CUSTIS

Системное мышление: что это и зачем нужно тестировщику?

Максим Цепков
Главный архитектор решений CUSTIS
Навигатор по миру Agile, бирюзовых организаций и спиральной динамики

http://mtsepkov.org

SQA Days #35

25–26 октября 2024, Москва

35 sqa days

Что полезного в этом докладе?

Хорошие тестировщики **отвечают за качество продукта,** а не просто проверяют функционал тикета.

- Это требует понимания пользователей, которые используют продукт.
- Это требует понять, какие аспекты качества требуется от продукта.
- Надо выстроить процесс тестирования, чтобы обеспечить нужное качество.
- А для проверки устойчивости и удержания нагрузок надо разбираться, как софт работает на уровне hardware, что происходит под нагрузкой.

Для этих целей недостаточно стандартных схем, таких как пирамида тестирования или C4 Model, хотя они помогают. Необходимо системное мышление — оно позволяет понимать устройство сложных систем и работу с ними.

Что такое системное мышление?

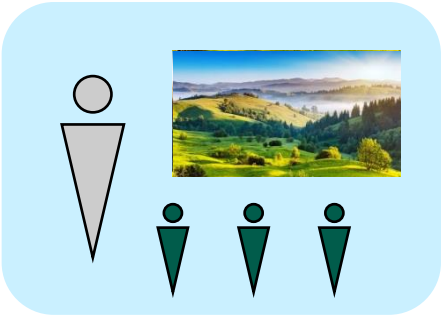
Как обычно, есть много ***разных*** мнений:

- это систематизированный подход к решению задач и проблем, обеспечивающий высокую вероятность их решения;
- восприятие мира как совокупности взаимодействующих систем;
- мышление, позволяющее создавать и изменять сложные системы.

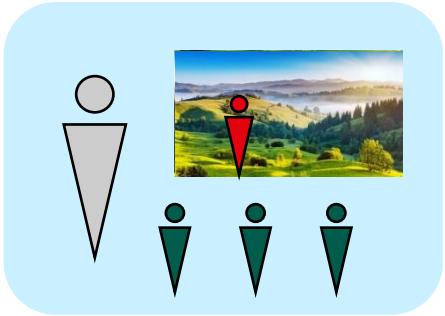
В определениях нет принципиальной разницы, отличие — в сложности решаемых задач, поэтому практически различие не столь важно.

Системное мышление включает много концептов, рассмотреть которые в одном докладе невозможно. В докладе будет лишь часть с примерами применения. Надеюсь, это побудит изучить глубже.

Не просто слушать, а думать о применении



Рассказ даёт карту местности
и возможные варианты движения.
Местность активно меняется



Ваша задача — увидеть на этой
карте будущий путь своей компании
и себя на этом пути

Путешествие не является необходимым, это зависит от ситуации. Но пока ты не представишь себя идущим, мой рассказ будет мёртвой теорией.

По мотивам схемы из моего доклада
«Как строить образ будущего и идти к нему: схемы самоопределения»

Какое качество нужно вашему проекту?

Весь мир состоит из систем

- Система — выделенный набор взаимодействующих объектов, для которых внутренние связи сильнее, чем внешние.
- Система может образовывать целостность, **большую чем сумма составляющих**: есть синергетический эффект, эмерджентность, и в этом случае описания частей недостаточно для описания целого.
- Первоначально систему понимали как объективное устройство объекта с физическими границами, как вскрытие чёрного ящика.
- Применение в биологии и социальных науках привело к понятию **soft system с нечёткими границами:** экосистема, система безопасности — от устройства мира перешли к способу мышления о мире.
- Выделение системы зависит от цели: для одних целей система «человек» ограничена его телом, а для других включает личные вещи, обеспечивающие функционирование, например смартфон.

Нечёткие границы систем

Пример: система доставки интернет-магазина

- Понятный процесс доставки: от склада до получателя.
- Внешние курьерские компании: это отдельные системы или материал для элементов нашей системы доставки, а SLA задаёт требуемую морфологию материала?
- А собственная служба доставки, она подобна внешним, она точно внутри?
- Формирование условий доставки при заказе клиента — в системе доставки или магазине?
- Если интернет-магазин в составе большой розничной сети, то у него отдельная служба доставки (система)? Или удобнее рассматривать всю службу логистики как одну систему?
- Может ли автономная собственная служба начинать выполнять чужие заказы?
- Что изменится, если наш интернет-магазин начинает продавать чужие заказы?

Все эти вопросы не только про бизнес, но и про гибкость ИТ-решений

Система и её окружение

Целевая система — та, над которой мы работаем в настоящий момент:

- различаем описание системы и её саму в реальном мире.

Целевая система встроена в **надсистему**:

- выполняет определённые функции в ней,
- отвечает на интересы агентов надсистемы.

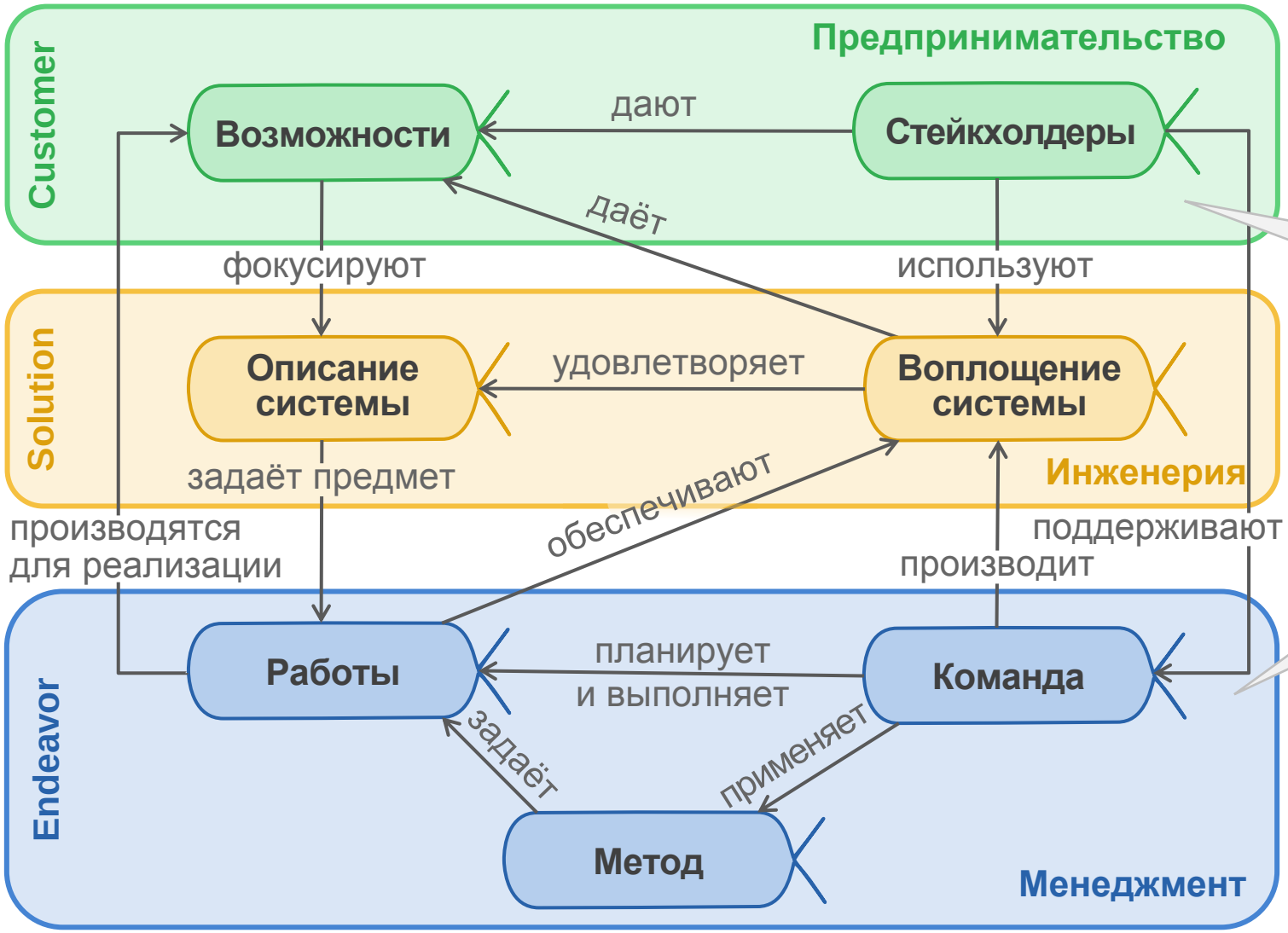
Эксплуатирующая (обеспечивающая) система обеспечивает работу целевой, обычно включает **команду** и другие части, живые и неживые.

Создающая система — та, что создаёт и развивает целевую.

Окружение системы — смежные системы, с которыми целевая взаимодействует в процессе своей работы.

Нетривиальные связи — закон Конвея: отражение в структуре создаваемой ИТ-системы орг. структуры команд создающей системы.

OMG Essence: сопряжение систем



Три системы:
надсистема, целевая и создающая;
для ИТ — бизнес, софт и команда.

Сопряжение с надсистемой:
ценность и люди

Создающая
и эксплуатирующая
система

Качество определяется **требованиями** надсистемы — бизнеса и **возможностями** создающей системы — команды.

Продукт как возможность

Пример. Бизнес требует поддержать услугу срочной доставки, так как без неё есть риск проиграть конкурентам, у которых она уже появляется.

- Вопросы о возможности: кто будет пользоваться, для каких заказов, что значит «срочная», как мы будем проверять гипотезы и какова мощность в динамике?
  - Справится пеший курьер или нужна машина?
- Будет ли отдельная система срочной доставки? Или надо существующую систему доставки дополнить элементами и наделить способностью так доставлять?
- Какова целевая операционная стоимость доставки, какова стоимость в пилоте, каковы допустимые инвестиции, в том числе для создания ИТ-систем?
  - Если курьер будет брать самокат, стоимость останется допустимой?
- Какие возможны лёгкие решения на пилоте, справится ли Excel + мессенджер?
- Как будем реагировать, когда не справляемся, кого из клиентов обижать?
- И так далее…

Требования к качеству от бизнеса

Реальные требования основаны на сценариях использования:

- пока машина едет от точки загрузки до ворот склада, необходимо успеть напечатать сопроводительные документы, чтобы выдать их водителю;
- после разгрузки фуры товар делится для отправки на несколько складов, сортировку должна провести та же бригада, что разгружала фуру;
- бухгалтер должен обработать все платежи по выписке до 11 утра, чтобы остаток по счёту в системе соответствовал остатку в банке;
- НДС должен составлять 18/118 от общей суммы накладной до копеек, так как налоговая сверяет общую сумму документа, а не строки.

