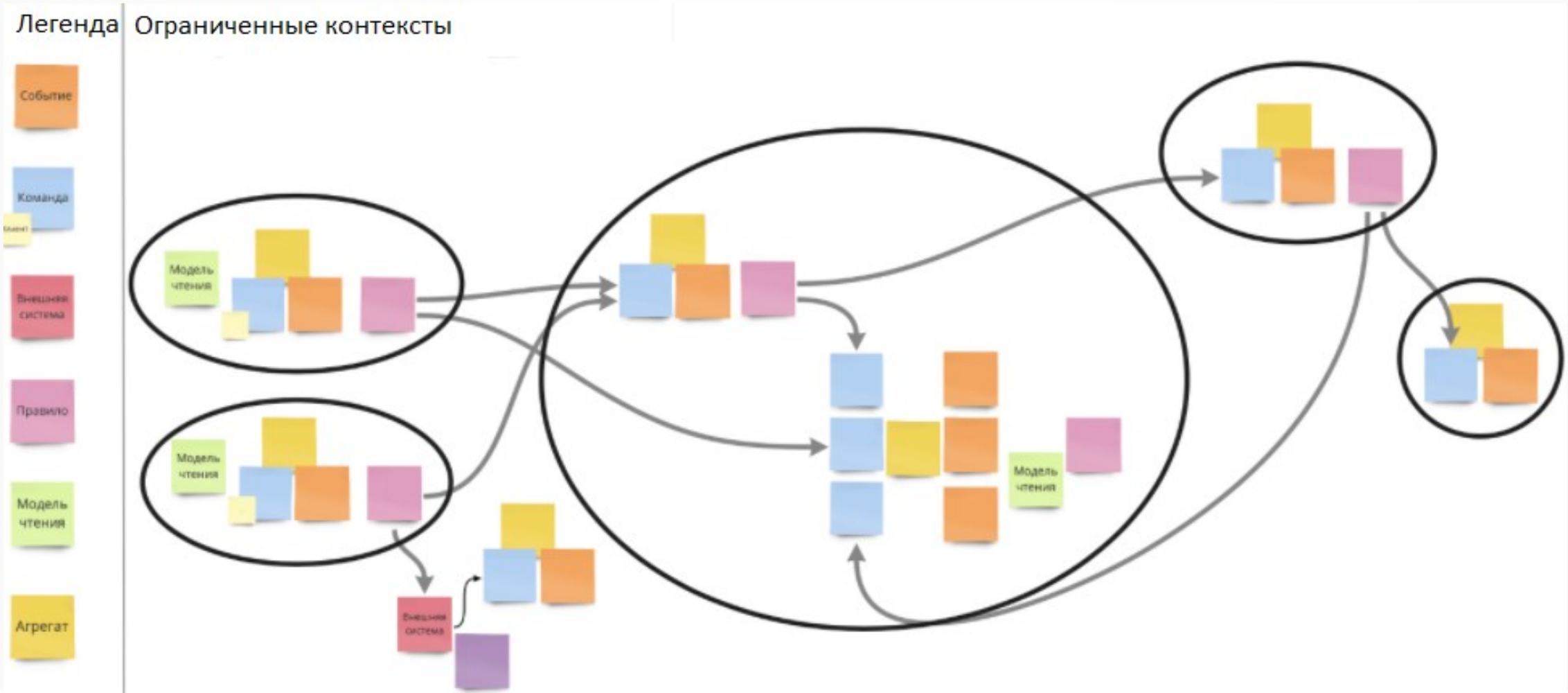
Event Storming
Процесс. Шаг 4

- Появляются агрегаты. В нотации Event Storming агрегаты обозначаются желтыми стикерами. Вначале мы можем оставить эти стикеры пустыми, без названия. Мы пока не знаем, что это за агрегат. Просто будем собирать вокруг него те команды и события, которые, как нам кажется, должны обрабатываться вместе. А уже потом назовем его.



