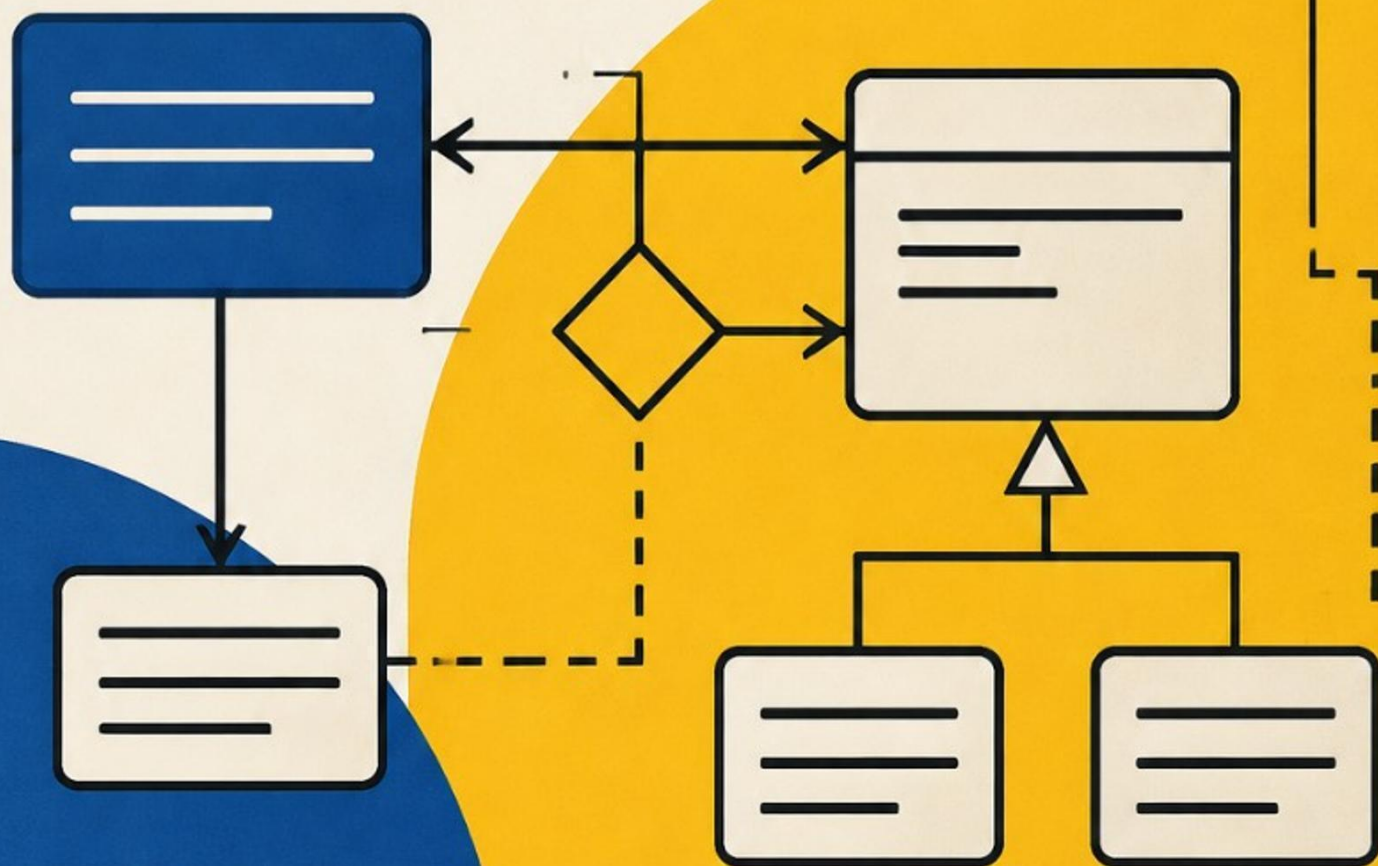


The logo for the University of Michigan Library (UML) features the letters 'UML' in a bold, dark blue, sans-serif font. A bright yellow vertical bar is positioned behind the letter 'L', extending from the top to the bottom of the text.