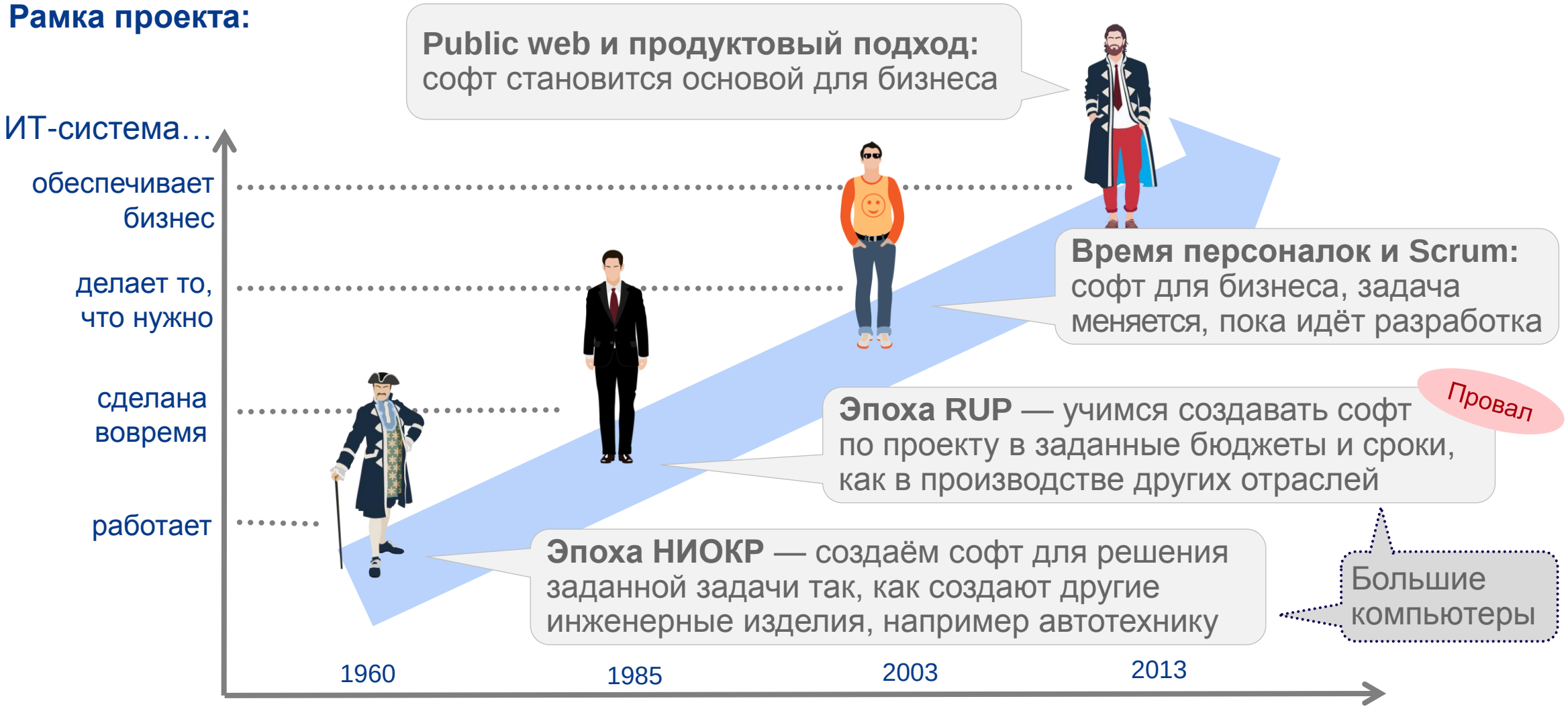
Надо выявлять такие требования качества в сценариях и думать о способе проверки, отличая их от формальных критериев на скорость отклика и им подобных, которые часто не имеют особых оснований.

Проектируем тестирование

- Есть модель пирамиды тестирования, она говорит про статистику и не определяет, что именно и каким образом тестируется.
- Тестировать весь функционал на всех слоях — дорого.
- Разумно, если слои дополняют, например, end-to-end тесты покрывают основной путь, а особые случаи тестируют unit-тесты, но только если проявления локальны.
- Часть блоков, которые нужны иногда, например при массовых распродажах или годовой отчётности, при выпуске релиза с функцией к празднику, — можно их исключить, только потом не забыть сделать заранее на случай проблем.
- Как модель здесь помогут тепловые карты по блокам или их аналог, а построить их и удержать сложность поможет системное мышление.

История культур ИТ-проектов

Рамка проекта:



Что такое качество?

Качественная система не просто соответствует спецификации, а **надёжно работает во всех во всех случаях**

Система **соответствует спецификации,** а разработка уложилась **в сроки и бюджет проекта**

Мы быстро создали и развиваем **софт,** который **нужен заказчику,** опираясь на обратную связь и сотрудничая с ним, а **неизбежные** ошибки быстро исправляются

С помощью нашего софта **стейкхолдеры** проекта могут **достигать своих бизнес-целей** в нужном для бизнеса темпе в соответствии с ожиданиями

Культура компании, включая процессы и представления о качестве должны быть адекватны типам проектов и **бизнес-модели.** Компании создают **гибридные варианты** со своей спецификой.

Подробнее — мой доклад на SQA Days #20, осень 2016 г.
«Ответственность за качество в разных ИТ-проектах: в чём она и как её разделять»

Влияние убеждений стейкхолдеров

У стейкхолдеров могут быть принципиальные различия убеждений, влияющие на проект:

- одни выступают за тщательное планирование как залог успешной работы, и нестрашно, если процесс планирования будет долгим;
- другие говорят, что есть много факторов, которые невозможно предусмотреть, и надо быть гибким, быстро реагировать на изменения.

Это влияет на организацию тестирования, на проведение демонстраций и т. д.

Убеждения связаны с представлениями о культуре, которые они когда-то приняли, и которую полагают правильной.

Надо видеть за словами в коммуникации по частным вопросам **убеждения стейкхолдеров** и работать с ними.

Агенты и роли

Деятельность ведут **агенты**: люди, команды, компании, сообщества.

Театральная метафора: **агент играет роль**, действуя в системе:

- человек может совмещать роли, например руководитель и разработчик;
- компания тоже играет роли в надсистемах: на рынке, в государстве, обществе.

Роль предполагает конкретное взаимодействие с другими ролями, которое определяется конструкцией системы.

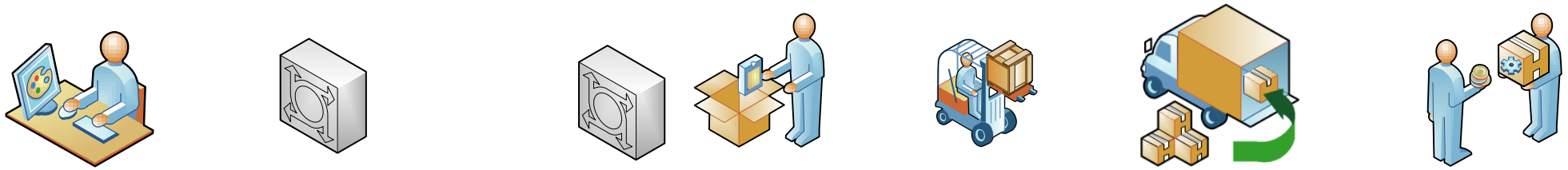
Агент имеет **цель**, для которой он встроился в работу системы, и она некоторым образом соотносится с **целью системы как агента**.

У агента есть **предпочтения** по поводу исполнения своей роли и **интересы**, касающиеся соучастия в результатах работы системы.

Бизнес и софт

Это две разных системы или два уровня в одной системе?
Ответ зависит от встройки софта в бизнес и от целей рассмотрения.

Три уровня представления: обработка интернет-заказа



Деятельность

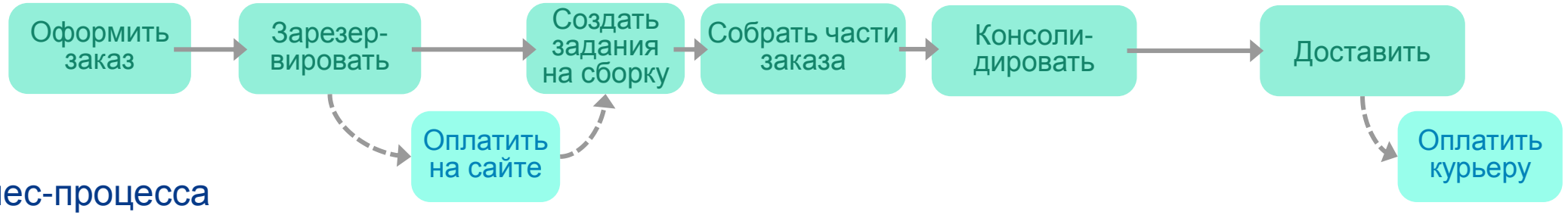
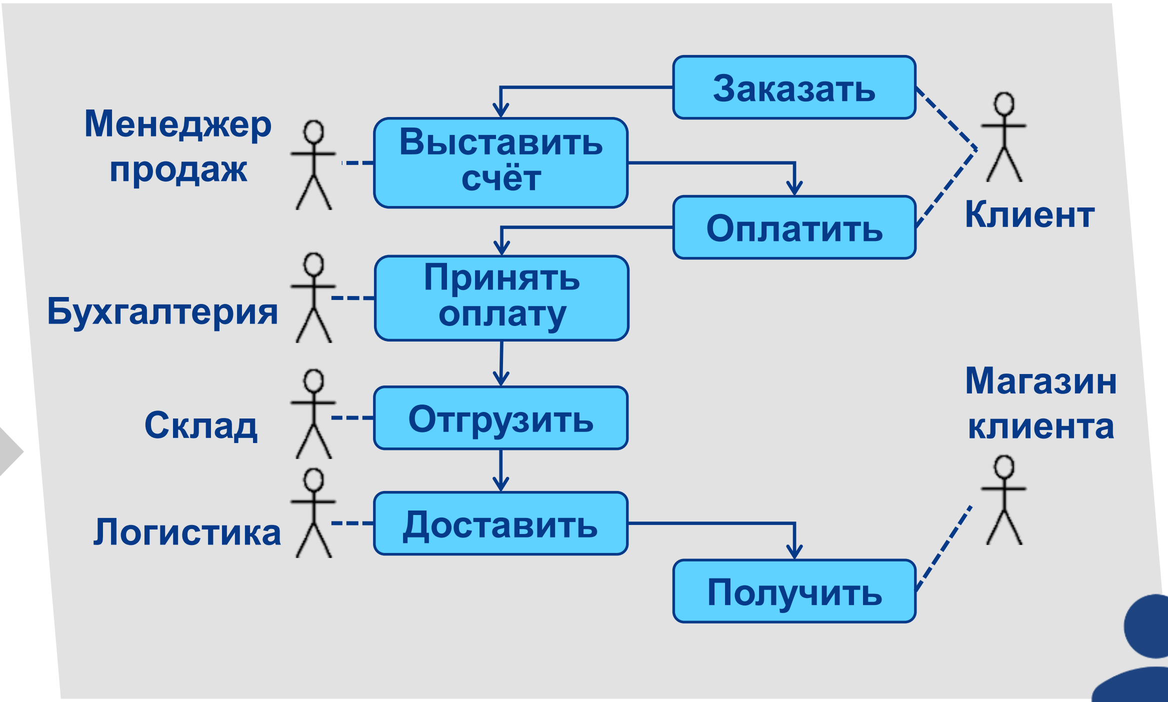


Схема бизнес-процесса

Объекты и их состояния

Формализация бизнес-модели

Оптовые продажи магазинам и торговым сетям



Как осуществляется переход к формальной бизнес-модели
от представлений о повседневной деятельности на интервью?