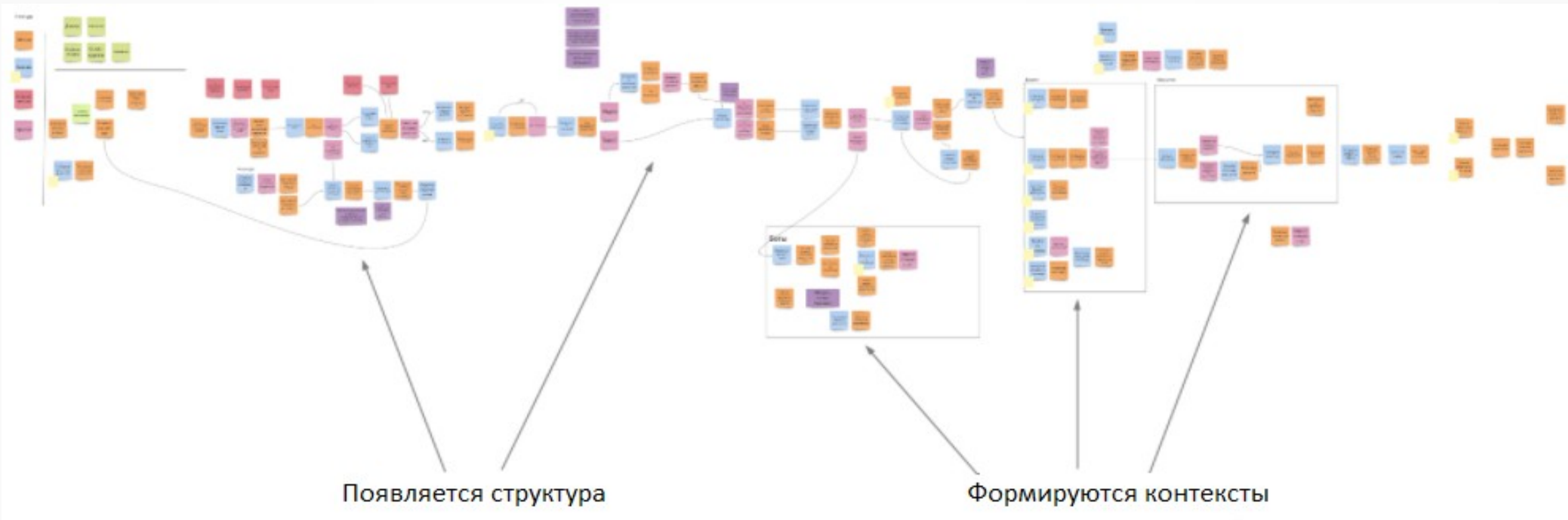
Event Storming
Процесс. Шаг 5

- Агрегаты сбиваются вместе и образуют то, что в терминах DDD называется ограниченными