Как проектировать системы и объяснять их с помощью диаграмм



III

Софья Вечерняя

Софья Вечерняя

**UML без догадок:
Как проектировать
системы и объяснять их
с помощью диаграмм**

«Автор»

2026

Вечерняя С.

UML без догадок: Как проектировать системы и объяснять их с помощью диаграмм / С. Вечерняя — «Автор», 2026

Одна стрелка на диаграмме — и заказчик уже понял одно, разработчик другое. Модель, которая должна прояснять систему, оборачивается спором, ошибками и переделками. «UML без догадок» помогает превратить схемы в общий язык команды. Выбирайте диаграмму под вопрос и аудиторию, очерчивайте границы системы, переводите требования в проверяемые сценарии. Обосновывайте классы и связи правилами предметной области, описывайте процессы, сообщения и состояния. Сверяйте требования с поведением и архитектурой — и не рисуйте то, что проще объяснить словами. Для аналитиков, разработчиков, архитекторов и всех, кому важно проектировать системы и объяснять решения. Книга поможет выявлять скрытые допущения, согласовывать поведение и сверять модель с реализацией — от первого обсуждения до сопровождения. Книга создана с помощью ИИ.
Обложка: GPT Image - Лицензия

Содержание

UML как язык проектирования: модели, диаграммы и границы применимости	6
Прежде чем рисовать: цель модели, аудитория, масштаб и выбор диаграммы	11
Сценарии использования: акторы, границы системы и функциональные требования	17
Диаграммы деятельности: процессы, условия, параллельность и ответственность	24
Конец ознакомительного фрагмента.	27

Софья Вечерняя

UML без догадок: Как проектировать системы и объяснять их с помощью диаграмм

UML без догадок: Как проектировать системы и объяснять их с помощью диаграмм

Софья Вечерняя

UML как язык проектирования: модели, диаграммы и границы применимости

В 8:42 утра на экране переговорной появился «Тёплый маршрут» — сервис, который должен помогать доставлять лекарства жителям удалённых посёлков. Диаграмма занимала всю стену. На ней были прямоугольники, овалы, подписи мелким шрифтом и линии, похожие на следы, оставленные участниками особенно оживлённого спора.

— Значит, как только заявка принята, жителю приходит сообщение, — сказала Марина, представитель заказчика.

— Нет, — ответил Илья, разработчик. — Эта стрелка обозначает внутреннее событие. Уведомление отправляется позже, если доступен шлюз сообщений.

Зоя, диспетчер, наклонилась к экрану:

— А я решила, что маршрут обновится сам. Если мост размыло, кто скажет курьеру ехать в объезд?

Архитектор Лев увеличил схему. Стрелка, о которой спорили, действительно соединяла заявку с уведомлением, но рядом с ней стояли ещё две подписи, а начало линии терялось среди других связей.

Никто не ошибался намеренно. Просто каждый читал в схеме то, чего ждал от системы. Заказчик видел обещание, разработчик — техническое событие, диспетчер — порядок действий. Перед командой лежала одна диаграмма, но в действительности — несколько разных объяснений.

— Мы пытались показать всё сразу, — признался Илья.

На стене это «всё сразу» выглядело внушительно. И почти ничего не объясняло.

Четыре слова, которые не стоит путать

В разговоре о UML часто смешивают четыре понятия: систему, модель, диаграмму и нотацию. Различить их — не академическая прихоть, а способ перестать спорить о разных вещах под видом одной.

Система — то, что существует и работает. «Тёплый маршрут» — не только приложение. Это ещё медработники, диспетчеры, курьеры, аптеки, правила выдачи лекарств, состояние дорог, мобильная связь и решения людей в нестандартных ситуациях. Если описать только программные модули, можно получить верную картину кода и неверную картину услуги.

Модель — выбранное описание системы, построенное ради конкретной цели. Она не копирует реальность целиком, а оставляет только то, что помогает разобраться в вопросе. Карта посёлка показывает дороги, но не передаёт запах хвои, наклон каждого склона и привычки собак у ворот. Иначе она была бы не картой, а попыткой перенести весь посёлок на бумагу.

Диаграмма — один из способов показать модель, её вид или часть. В UML диаграмма может описывать структуру, поведение, взаимодействия — но всегда с некоторой точки зрения. Чтобы понять систему, иногда нужно несколько согласованных диаграмм, а не одна гигантская.