От нарратива к объектам

В процессе понимания текст размечается: в нём выделяются объекты, которые служат основой моделей:

- **физический объект** — нечто, выделенное в мире, имеющее идентичность, границы в пространстве и существующее во времени;
- **ментальный объект** — абстракция, идеальный объект в мышлении, полученный обобщением некоторых объектов, физических или ментальных, или их множеств.

Два типа моделей: мозаичная с экземплярами и концептуальная с обобщениями.

Разные типы **отношений** между объектами:

- обобщения и абстракции: экземпляр — класс, тип или аспект;
- иерархия абстракций: класс — подкласс или тип — подтип;
- часть — целое;
- агент — роль, интересы и поведение.

Конкретные отношения часто опускают в текстах и путают между собой.

Схемы для прояснения мысли

- ✓ Тексты для понимания требуют разметки, а схемы уже размечены.
- ✗ Однако, схемы не гарантия разметки, там тоже могут быть свои нарративы.
- ✓ Наработан ряд эффективных представлений для разных целей: диаграммы классов и ER-диаграммы, схемы состояний и бизнес-процессов и др.
- ✗ Проблема с представлением больших объёмов на схемах не решена.
- ✓ Табличные представления более строгие, позволяют представить больший объём, но воспринимаются значительно хуже.
- ✓ Полное представление — композит: схемы, таблицы, текст.

Event Storming и бизнес-процессы

BPM — моделирование бизнес-процессов:

- представляем деятельность как последовательность шагов с ветвлениями;
- хорошо подходит для основного потока операций, плохо — для исключений;
- есть работы, которые «состоят из исключений»: выверка отчётов, увязка планов, для них подходит Case Management, а не Process Management.

Event Storming:

- выдаёт событийную, а не процессную модель устройства деятельности;
- хорошо подходит для автоматизации «от событий» в сервисной архитектуре;
- помогает разобраться в бизнесе, но не даёт целостного представления;
- восстановить целостное представление, связав разные события, можно по-разному: через схему бизнес-процессов, через систему целей или через учёт.

Слабо структурированная деятельность

Есть довольно много примеров бизнес-функций, для которых сложно построить описание в виде процесса, при том что вовлечено много людей:

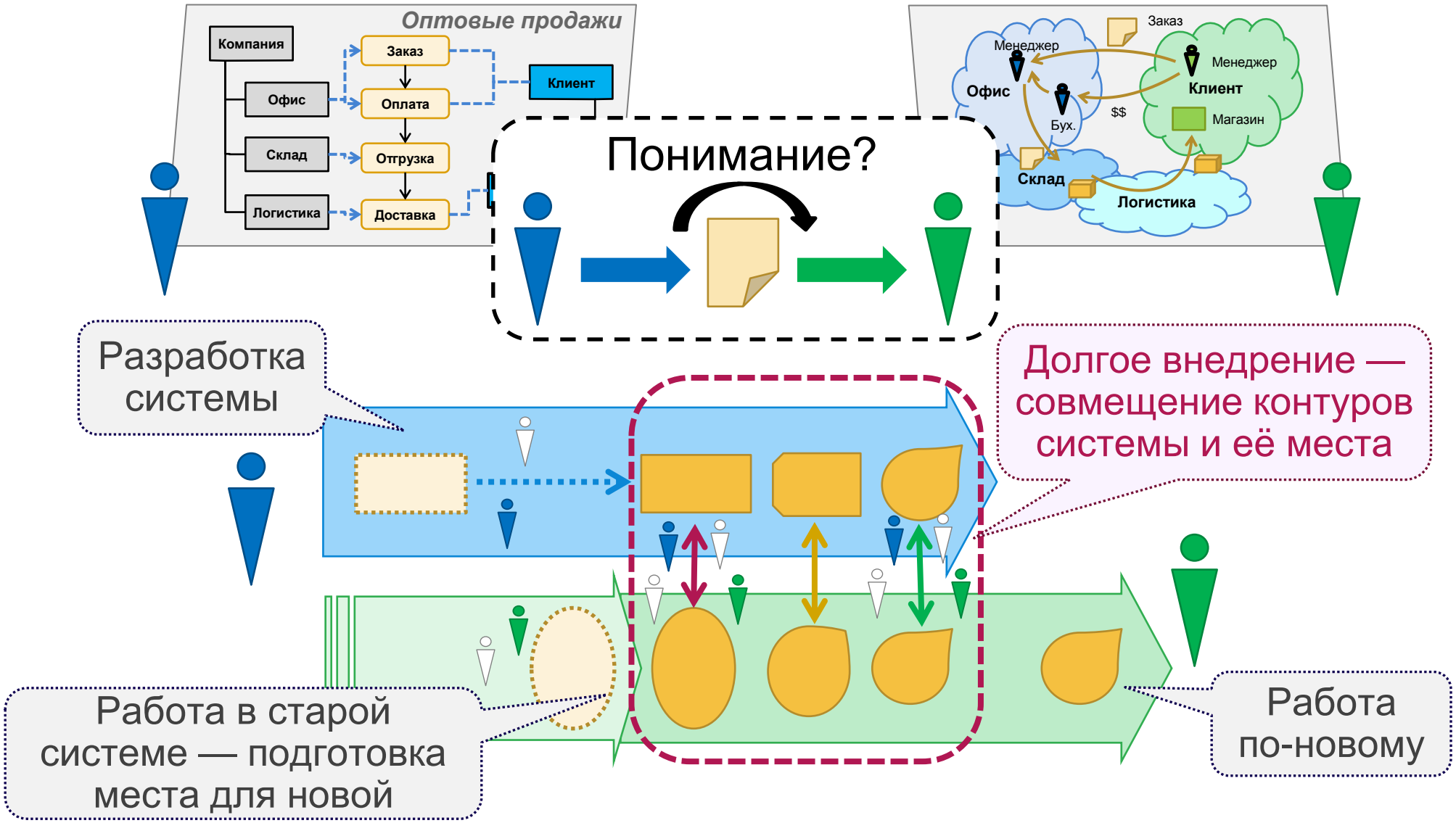
- выверка бухгалтерской отчётности с решением обнаруженных проблем;
- планирование расписания при недостатке ресурсов с переговорами;
- поиск баланса между текущей работой, ремонтом (техдолг) и развитием;
- а также обработка исключений и особых случаев.

Не надо пытаться описать подобное языком бизнес-процессов!

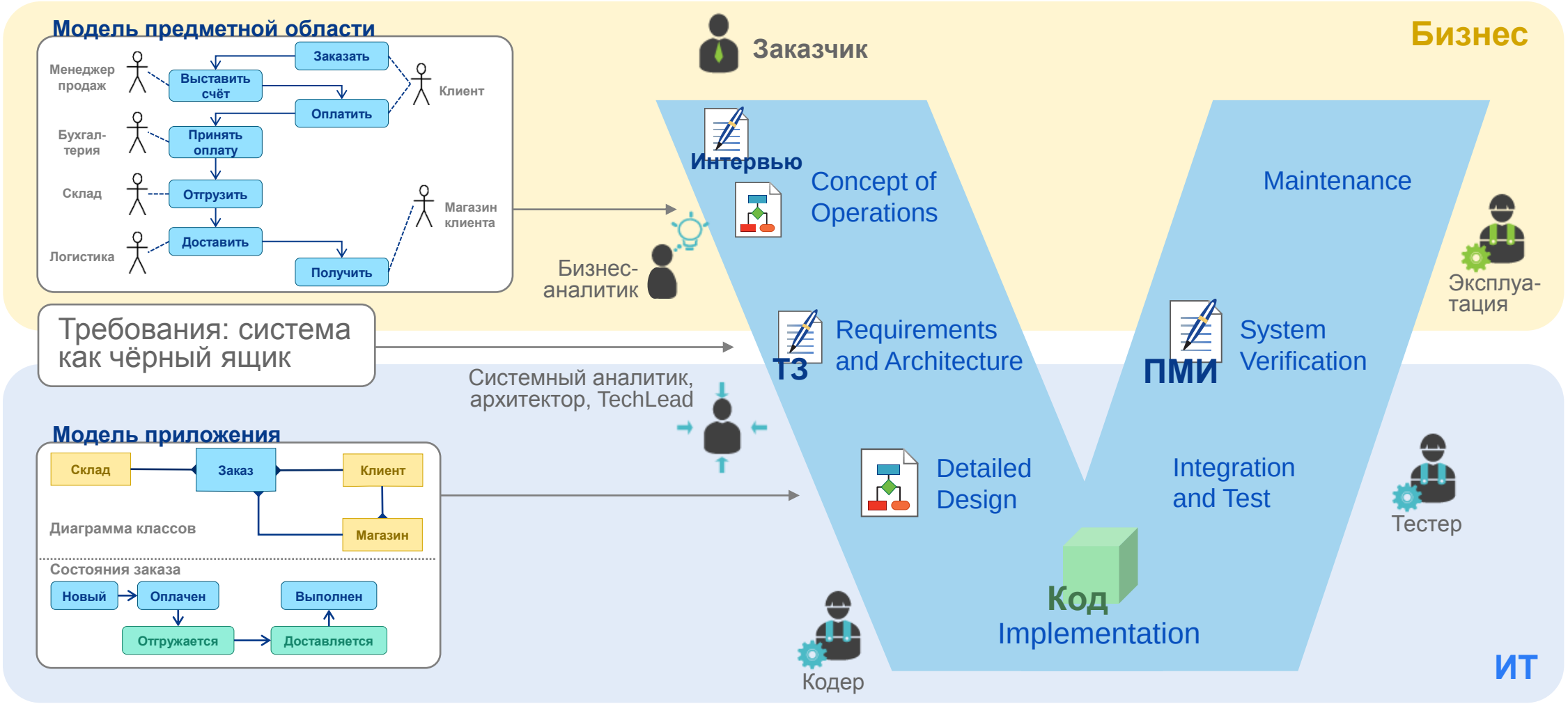
Альтернативой может быть координация по целям: критерии продвижения к цели и её достижения.

Это верно для любого метода: надо уметь видеть границы, и **не пытаться применять его не по назначению.** Апологеты методов часто выступают за повсеместное применение.