контекстами. Они представляют из себя модель + общий словарь.



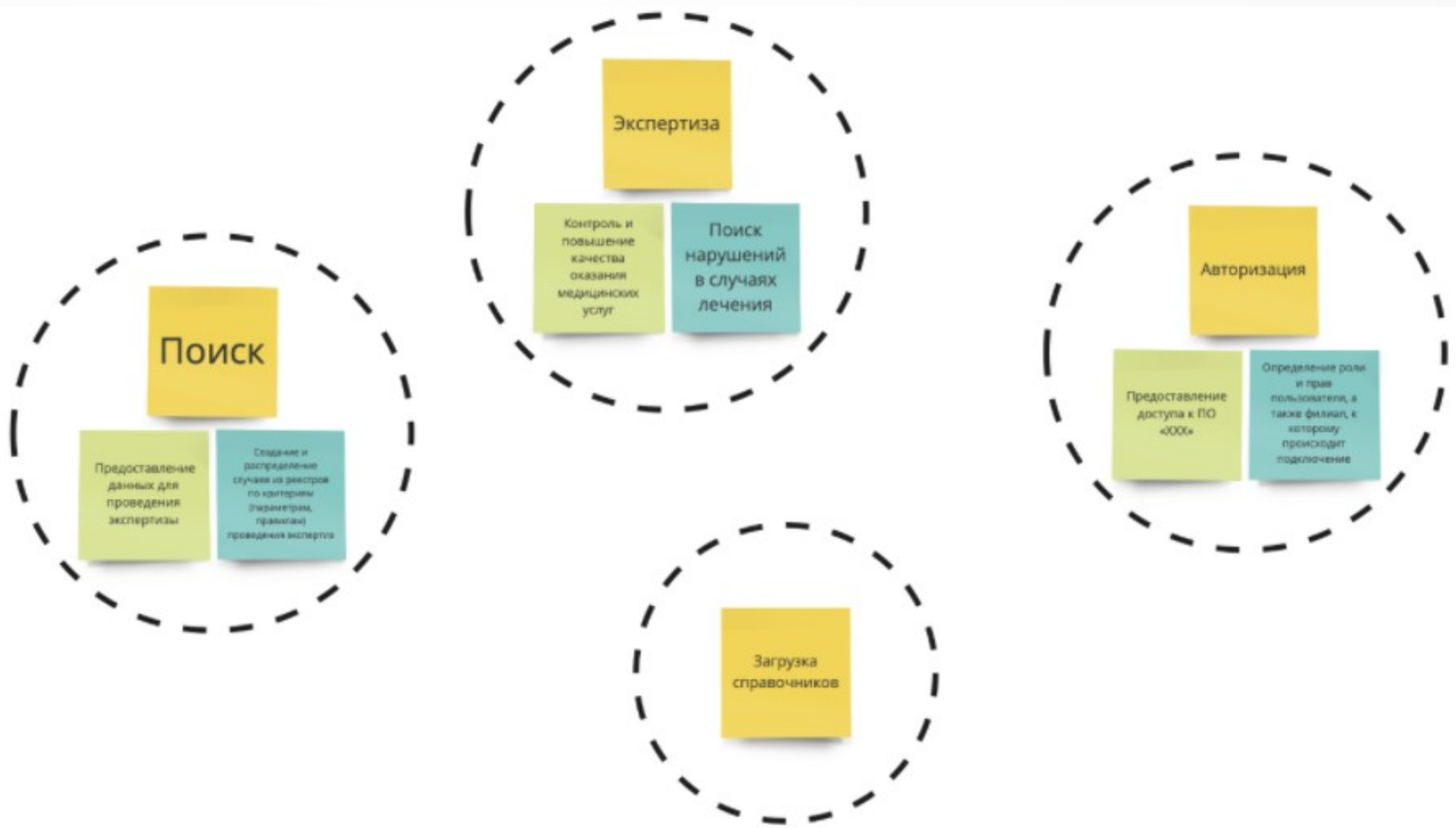
Event Storming
Процесс. Шаг 5 (структура)