Нотация — правила, по которым устроены условные знаки и связи. UML задаёт общий словарь и грамматику: например, форму элементов и смысл некоторых типов линий. Важно, что линия в UML — не просто «между этими двумя штуками есть какая-то связь». Её значение зависит от вида диаграммы и обозначений на концах. Если правила не соблюдены или детали не читаются, зритель вынужден угадывать.

UML — язык моделирования, а не метод управления проектом и не волшебный переводчик с замысла на работающий продукт. Он помогает выражать структуру и поведение системы

в достаточно общем для команд понимании. Но правильный символ не делает верным допущение. Аккуратная диаграмма может убедительно описывать систему, которой никто не собирается строить.

Модель не заменяет требования: она может помочь уточнить их, но не отвечает за заказчика на вопрос, какую услугу тот хочет получить. Она не заменяет код: диаграмма компонентов не доказывает, что система запускается. Она не заменяет разговор с пользователем: ни один овал не расскажет, как фельдшер действует, когда связь пропадает. И, наконец, она не заменяет профессиональное суждение. UML не принимает решение о том, можно ли безопасно поручить алгоритму распределение лекарств или где нужна человеческая проверка.

У моделей есть ещё одна важная особенность: они избирательны. Умолчание может быть полезным — если оно осознанное. И опасным — если участники не заметили, что трактуют пробел по-разному.

Один сервис — три полезных взгляда

На совещании команда решила не улучшать огромную диаграмму, а сначала выбрать вопросы, на которые должна ответить модель. Оказалось, что таких вопросов как минимум три, и каждый требовал собственного ракурса.

Заказчику важно понять границы услуги. За что отвечает «Тёплый маршрут», а где начинается ответственность аптеки, медработника или курьера? На этом уровне полезна модель вариантов использования: граница сервиса, участники снаружи и действия, ради которых они с ним взаимодействуют.

В такой модели фельдшер может передать заявку, диспетчер — проверить её и организовать доставку, курьер — принять поручение и сообщить результат, житель — получить лекарство. Эти действия помогают обсуждать обязанности и обещания. Но схема сама по себе не означает, что заявка автоматически подтверждает наличие лекарства или что житель получает сообщение на телефон. Если это важно, нужно прямо обозначить такое правило или признать его открытым вопросом.

Именно здесь заказчик может спросить: «Когда мы вправе сказать жителю, что лекарство будет доставлено?» Хорошая модель не прячет ответ в стрелке. Она помогает команде обнаружить, что слово «принято» может означать регистрацию заявки, подтверждение наличия препарата или гарантию доставки. Это три разных обещания.

Разработчику важно понять взаимодействие во времени. Что происходит после того, как медработник отправил заявку? В каком порядке система проверяет данные, обращается к другим участникам и сообщает результат? Здесь полезна диаграмма последовательности: она показывает участников взаимодействия и сообщения между ними.

Например, команда может разобрать базовый сценарий так:

Шаг

Кто действует

Что происходит

1

Фельдшер

Передаёт заявку через интерфейс сервиса

2

Сервис заявок

Проверяет обязательные сведения и регистрирует запрос

3

Диспетчер

Просматривает заявку и принимает решение об организации доставки

4

Сервис и диспетчер

Фиксируют назначение курьера

5

Курьер

Сообщает о передаче лекарства или о том, что доставку выполнить не удалось

У этой последовательности есть практическая ценность: команда видит, где находится человек, где действует программа и какие результаты нужно передавать между ними. Но даже такая схема оставляет вопросы. Что считать обязательными сведениями? Что делать, если сервис недоступен? Кто и как проверяет наличие лекарства? Диаграмма не обязана ответить на всё сразу. Она обязана сделать видимым то, что относится к её цели.

Архитектору важно понять устройство решения. Какие крупные части есть внутри сервиса, что с чем взаимодействует и какие внешние системы задействованы? Здесь уместна модель компонентов. На ней можно показать интерфейс диспетчера, сервис заявок, модуль планирования маршрутов, хранилище данных и интеграцию с внешним сервисом сообщений. Такой вид помогает обсуждать разделение ответственности, зависимости и места возможных изменений.