Непонимание влечёт долгое внедрение

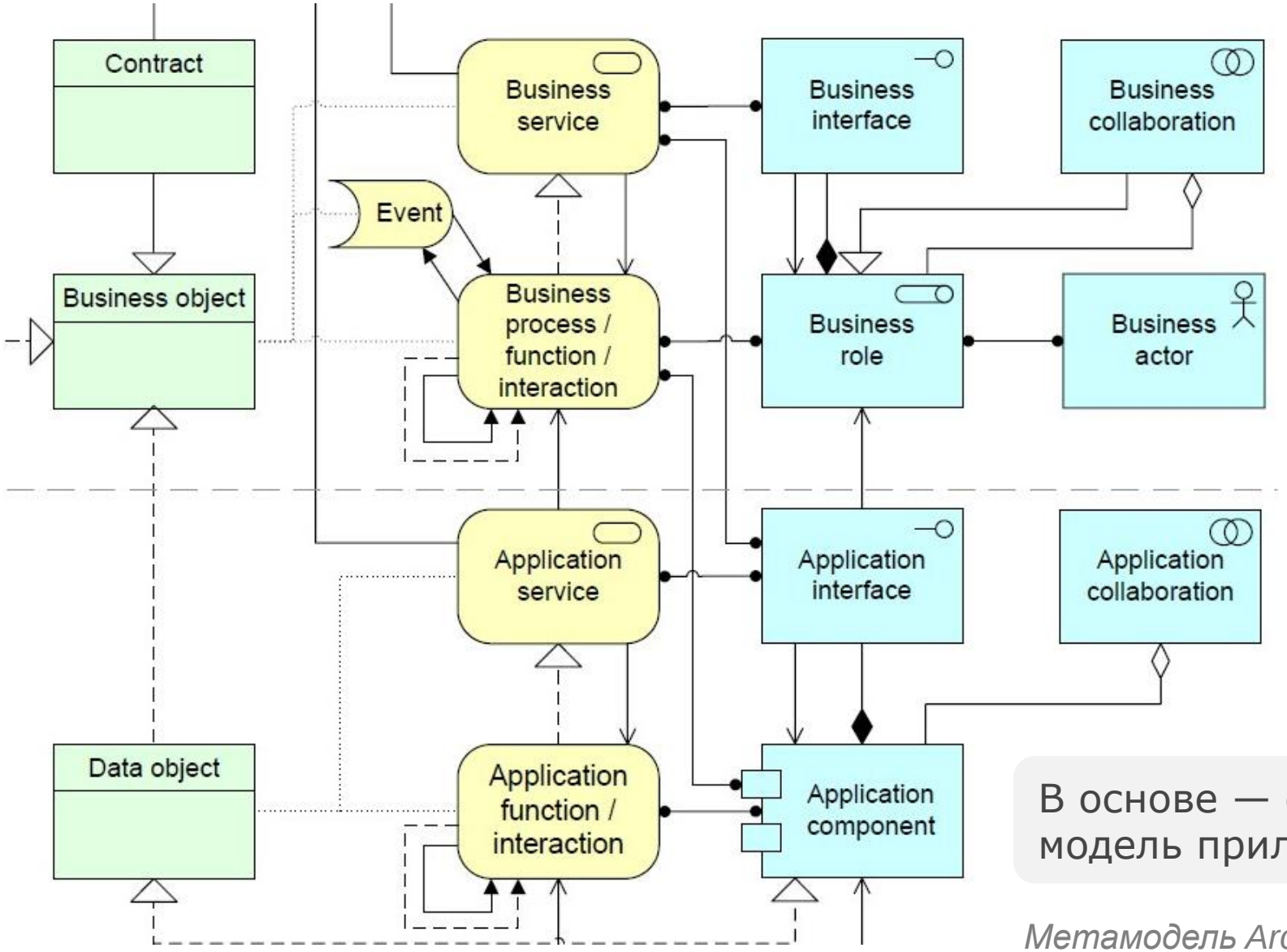


Разделение труда предполагало две системы

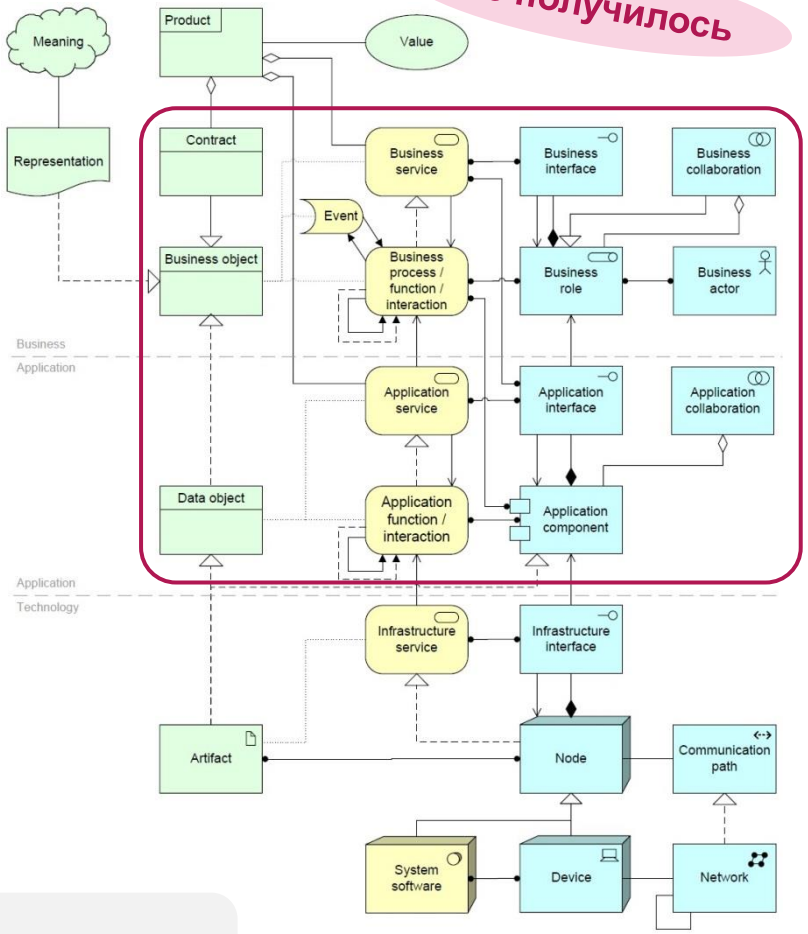


Опыт показал, что такое разделение приводит к несоответствию ИТ-системы ожиданиям бизнеса. Agile-методы это поменяли, но подход к описаниям остался.

Archimate — гибкая связь бизнеса и софта



Не получилось



В основе — сервисная модель приложения

Метамодель ArchiMate 1.0

Система в развитии, а не в моменте

Система рассматривается в динамике **развития**, различаем функционирование системы и **изменения** этого функционирования.

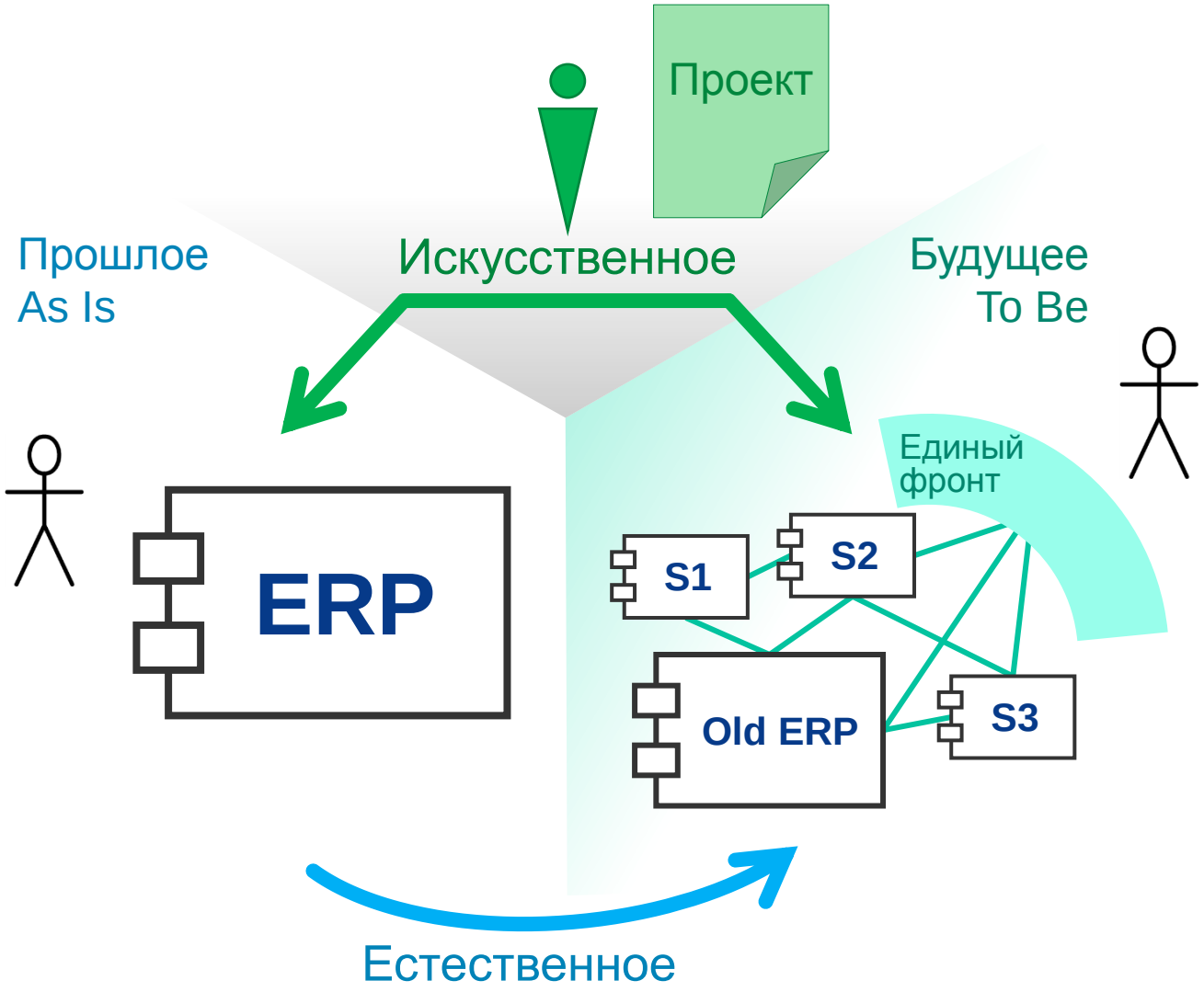
Системы **не стремятся** прийти к равновесию, гомеостазу с миром, у них есть **собственное движение развития**.

Источник развития — **неустроенности**: напряжения и противоречия между разными системными уровнями — баланс интересов системы и надсистемы:

- эволюция устраняет неустроенности экспериментально, методом проб и ошибок;
- люди могут сознательно менять системы для устранения неустроенностей, если знают их устройство, имеют хорошую модель; это эффективнее эксперимента.