- Над этой схемой работали около пяти часов. Вначале выявили события, затем команды, добавили бизнес-правила. Со временем результатом командной работы стал сбор некоторых команд и событий в контексты. Например, контекст работы с ботами, контекст диалога с клиентом, контекст закрытия заявки.



Event Storming
Процесс. Шаг 5 (контексты)

- В примере выше речь идет о страховании.
- Разберем 4 контекста: поиск; экспертиза; авторизация; загрузка справочников.
- У них есть свои цели и своя ответственность. Например, цель экспертизы — контроль повышения качества оказания услуг, а ответственность — это поиск нарушений.



Event Storming
Процесс. Шаг 6

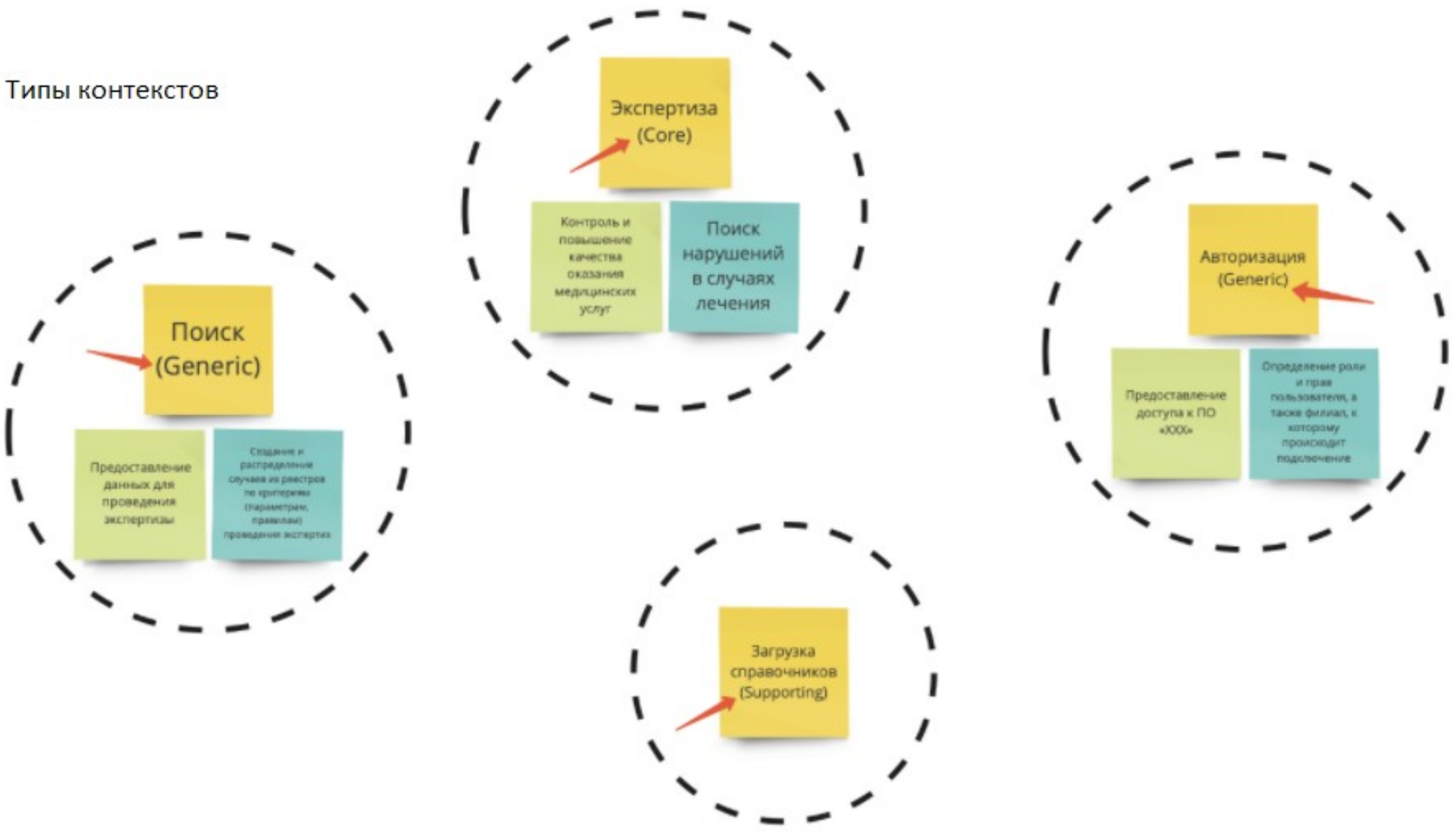
- Проставим типы этих контекстов. В общем случае их три:
  - Core — основной контекст;
  - Supporting — поддерживающий контекст;
  - Generic — общий для всех.
- Зачастую типы достаточно сложно определить. Люди в команде начинают о них спорить. Чтобы этого не произошло, можно использовать простую практику:

Контекст	Конкурентное преимущество	Сложность/стоимость реализации
Экспертиза	L	L
Авторизация	S	M
Загрузка справочников	S	S
Поиск	S	M