Но диаграмма компонентов не показывает автоматически, на каких машинах работают компоненты, как система переживает отказ сети и сколько запросов выдерживает. Для этих вопросов нужны другие модели, данные и проверки. Попытка заставить компонентную диаграмму ответить ещё и на вопросы развёртывания, задержек и ежедневной работы диспетчера не делает её универсальной. Она делает её перегруженной.

Три взгляда описывают один сервис, но не повторяют друг друга. Первый проясняет ответственность, второй — ход сценария, третий — устройство решения. Они могут быть связаны и согласованы, но каждый должен быть достаточно самостоятельным, чтобы его можно было прочитать без экскурсии автора.

Паспорт модели: пять вопросов до рисования

Чтобы диаграмма не превратилась в собрание пожеланий, команда завела простой инструмент — **паспорт модели**. Его заполняют перед тем, как выбирать символы.

Поле паспорта

Что нужно решить

Цель

На какой конкретный вопрос должна ответить модель?

Аудитория

Кто будет её читать и какие решения принимать?

Границы

Что включено в систему, а что остаётся снаружи?

Допущения

Что мы пока считаем верным, хотя ещё не подтвердили?

Уровень детализации

Насколько глубоко описываем структуру или поведение?

Для первой модели «Тёплого маршрута» паспорт мог бы выглядеть так:

- **Цель:** согласовать, кто участвует в базовой доставке и где заканчивается ответственность сервиса.
- **Аудитория:** заказчик, руководитель диспетчерской и команда разработки.
- **Границы:** от передачи заявки медработником до фиксации результата доставки.
- **Допущения:** диспетчер участвует в организации доставки; курьер сообщает результат, а сервис его фиксирует.
- **Уровень детализации:** основные участники, действия и исходы, без технического устройства модулей.

В паспорте важно не только то, что модель включает. Не менее важно, что она пока не описывает: паспорт не позволяет выдать умолчание за принятое решение. Например, если модель не показывает проверку аптечного запаса, это не значит, что команда решила считать лекарства всегда доступными. Это значит, что вопрос отложен, и его нужно будет решить отдельно.

Такой подход согласуется и с тем, как люди воспринимают информацию. Рабочая память ограничена: когда на одном изображении слишком много элементов и отношений, читателю приходится удерживать в уме сразу несколько контекстов. Разбивка на смысловые части и группировка по задаче помогают ориентироваться. Это не означает, что существует универсальное «правильное» число блоков на диаграмме. Но если участник встречи снова и снова водит пальцем по экрану, разыскивая нужную линию, проблема, вероятно, уже не в его внимательности.

Первая модель — и её честные пробелы

Команда выбрала вопрос: «Кто участвует в базовом сценарии, и за какой результат отвечает “Тёплый маршрут”?» Для начала составили компактную модель границ сервиса.

Участник

Действие в рамках базового сценария

Результат

Фельдшер

Передаёт заявку на доставку

Запрос поступает в сервис

Диспетчер

Проверяет заявку и организует доставку

Назначение фиксируется

Курьер

Выполняет поручение и сообщает результат

Передача лекарства или неудача доставки отмечена

Житель

Получает лекарство

Факт получения подтверждается согласованным способом

Это ещё не большая диаграмма — и в этом её достоинство. В терминах UML такую модель можно оформить как диаграмму вариантов использования: обозначить границу сервиса, вынести участников наружу, а их цели показать внутри. Таблица уже помогает проверить смысл, прежде чем команда начнёт выбирать графические элементы.

В ней намеренно не решены несколько важных вопросов:

- как проверяется рецепт и какие сведения обязательны в заявке;
- кто подтверждает наличие лекарства и как это подтверждение связано с аптекой;
- как выбирается маршрут и что происходит при перекрытой дороге;
- какие статусы видит житель, диспетчер и курьер;
- как сервис работает при потере мобильной связи;
- какие персональные данные собираются и как долго хранятся;
- как система должна быть развёрнута и какие отказы обязана выдерживать.

Это не забытые детали. Это вопросы за пределами выбранной модели, которые команда должна рассмотреть в подходящих требованиях, обсуждениях или других моделях. Если позже окажется, что один из них критически важен для границы ответственности, паспорт придётся обновить — и это нормальная работа, а не признание поражения.