Надо отличать **неустроенности** от логических противоречий в описаниях и моделях, связанных с языком и ограничениями моделей.

Шаг развития: не забываем динамику



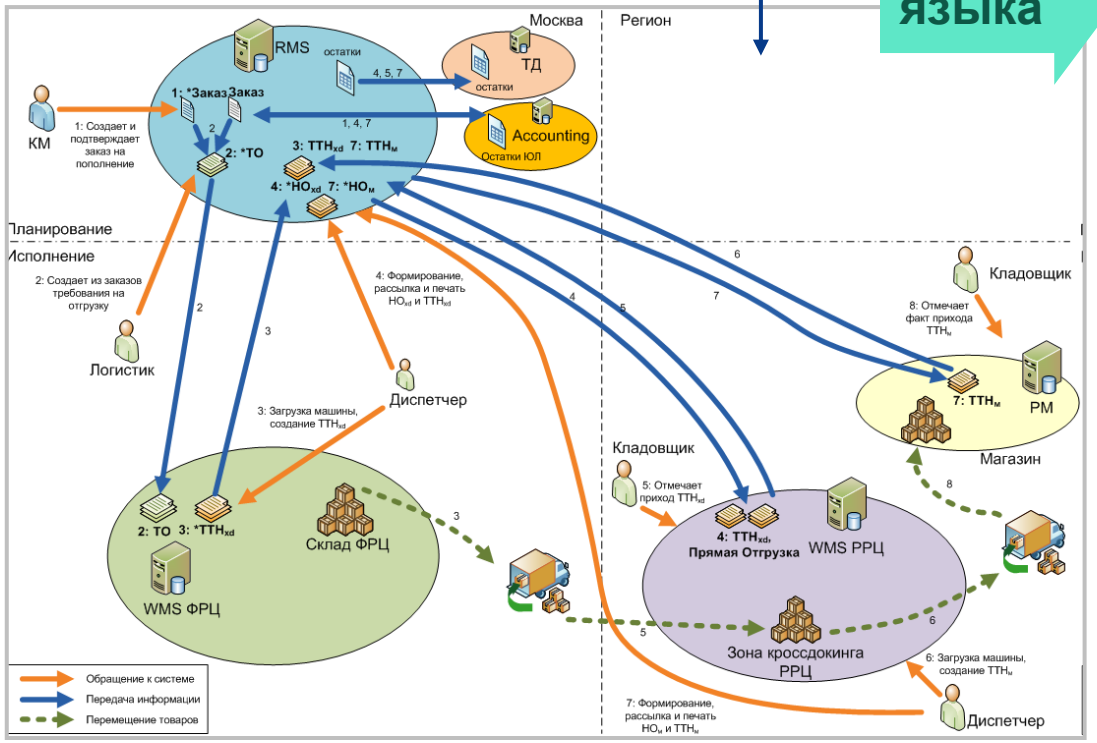
Бизнес меняется,
ERP дорабатывается —
в проекте надо показать,
как догоним развитие.

Не всегда вектор развития
ERP и нового проекта
соответствуют.

СМД-методология. Схема шага развития (одна из рисовок)

Формализация снабжения магазинов

Неформальная схема деятельности и её отражение в существующих системах

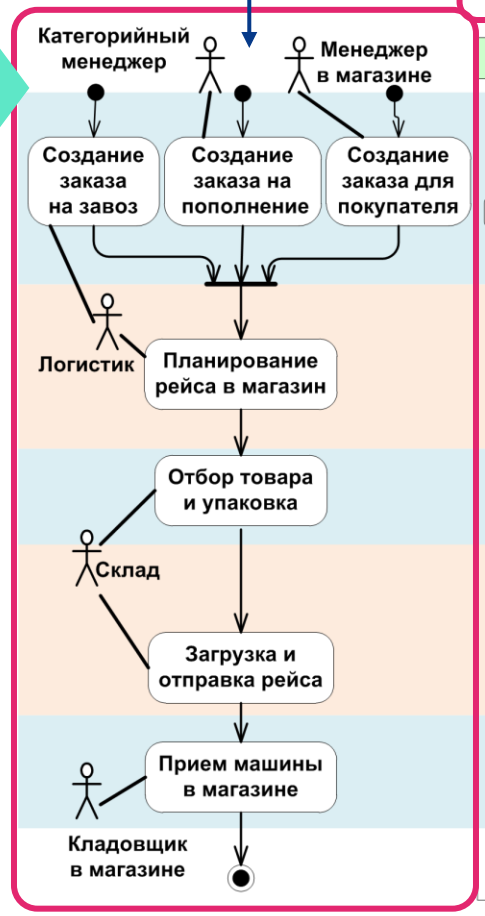


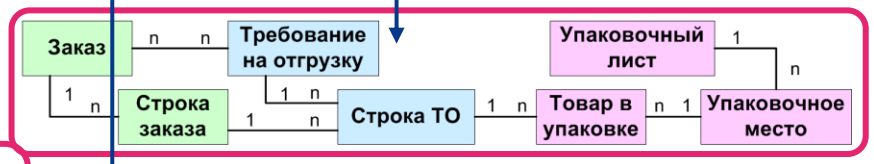
Смена языка

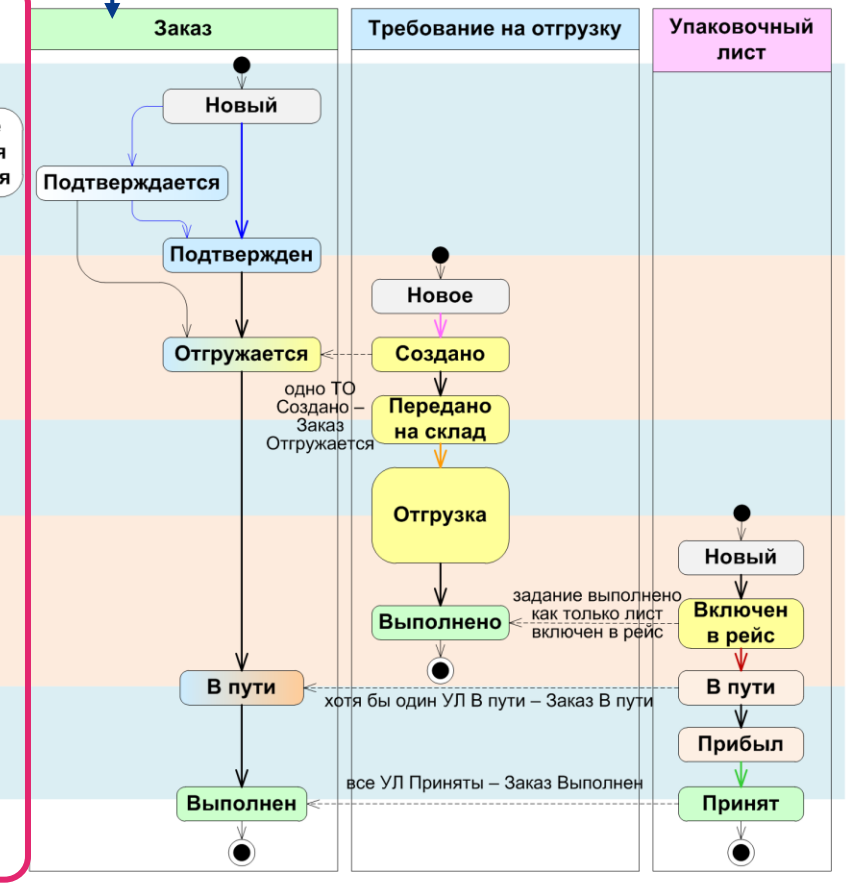
Состояния документов — State Diagram

Объекты — Class Diagram

Бизнес-процесс — Activity Diagram







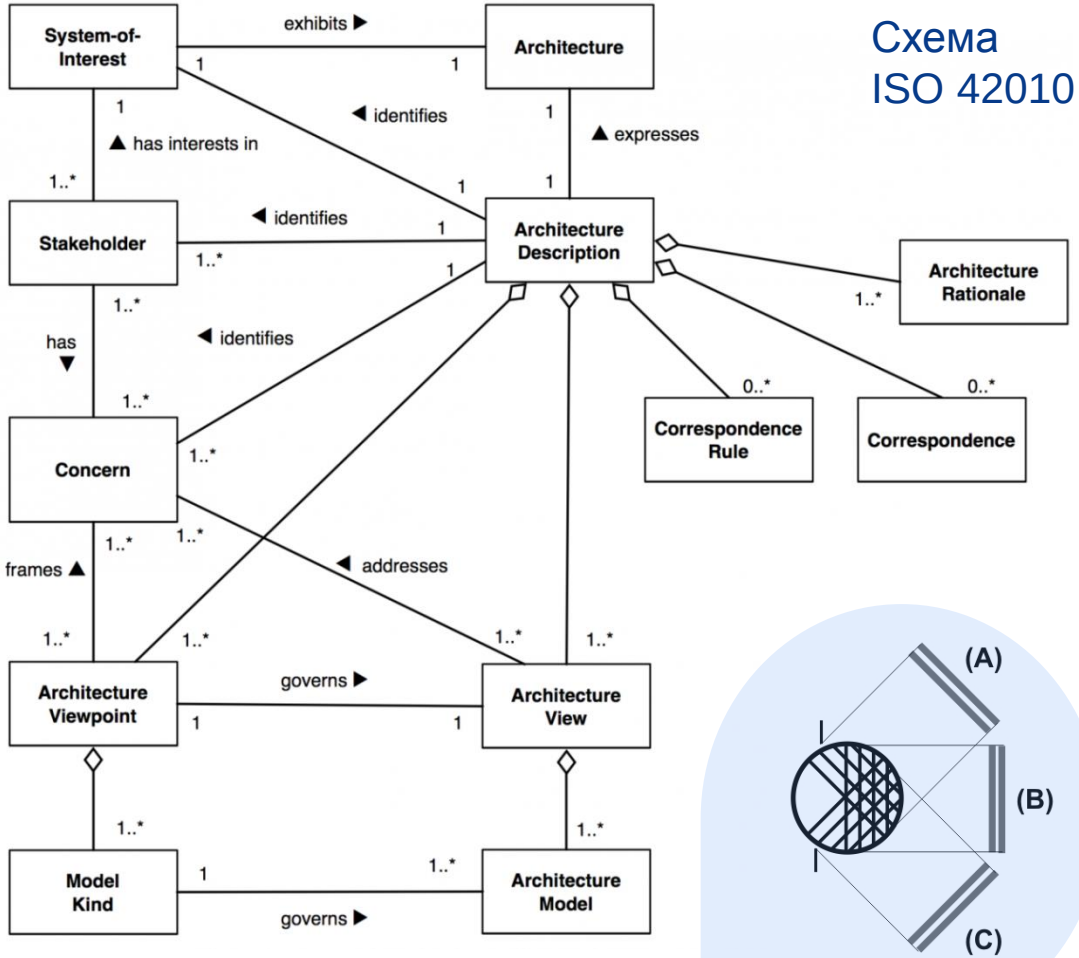
Многоплановый взгляд на системы

Система представляется через множество планов, viewpoint, которые имеют своего адресата и назначение.