Event Storming
Процесс. Шаг 6

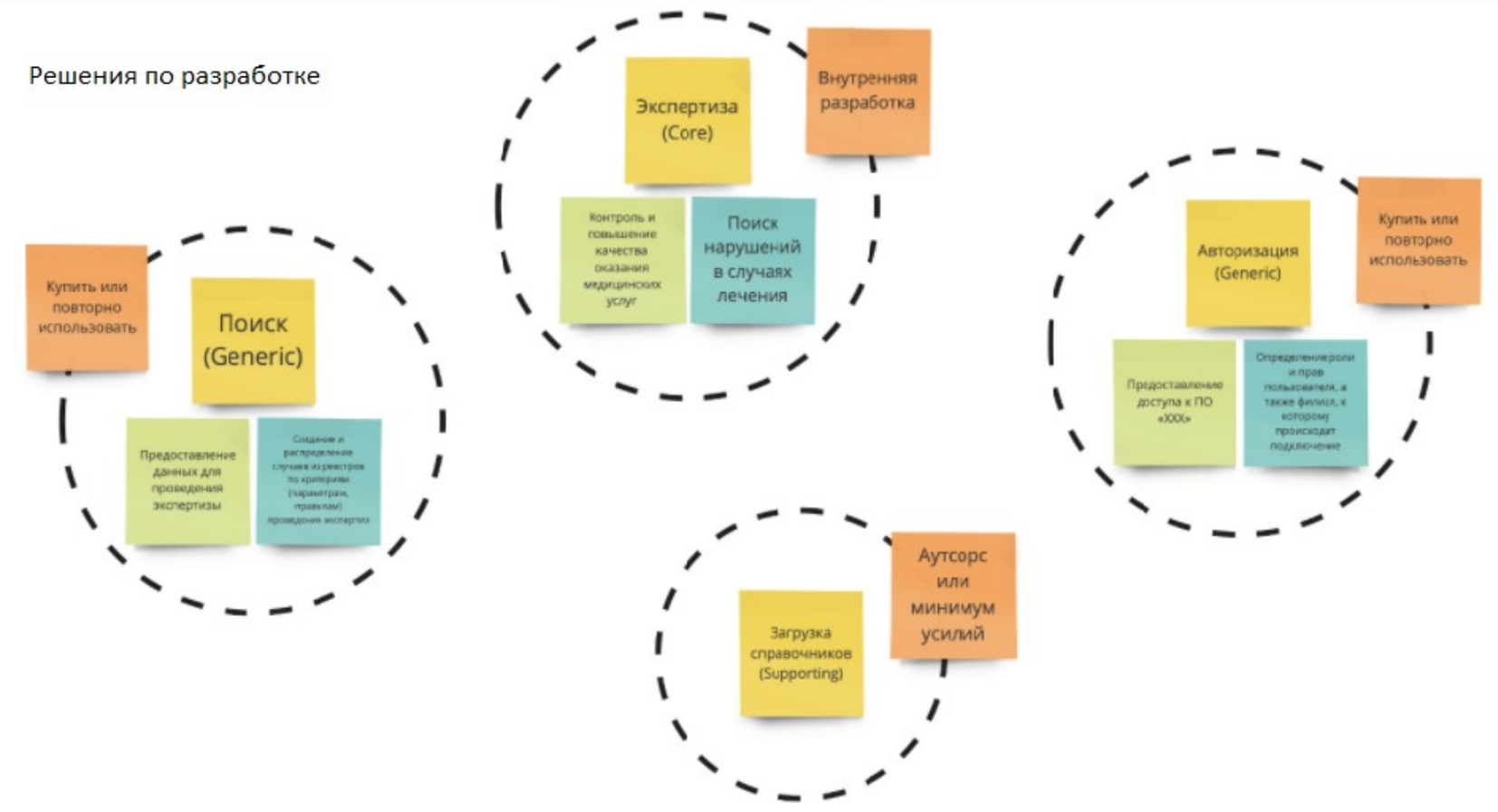
- Выписываем контексты и спрашиваем, насколько «Экспертиза» обеспечивает наше конкурентное преимущество/насколько сложна ее реализация:
  - L — сильное влияние/сложная реализация;
  - M — среднее влияние/средняя сложность;
  - S — слабое влияние/низкая сложность.
- Получаем набор значений. Нас интересует в первую очередь строчки с двумя L — то, что обеспечивает конкурентное преимущество, но может быть сложно реализовано. Кажется, что мы не можем отдать это на аутсорс или использовать внешнее ПО: это сердце нашего бизнеса.
- Дальше проставляем:
  - Core туда, где есть, как минимум, (L,L), (L,M);
  - Supporting туда, где нет особого конкурентного преимущества. В данном случае это загрузка справочников;
  - Generic (S, M).

Event Storming
Процесс. Шаг 6 (типы контекстов)