Проверка перед тем, как показать диаграмму

Перед тем как вынести модель на встречу, спросите себя:

1. Какой вопрос она решает? Если ответ звучит как «показывает всю систему», сузьте задачу.

2. Для кого она создана? Заказчику, разработчику и диспетчеру могут понадобиться разные виды.

3. Что включено в границы? Назовите важные внешние системы, роли и процессы, которые остались снаружи.

4. Какие допущения сделаны? Отделите подтверждённые факты от временных договорённостей.

5. Какой уровень детализации выбран? Не смешивайте обзор услуги с внутренним устройством метода или расписанием сообщений.

6. Что модель намеренно не объясняет? Запишите это, чтобы читатель не заполнял пробел собственной догадкой.

Небольшое упражнение: выберите знакомую вам систему — например, запись к врачу, оформление командировки или доставку продуктов. За пять минут заполните паспорт модели для одного конкретного вопроса: скажем, «кто отвечает за перенос записи?» Затем перечислите три вещи, которые вы сознательно не станете включать в первую модель. Если после этого вопрос стал яснее, значит, проектирование уже началось — ещё до появления первого овала.

На следующем совещании команда «Тёплого маршрута» не стала увеличивать старую схему. Вместо этого участники согласовали границу сервиса, выбрали базовый сценарий и записали допущения. Новая модель была меньше прежней и отвечала лишь на один вопрос. Зато заказчик, разработчик и диспетчер прочли её одинаково.

UML ценен не количеством символов на экране, а точностью разговора, который эти символы позволяют вести. Хорошая модель не притворяется территорией. Она служит картой для конкретного путешествия — и честно отмечает, какие места пока не нанесены.

Прежде чем рисовать: цель модели, аудитория, масштаб и выбор диаграммы

В 8:17 диспетчерская уже гудела, хотя рабочий день только начался. На экране одна за другой появлялись заявки на доставку лекарств, требующих температурного режима. Большинство проходило быстро. Но некоторые застревали в статусе «Ожидает назначения» — на час, на полдня, иногда дольше.

— Мы не понимаем, что именно там происходит, — сказала диспетчер Ирина. — Клиника думает, что мы забыли про заявку. Мы думаем, что ждём машину. А иногда выясняется, что не хватает данных. Статус один, причины разные.

На рабочей встрече разработчик предложил диаграмму классов. Руководитель продукта — диаграмму вариантов использования. Архитектор добавил диаграмму компонентов. Кто-то вспомнил о диаграмме деятельности. За минуту на доске возник целый каталог UML, а исходная жалоба всё ещё ждала ответа.

Так часто начинается моделирование: не с карандаша, а с соблазна нарисовать всё. Однако диаграмма — не витрина значков. Это инструмент, который должен помочь кому-то принять решение, разобраться в ситуации или согласовать границы работы. Если сначала выбрать значки, а потом искать для них вопрос, получится красивая карта, по которой никто не собирался идти.

Одна система — разные вопросы

Команда работает над «Тёплым маршрутом» — сервисом доставки термочувствительных лекарств в клиники. На встрече присутствуют три человека с разными задачами:

- диспетчеру нужно понять, где и почему задерживается заявка;
- разработчику важно разобраться, какие данные и связи нужны для назначения транспорта;
- руководителю проекта нужно определить, что входит в систему, какие внешние участники задействованы и где возможны риски.

Это не три формулировки одного вопроса. Это три разных вопроса, а значит, и три разных ракурса.

Представьте, что вы попросили одного человека одновременно нарисовать схему метро, карту местности и план эвакуации. Все три изображения могут описывать одно здание или город, но отвечают на разные нужды. С UML так же: универсальная диаграмма, которая одинаково хорошо показывает рабочий процесс, структуру данных и границы проекта, обычно оказывается либо слишком общей, либо перегруженной.

Вместо немедленного рисования ведущая встречи задаёт первый вопрос:

— Какое решение должен помочь принять наш рисунок?

Ответ диспетчера звучит конкретно: «Хочу увидеть, на каких шагах заявка задерживается и что происходит, если подходящей машины нет».