- **Два взгляда на систему:** снаружи из надсистемы и устройство внутри.
- **Три схемы устройства системы:** функциональная, модульная и размещения, исполнение функции может быть локализовано в конкретных модулях или распределено в виде аспекта.
- **Выделение системных уровней:** архитектура бизнеса, приложений и техническая в архитектуре приложений, или UI, бизнес-логика и хранение, или фронт и бэк.

ISO 42010 представление через viewpoint — раскрытие схемы многих знаний

Схема ISO 42010

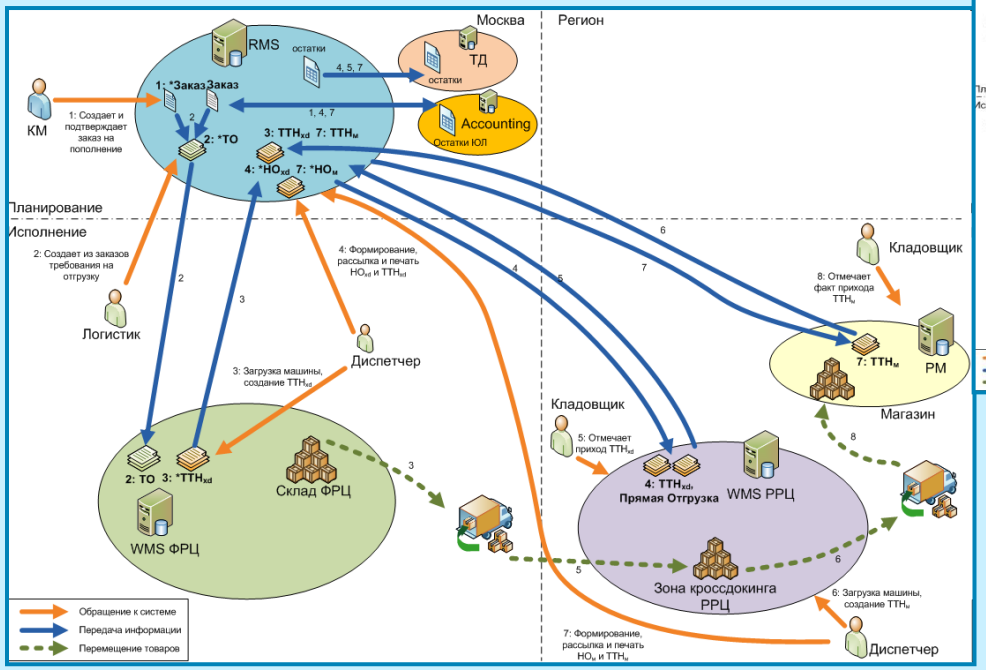


Проверка в неформальной модели

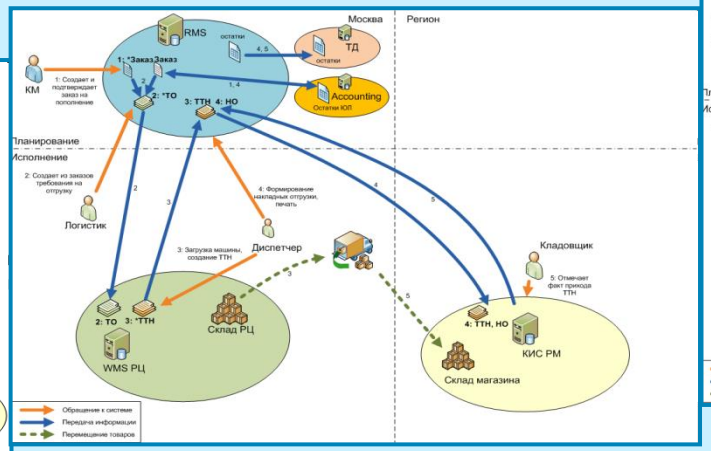
Мы можем построить формальную модель процесса и его реализации, но не всегда заказчик может её проверить. Часто он доверяет, а на внедрении вскрываются проблемы.

Решение — вернуться в неформальную модель или показывать прототипы.

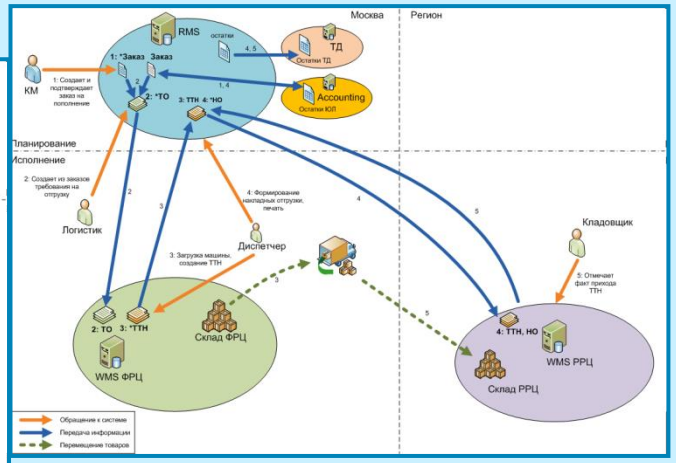
1



2



3



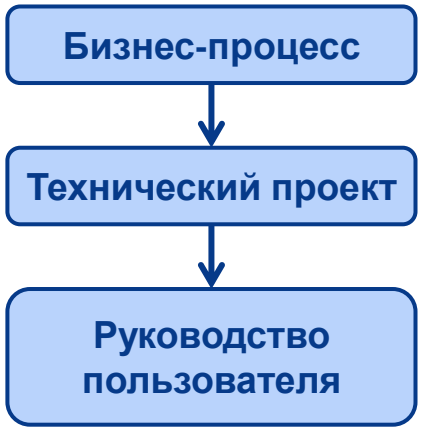
Снабжение магазинов: на неформальных моделях показывали кейсы работы будущей системы

Софт и пользователи

Классический подход

Бизнес и софт — разные системы, поэтому создаём несколько отдельных документов:

- описание бизнес-процессов, которые система поддерживает;
- руководство пользователя по работе с системой;
- технический проект системы: при проектировании именно показывал связь и объяснял, как система будет поддерживать бизнес-процессы.



Проблема. Системы сильно связаны, а описания живут независимо, устаревают и перестают соответствовать друг другу:

- руководство пользователя описывает экраны, связь с процессом неочевидна;
- описание бизнес-процессов обычно сильно отстаёт от фактической работы;
- технический проект оказывается ориентирован на админов.

Способ борьбы — ограниченное описание бизнес-контекста: user story, или use case, или context diagram, включённые в описание ИТ-системы.