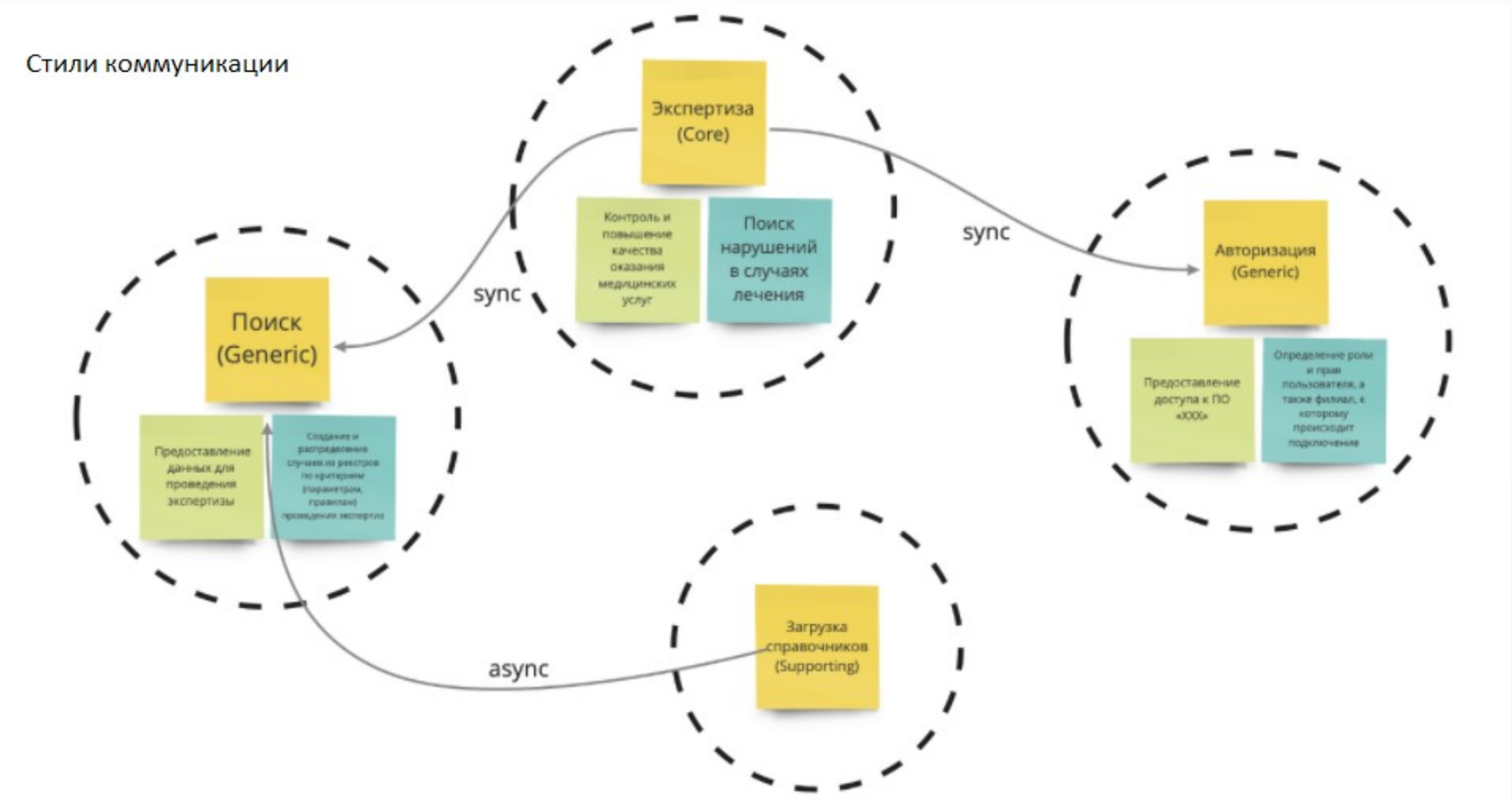
Event Storming
Процесс. Шаг 7 (решение по разработке)

- «Экспертиза» останется во внутренней разработке. Здесь будут сосредоточены основные микросервисы с основной бизнес-логикой.
- Загрузку справочников есть смысл либо отдать на аутсорс, либо оставить для прокачки джуниоров.
- Generic (поиск и авторизация) — это то, что можно использовать повторно. Здесь открываются новые варианты: купить решение, закрутить его в контейнер, и пусть оно там живет с небольшой кастомизацией.