Это уже рабочее направление. Но прежде чем выбирать диаграмму, нужно уточнить, для кого она и какой участок системы рассматривает.

Шаг 1. Сформулируйте вопрос, на который можно ответить

Фраза «нам нужна схема процесса» пока ничего не говорит о цели. «Нам нужна UML-диаграмма» — тоже. Попробуйте записать вопрос так, чтобы читатель мог проверить: дала ли модель ответ.

Например:

- «На каком этапе заявка чаще всего переходит в ожидание назначения?»
- «Какие данные нужны, чтобы подобрать машине подходящий температурный режим?»

- «Какие внешние системы и участники входят в проект, а какие остаются за его границами?»

Полезно отличать вопрос о том, **что происходит**, от вопроса о том, **из чего система состоит**. Первый тянет нас к процессу и поведению, второй — к структуре. Вопрос о том, кто взаимодействует с системой и ради каких целей, ведёт к границам и функциям.

Не всякий вопрос удастся окончательно решить диаграммой. Например, рисунок может показать, что несколько ветвей процесса сходятся к одному статусу. Но он не докажет, какая ветвь чаще вызывает задержку. Для этого понадобится статистика по заявкам, журнал событий или наблюдение за работой. Хорошая модель не подменяет данные: она помогает понять, какие данные искать и как интерпретировать.

Контрольная остановка. Допишите фразу: «После просмотра модели читатель сможет...» Если окончание звучит как «понять систему вообще», вопрос пока слишком широк.

Шаг 2. Назовите будущего читателя

Одна и та же схема для разных людей работает по-разному. Диспетчер знает реальные обходные пути и названия статусов. Разработчик понимает технические термины, но может не знать, почему оператор иногда звонит в клинику до появления автоматического предупреждения. Руководителю не нужно разбирать каждое поле заявки, зато важно видеть, кто отвечает за интеграции и что команда не обещает включить в первую версию.

Аудитория — это не только должность. Важно выяснить:

- что читатель уже знает;
- какое решение он должен принять;
- какими словами он пользуется в своей работе;
- будет ли он обсуждать модель с другими или использовать её как справочник;
- какие детали ему нужны, а какие станут помехой.

Если диспетчеру показать десяток технических классов, он потратит внимание на расшифровку модели вместо обсуждения задержек. Если разработчику показать только «клиника отправляет заявку, система назначает машину», останется неясно, какие именно данные влияют на назначение.

Визуальная модель снижает необходимость держать все связи в голове: читатель может посмотреть на схему, а не мысленно воспроизводить длинное объяснение. Но этот выигрыш исчезает, если на одном листе слишком много деталей, пересечений и условных обозначений. Рабочая память ограничена, и диаграмма не отменяет этого ограничения — она лишь помогает распределить нагрузку. Группировка, понятные подписи и умеренное число элементов обычно полезнее, чем попытка уместить «всё важное».

Контрольная остановка. Назовите одного главного читателя. Если приходится отвечать «всех понемногу», подумайте, не нужны ли вам несколько видов модели.

Шаг 3. Выберите масштаб и явно обозначьте границы

Масштаб отвечает на вопрос: какую часть действительности мы рассматриваем? Можно показать весь контекст услуги, один процесс обработки заявки или внутреннее устройство одного модуля. Эти уровни не взаимозаменяемы.

Для «Тёплого маршрута» границы можно задать так:

- **контекст системы:** клиника, диспетчер, перевозчик и сервис уведомлений взаимодействуют с «Тёплым маршрутом»;
- **сценарий или процесс:** от поступления заявки до назначения перевозчика и подтверждения доставки;
- **модель предметной области:** заявки, грузы, требования к температуре, транспорт и назначения;
- **архитектура:** программные компоненты и их взаимодействия с внешними сервисами.

Масштаб полезно задавать не только по охвату, но и по детализации. Если цель — найти задержку, можно показать важные действия, ветвления и ответственных, но не каждый щелчок мышью. Если цель — согласовать модель данных, понадобятся типы сущностей и связи, но не маршрут каждого звонка диспетчера.