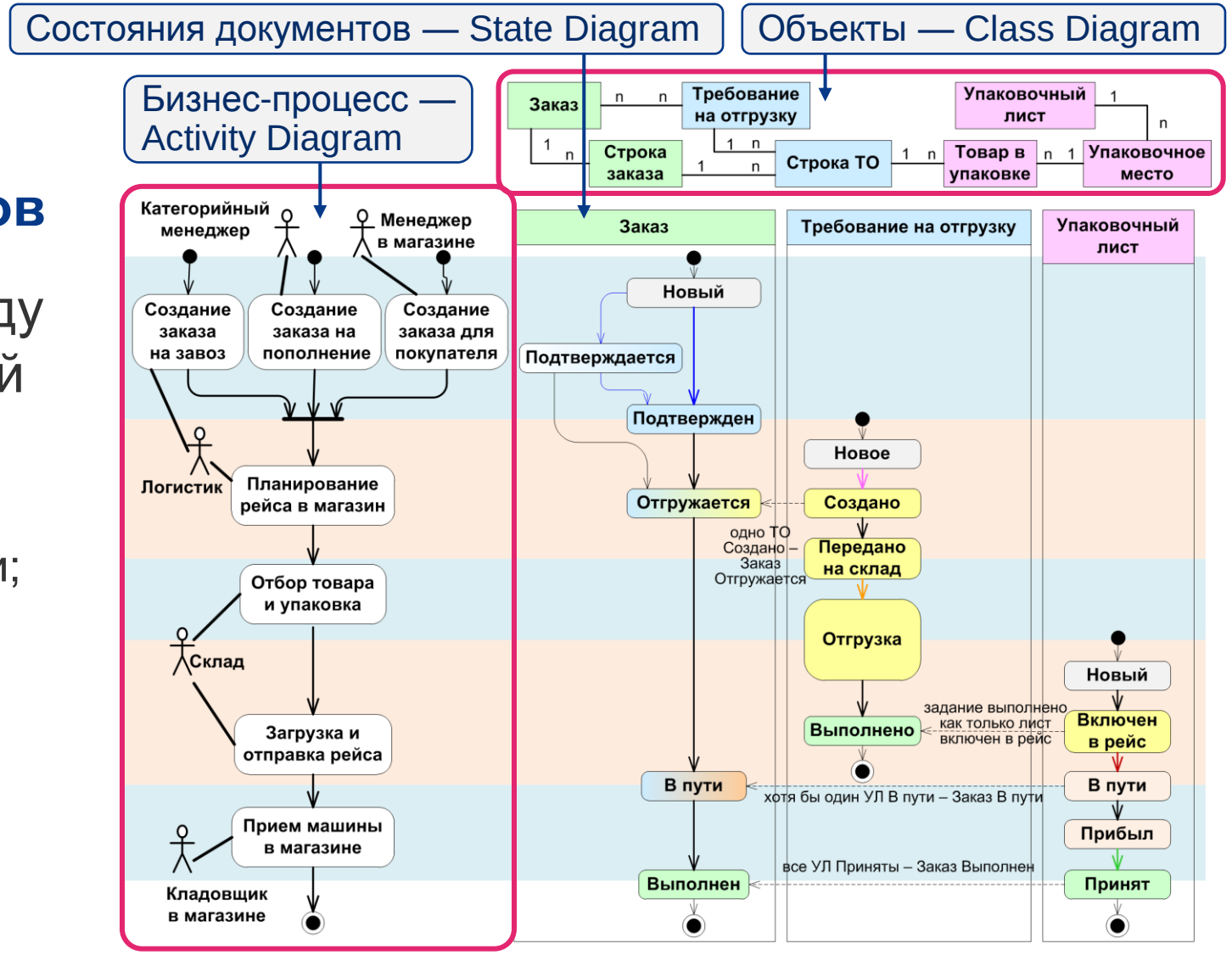
Domain Driven Design

Пример: снабжение магазинов

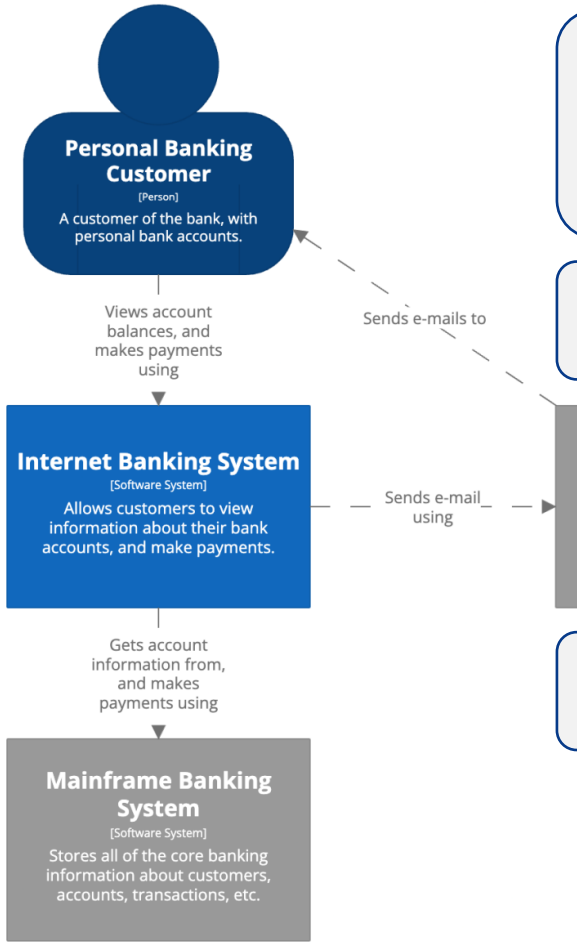
Прозрачное соответствие между бизнес-моделью и реализацией в ИТ-системе:

- классы системы соответствуют документам предметной области;
- workflow документов реализует бизнес-процесс.

Можно обсуждать изменения непосредственно на модели системы.



Контекстная диаграмма C4 Model — система в окружении

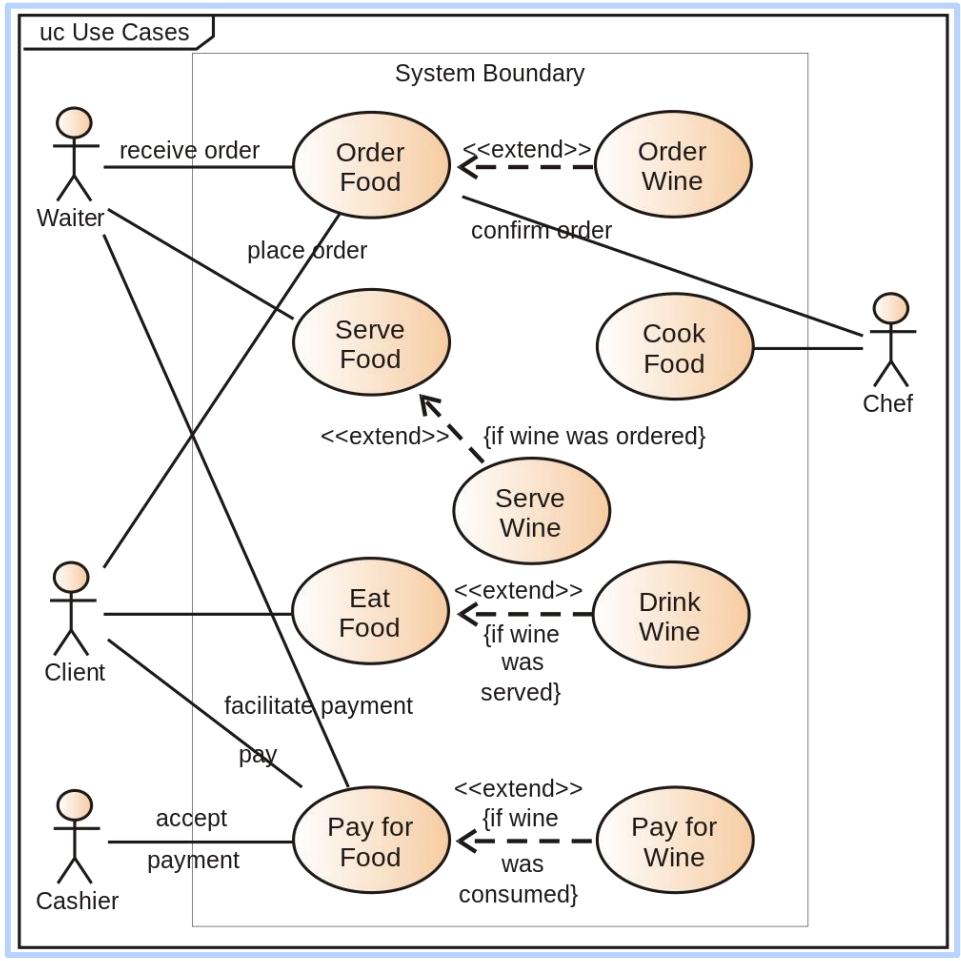


В центре — наша система, вокруг — пользователи и взаимодействующие системы

Стрелки — акты взаимодействия

Множество внешних систем — смена парадигмы на сервисы

По сути, это свёрнутая use case диаграмма: вместо овалов — стрелки с подписями



Представляем пользователя!

Запрос на доработку. Сейчас при вводе оплаты система автоматически привязывает её к первому не закрытому договору клиента. Надо уметь указывать договор при вводе платежа, так как клиент может внести предоплату по новому договору, не закрыв старый, чтобы ему отгрузили новый товар.

Предложено решение: убрать заполнение договора в сервисе создания платежей, добавить поле договора на форму ввода платежа со списком незакрытых договоров, и если там один договор — заполнять поле автоматически.

Вопрос тестировщика: а как бухгалтер, вводя платёж, узнает, по какому он договору?

- Если это написано в тексте основании платежа — то можно определить, а если нет?
- Можно спросить у менеджера, но платежи бухгалтеру надо ввести быстро — он не спросит.
- В любом случае, решение, когда для ввода надо спросить другого пользователя, сомнительное.

Хорошо, если вопрос возникнет на этапе оценки постановки, а не после реализации.

Указание пользователя конкретизирует встройку в надсистему, указывая конкретный элемент, и мы проверяем работоспособность связи — компетенции пользователя, необходимые для её работы.

Процесс в целом: выходим за рамки системы

→ Если работать только с взаимодействиями пользователя, явно относящимися к системе, то не все шаги бизнес-процесса могут быть поддержаны.

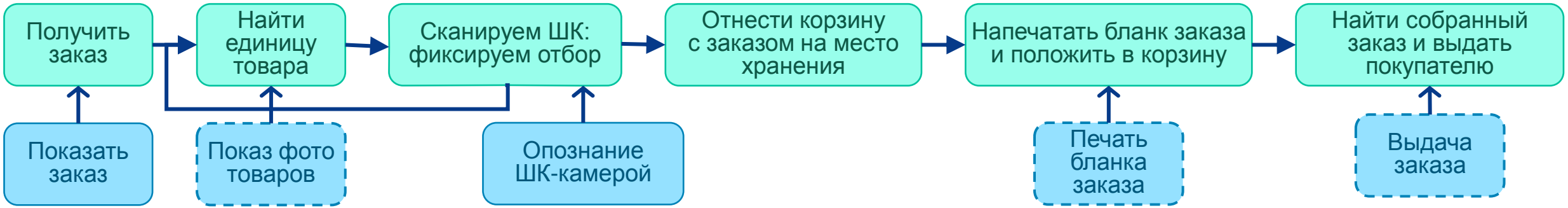
Запрос на доработку. Поддержать на мобильном рабочем месте отбор одежды по заказу в торговом зале магазина.

Очевидное решение — два шага: получить заказ и фиксировать товар поштучно.

Проблемы:

- Как продавцу быстро найти нужное? Модели и цвета похожи, смотреть этикетки — долго;
- После отбора заказ надо куда-то положить, чтобы потом выдать. Как?

Проблемы видны, если представить процесс отбора по шагам и подумать, какая поддержка нужна в системе для каждого шага процесса.



Дружественный интерфейс

Репрезентативные данные при тестировании:

- смотреть, что размеры полей достаточны для представления данных;
- проверять, что выводится содержательная информация: если в списке документов 90% названий вида «Письмо 45678 от 25.07.22», то форма не даст быстро найти документ;
- знать характерные размеры таблиц в документах, проверять работу на них.

Позволять копировать номера и фрагменты текста, чтобы вставить на другие формы, в идеале — встроить переход на связанные объекты.

Придумывать workaround: таблицу из 10к строк обычно не смотрят глазами, но может быть нужен экспорт, и его удобно сделать без дочитки.

Понимать, что какие-то формы пользователь использует, общаясь по телефону — в разговоре трудно передать длинный номер или сложный код.

Это всё — конкретные советы из моего опыта. Но источник — представление себя на месте пользователя. Лучше — на этапе проектирования или тестирования, а не когда пользователи говорят «неудобно».

Устойчивость к сбоям и пикам нагрузки и другие аспекты качества в микросервисной архитектуре

Что нам принесли микросервисы?

- Уход от единой СУБД приложения, использование NoSQL-баз.
- Кластерное развёртывание с независимым хранением на узлах.
- Транзакционность и консистентность при обработке запроса обеспечиваются в приложении, а не в базе данных.
- При восстановлении узла кластера данные не консистентны.
- Каждый бизнес-запрос обрабатывает много сервисов.
- Много экземпляров одного сервиса для масштабирования, экземпляры сервисов падают по ошибкам или блокировкам.
- Асинхронные сообщения, очереди выравнивают производительность — сообщения остаются в очереди, даже если запрос отменен.

Для тестирования устойчивости к сбоям и нагрузкам надо хорошо понимать структуру приложения, включая взаимодействие и масштабирование сервисов, способы их взаимодействия и организацию очередей.