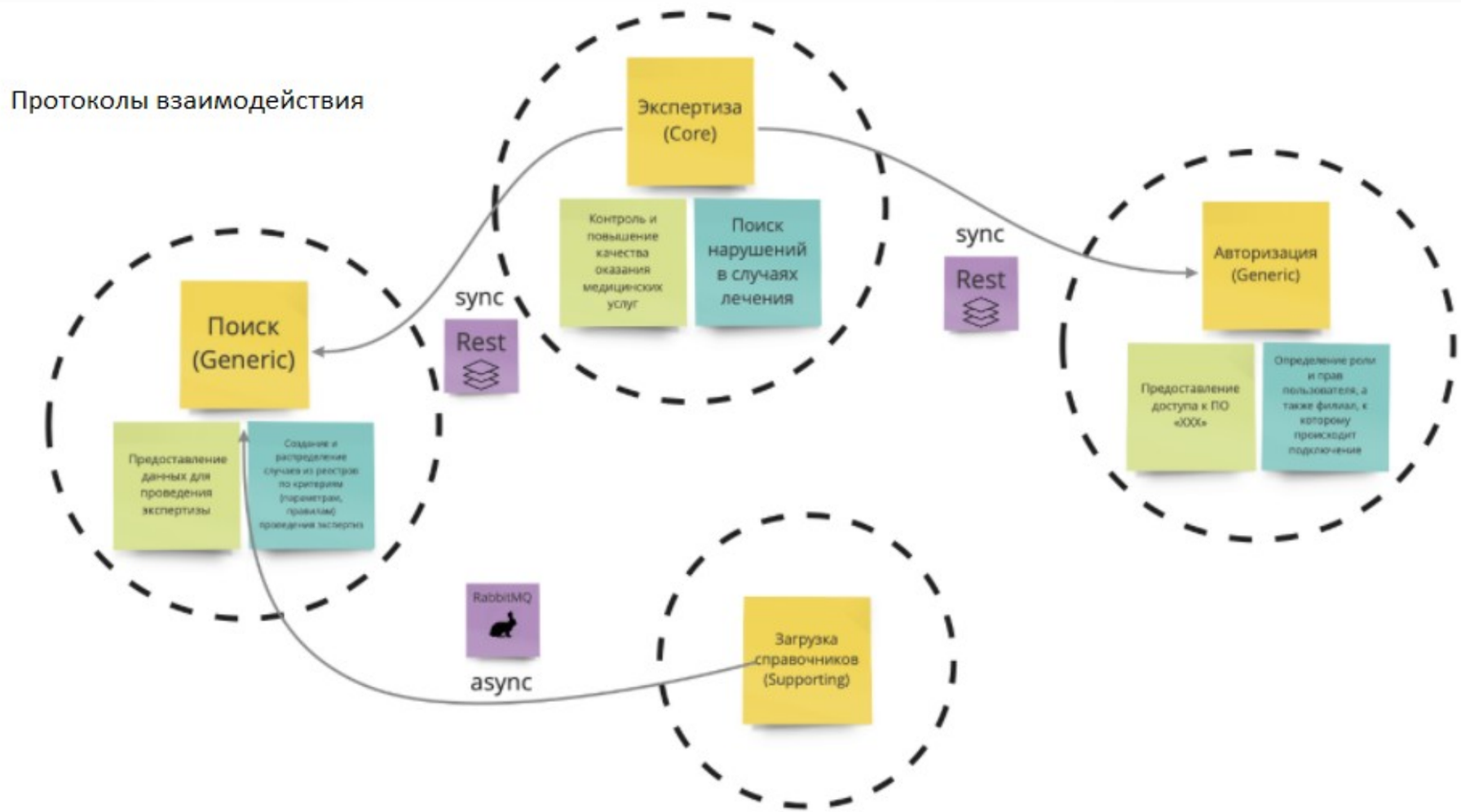
Event Storming
Процесс. Шаг 8 (стили коммуникации)

- Например, для экспертизы они таковы: как часто экспертиза будет обращаться к авторизации, и какую пропускную способность должна авторизация держать.
- У нас уже есть предметная область для экспертизы. И мы можем сказать, каким из этих элементов предметной области требуется постоянная авторизация доступа, или какие объемы данных будут передаваться из справочников. То есть появляются интеграционные связи. И на то, чтобы этого добиться, потребовалось меньше двух дней.



Event Storming
Процесс. Шаг 8 (протоколы взаимодействия)

- В качестве протоколов взаимодействия мы будем использовать Rest и RabbitMQ. Почему RabbitMQ? При загрузке справочников понадобится трансформация данных, не только чисто техническая, но и обеспечивающая защиту предметной области от протекания абстракций.