Фиксируйте границы словами, прежде чем переносить их на бумагу. Например: «Рассматриваем текущий процесс от регистрации заявки до передачи её перевозчику. Доставку и подтверждение температуры в пути пока не анализируем». Такая оговорка не ослабляет модель, а защищает её от ложных ожиданий.

Здесь же полезно записать допущения и неизвестное. В случае «Тёплого маршрута» команда отмечает:

- в текущем процессе статус «Ожидает назначения» используется и при нехватке данных, и при отсутствии подходящего автомобиля;
- подбор машины может выполнять система или диспетчер вручную;
- частоту каждой причины задержки ещё нужно проверить по журналу заявок.

Последний пункт особенно важен. Если модель показывает предполагаемый процесс, обозначьте его как текущую гипотезу или черновик. Не позволяйте уверенной линии на диаграмме выдать предположение за установленный факт.

Шаг 4. Подберите диаграмму под задачу

Участники встречи возвращаются к исходным вопросам и выбирают не «любимый вид UML», а подходящий способ разговора.

Вопрос

Главная аудитория

Масштаб

Подходящая диаграмма

Где заявка задерживается и что происходит при отсутствии машины?

Диспетчеры и аналитики

Один процесс обработки

Диаграмма деятельности

Какие функции входят в сервис и кто ими пользуется?

Руководитель, заказчик, команда

Граница системы и цели участников

Диаграмма вариантов использования

Какие понятия и связи нужны для назначения транспорта?

Разработчики

Предметная область или структура данных

Диаграмма классов

Какие части приложения взаимодействуют с внешними сервисами?

Архитектор, технический руководитель

Архитектурная структура

Диаграмма компонентов

Диаграмма деятельности: когда важны шаги и развилки

Для Ирины команда выбирает диаграмму деятельности. Она покажет порядок действий, условия переходов и, при необходимости, зоны ответственности — например, клиники,

системы и диспетчера. Ветвление можно обозначить условием: данных достаточно или нет; подходящая машина найдена или нет. Если машина не находится автоматически, процесс может перейти к ручному поиску перевозчика и ожиданию подтверждения.

Важен именно ход работы. Диаграмма помогает спросить: где заявка может остановиться? Какие разные причины приводят к одному статусу? Кто должен выполнить следующее действие?

Она не должна превращаться в копию инструкции на каждую кнопку. Если диспетчер может исполнить условие процесса без знания расположения кнопок в интерфейсе, скорее всего, эти подробности не нужны на уровне процесса. А если нужно обсудить последовательность сообщений между конкретными объектами программы, может оказаться уместнее другой тип диаграммы — например, диаграмма последовательности.

Диаграмма вариантов использования: когда важны цели и граница

Руководитель спрашивает, что команда вообще берёт в работу. Для этого подходит диаграмма вариантов использования: акторы располагаются за границей системы, а внутри показываются цели, которые система поддерживает. Клиника может зарегистрировать заявку, диспетчер — назначить перевозчика или просмотреть причину задержки; внешний перевозчик может подтвердить приём доставки.

Такая диаграмма помогает согласовать функциональный охват. Но она не расскажет, в каком порядке выполняются шаги внутри назначения и почему заявка ждёт. Вариант использования — не подробный сценарий процесса, даже если его название звучит как действие. Для последовательности шагов потребуется описание сценария или подходящая поведенческая диаграмма.

Диаграмма классов: когда важны понятия и связи

Разработчику нужна модель данных, но прежде всего команда обсуждает предметную область. В черновой диаграмме могут появиться классы «Заявка», «Клиника», «Груз», «ТребованиеТемпературы», «Транспорт» и «Назначение». Связи показывают, например, что заявка относится к клинике, груз имеет температурное требование, а назначение связывает груз с транспортом.

На этом этапе важно решить, что нужно для анализа причин ожидания. Одного общего статуса может быть недостаточно: может понадобиться различать «нет подходящего транспорта», «неполные данные» и «ожидается ответ перевозчика». Это предметное решение, а не просто вопрос о том, в какую таблицу базы данных положить поле.

Диаграмма классов хорошо показывает статическую структуру: понятия, атрибуты, ассоциации и кратности связей. Но она не объяснит, в каком порядке диспетчер проверяет заявку и что делает при отказе перевозчика. Не стоит также автоматически считать каждый класс будущей таблицей базы данных: модель предметной области и физическая схема хранения могут различаться.