Тестировать сервисы надо иначе

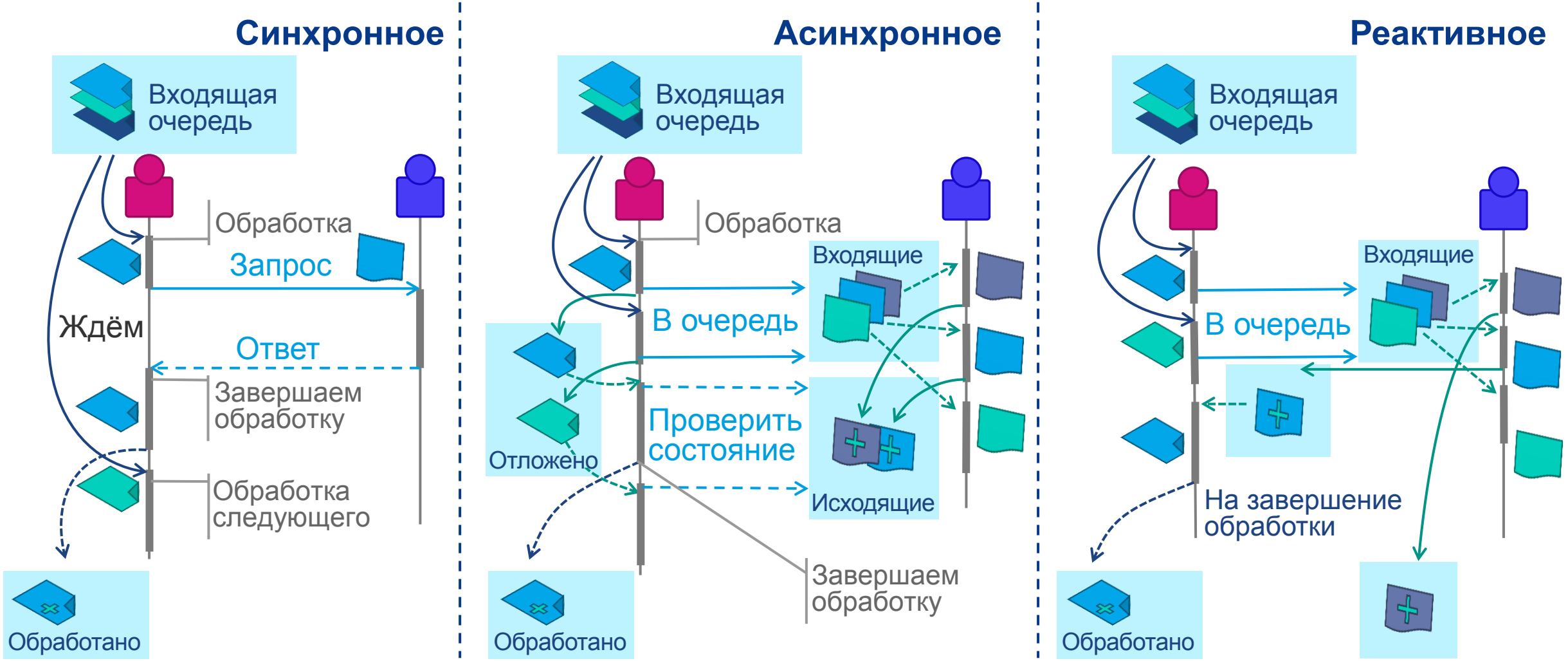
Классические архитектура и тесты рассчитаны на **атомарное исполнение**: если в интерфейсе и базе данных ожидаемое — функция работает.

В сервисной новой архитектуре приложений это не так:

- обработка организована через асинхронные сообщения;
- постановка в очередь не даёт гарантий обработки;
- при распределённой обработке операция может быть выполнена частично.

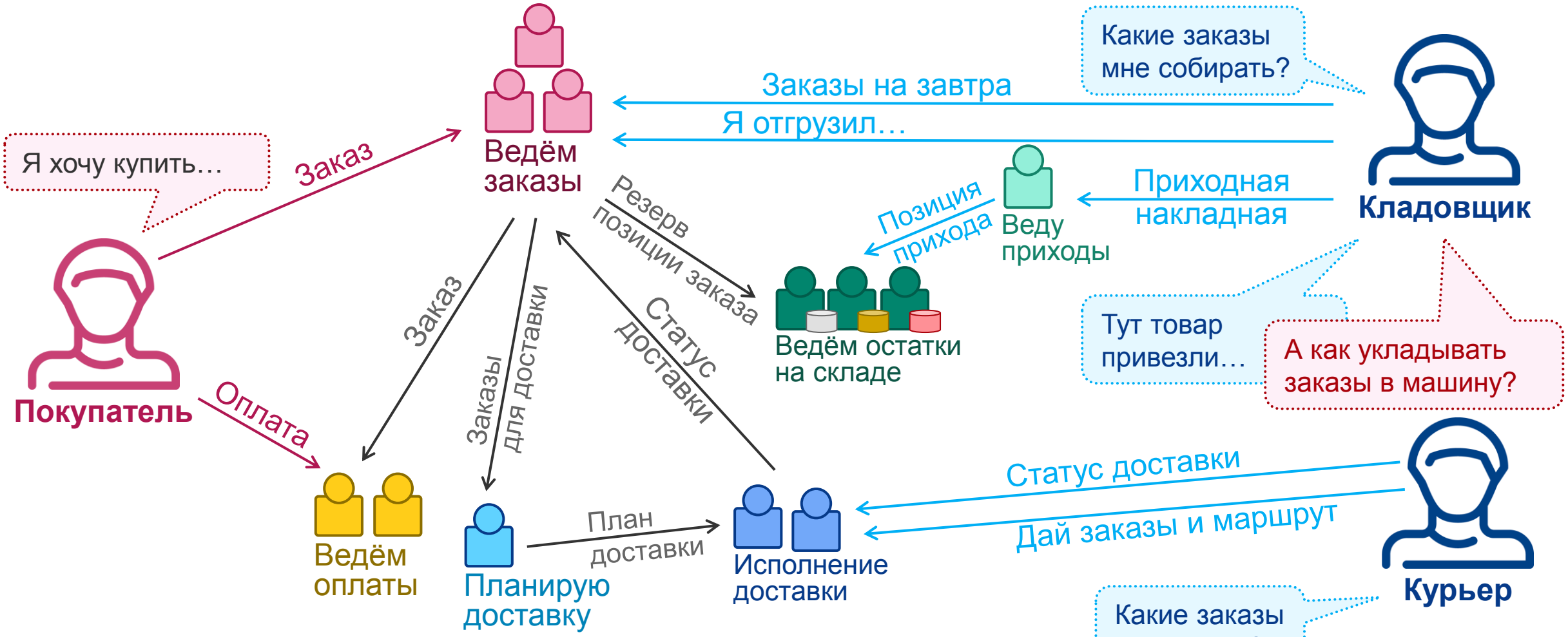
Надо понимать устройство и прорабатывать сложные случаи сбоев.

Взаимодействие сервисов



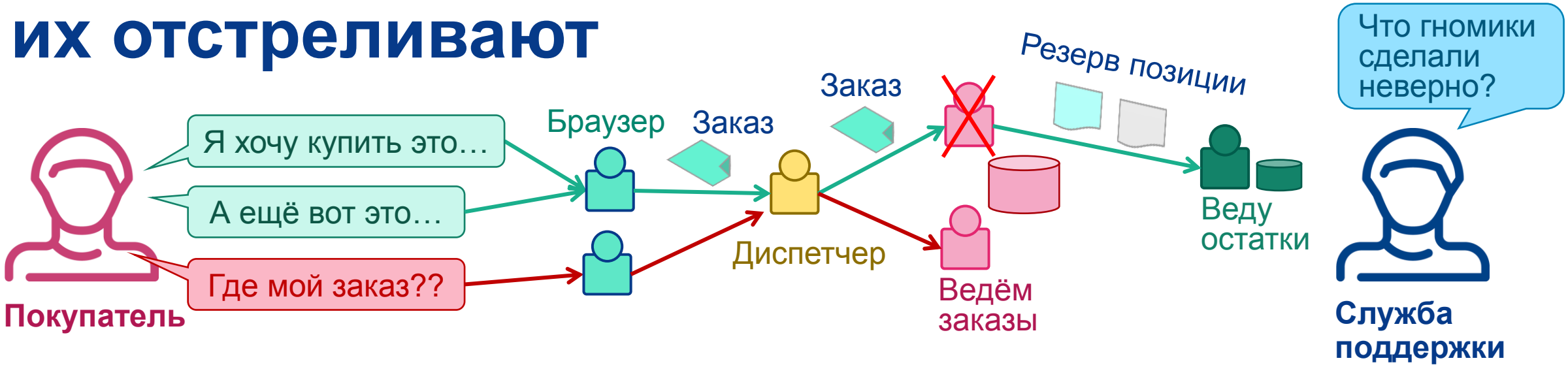
Это — для пары сервисов. А надо представить всю структуру обработки каждого запроса, возможные сбои и поведение при росте нагрузки.

Модель для сервисной архитектуры



Смотри доклады «Модели приложения для разных парадигм программирования» (SQA Days #27, осень 2020) и «Визуальное проектирование масштабируемых приложений» (TechLead-2021)

Устойчивость: гномики умирают, их отстреливают



Ситуация: идёт обработка и резервирование заказа, и в этот момент:

- инстанс, ведущий резервирование, падает или его убивают;
- покупатель долго не видит ответа в браузере — и открывает новый…

Покупатель не обязательно авторизован — до этого могло не дойти.

Покупатель может ехать в «Сапсане» или в месте с плохой связью.

Ситуация может быть и при оплате — там чужой платёжный шлюз.

Модель гномиков: в чём системное мышление?

Модели гномиков не было, были частные задачи:

- покажите, как система будет работать, если отвалится один дата-центр?
- как сделать, чтобы время отклика на двух дата-центрах было приемлемо?
- фискальные регистраторы часто падают, особенно с новой прошивкой. Как уменьшить количество случаев, требующих ручного разбора?
- И другие подобные…

В каждом случае надо было рисовать понятные диаграммы, проигрывать на них процессы, ставить релевантные эксперименты. Системное мышление помогало этому.

А модель гномиков — результат обобщения частных моделей.

Компетенции мышления

1

- Уметь мыслить абстракциями и обобщениями, а также наборами фактов;
- Размечать при восприятии тексты и картинки, соотнося их со своими моделями мира и дорабатывая свои модели;
- Создавать новые ментальные объекты — абстракции, концепты и модели;
- Использовать известные, наработанные способы мышления.

2

- Структурированно и понятно описывать свои модели для других;
- Сопоставлять абстракции и модели с реальным миром, понимать зазор.

3

- Фокусироваться на задаче;
- Рефлексировать ход своего мышления и работать над ошибками;
- Знать типовые проблемы мышления — когнитивные искажения, влияние эмоций, и уметь мыслить с их учётом;
- Использовать экзокортекс: компьютер, бумагу, таблицы, схемы.

Мышление тренируемо, но путь развития надо проектировать: всем и сразу овладеть невозможно.

Итоги

Системное мышление помогает:

- Разобраться со сложными конструкциями, построить модель;
- Видеть за структурной схемой динамику работы;
- Увидеть за моделью реальный мир, оценивать разрыв между моделью и миром и учитывать его, принимая решения;
- Видеть устройство бизнеса и места софта в нём, представить пользователя;
- Разбираться в том, как софт устроен внутри, как реально работает, сделать адекватный план тестирования.

Вопросы по докладу и обратная связь

Надеюсь, рассказ вас убедил и вы решите познакомиться глубже!

Максим Цепков

http://mtsepkov.org

@MaximTsepkov

На сайте много материалов по анализу и архитектуре, Agile и менеджменту самоуправления, моделям soft skill, мои доклады, статьи и конспекты книг

Вакансии

Пишите на hr@custis.ru, подходите с вопросами!