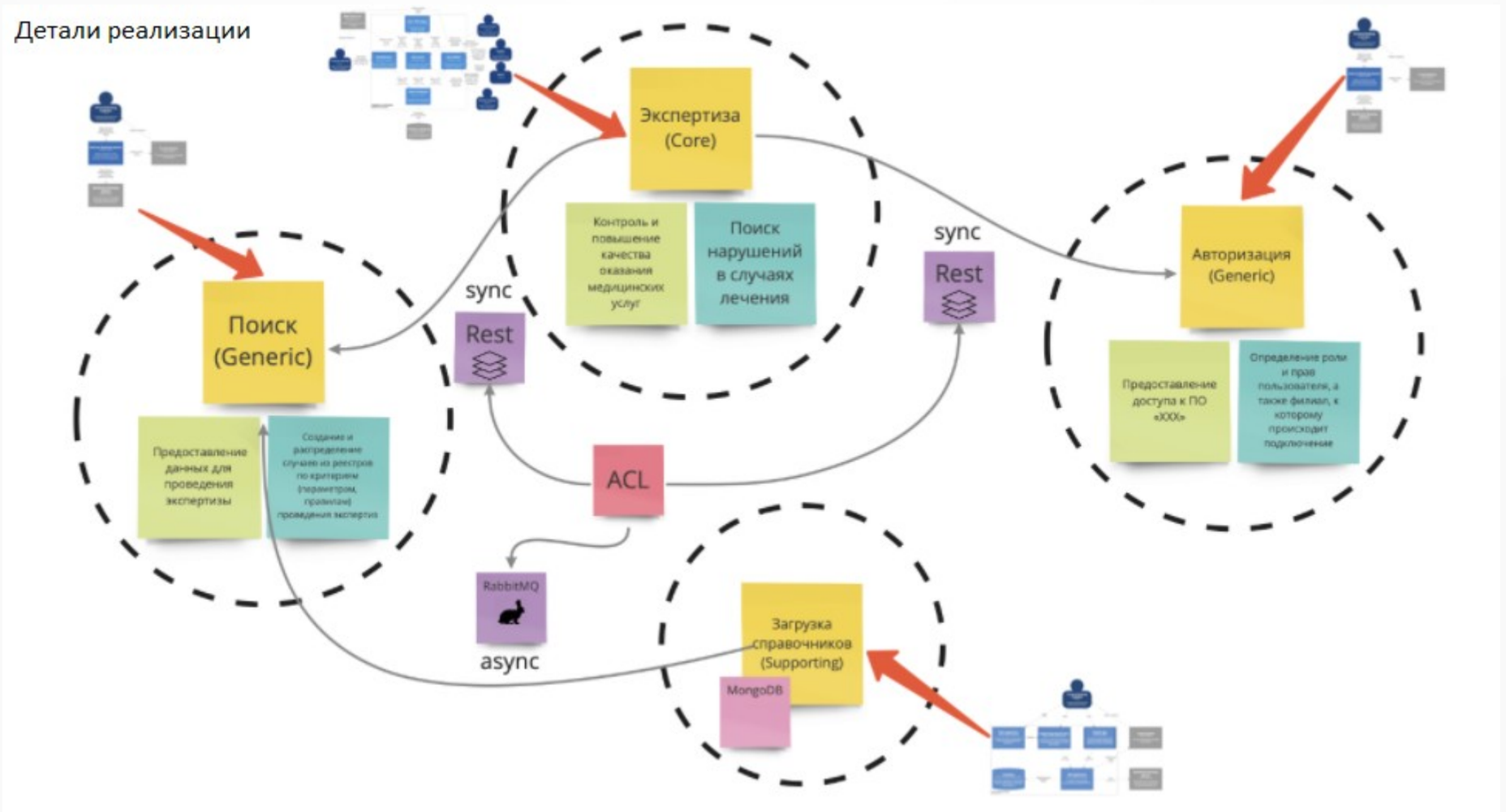


Event Storming
Процесс. Шаг 8

- В DDD для защиты модели мы можем использовать шаблон ACL (nti-Corruption Layer). По сути, это дополнительный слой адаптеров между контекстами, позволяющий осуществлять взаимодействие в обоих направлениях и снимающий зависимость контекстов друг от друга. То есть он защищает вашу предметную область, чтобы туда не попадали специфичные для других контекстов данные.
- Сами справочники могут храниться абсолютно в любом виде. Важно, чтобы в конкретных микросервисах мы работали со своей предметной областью, то есть брали из справочников только то, что нам нужно, и переводили эти данные в нашу предметную область.
- А еще важно придерживаться простого правила: именовать микросервисы и переменные в коде терминами, понятными бизнесу. Тогда нам не нужно будет переводить данные из одного языка в другой. Эти термины мы выявили еще на этапе поиска событий, команд и агрегатов.

Event Storming
Процесс. Шаг 9 (детали реализации)

- Важно, что внутри каждого из контекстов команда самостоятельно может провести дальнейшую декомпозицию и проектирование.



Event Storming
Выводы

- Event Storming помогает быстро получить стратегический дизайн, определив границы, в рамках которых технические решения можно принимать автономно.
- В результате бизнес-процесс можно увидеть на пространстве моделирования. Но важнее те знания, которые были заложены в сознании участников.
- Event Storming помогает быстро и непринуждённо наладить коммуникацию с экспертами предметной области от заказчика.