Диаграмма компонентов: когда важны программные части и зависимости

Если руководитель уточнит: «Где интеграции могут сорвать запуск?», фокус изменится. Тогда команда может рассмотреть диаграмму компонентов. Она показывает крупные части программного решения, предоставляемые и требуемые интерфейсы, зависимости между компонентами и внешними системами — например, сервис уведомлений или платформу партнёрского перевозчика.

Диаграмма компонентов полезна для обсуждения архитектурных границ и технических рисков. Но диспетчеру она не покажет рабочий процесс, а бизнес-заказчику не всегда поможет понять, какие функции обещаны. Если вопрос — «входит ли отслеживание доставки в первую версию?», сначала согласуйте функциональную границу; архитектурный рисунок сам по себе на это не ответит.

Выбор не обязан быть единственным на весь проект. Но у каждой отдельной диаграммы должен оставаться свой главный вопрос.

Шаг 5. Сделайте два наброска — и отдайте их читателям

Команда готовит первый черновик для диспетчера. На диаграмме деятельности остаются регистрация заявки, проверка обязательных данных, поиск подходящего транспорта и две важные развилки. Если данных не хватает, клиника получает запрос на уточнение. Если машина не найдена, начинается ручной поиск или ожидание подтверждения перевозчика. Команда подписывает схему: «Текущий процесс, черновик для проверки». Так никто не примет её за утверждённый регламент.

Затем разработчики делают отдельный набросок диаграммы классов. В нём они обсуждают, следует ли хранить причину ожидания отдельно от общего состояния заявки, как связать назначение с конкретным транспортом и нужно ли сохранять историю попыток назначения. На диаграмме процесса это обсуждение утонуло бы среди шагов и развилки; в модели структуры оно становится центральным.

После этого черновики показывают будущим читателям по одному, без длинной вводной лекции. Вместо вопроса «Ну как, понятно?» команда задаёт три более полезных:

1. Какой вопрос, по-вашему, решает эта диаграмма?
2. Покажите, что произойдёт в случае, который для вас важен.
3. Что, как вы считаете, осталось за границей модели?

Диспетчер замечает, что в черновике не отражено ожидание звонка от перевозчика. Разработчик спрашивает, можно ли отличить повторную попытку назначения от первоначальной. Эти реакции не означают, что диаграмма провалилась. Напротив, она выполнила важную работу: сделала расхождения видимыми.

Для быстрой проверки полезно наблюдать, где читатель остановился, какое обозначение понял неверно и какие детали начал искать. Не подсказывайте сразу. Если модель читается только при участии автора в роли экскурсовода, её ещё рано считать ясной. Исправлять лучше сначала смысл и уровень детализации, а уже потом — расположение фигур и аккуратность линий.

Практикум: три вопроса до первого значка

Запишите три вопроса о своей системе. Для каждого определите читателя, масштаб и диаграмму. Затем добавьте, почему ближайшая альтернатива хуже отвечает именно на этот вопрос.

Вопрос

Вероятный выбор

Почему не ближайшая альтернатива

Что делает сотрудник, когда заявка не проходит проверку?

Диаграмма деятельности

Диаграмма классов покажет данные заявки, но не порядок действий и ветвление

Какие цели клиника и диспетчер могут достигнуть с помощью сервиса?

Диаграмма вариантов использования

Диаграмма деятельности слишком подробно сосредоточится на ходе одного процесса

Какие сведения и связи нужны для выбора машины с нужным температурным режимом?

Диаграмма классов

Диаграмма вариантов использования покажет цель «назначить транспорт», но не структуру понятий

Это не таблица универсальных ответов. Если вопрос о взаимодействии программных частей, диаграмма компонентов может оказаться уместнее диаграммы классов. Если нужно проследить обмен сообщениями во времени, понадобится другой поведенческий ракурс. Выбирайте не по привычке, а по тому, какие отношения или события читатель должен увидеть.

Короткий алгоритм перед рисованием

- 1. Сформулируйте вопрос.** Что читатель должен понять или решить?
- 2. Назовите аудиторию.** Кто будет читать модель и что ему уже известно?
- 3. Выберите масштаб.** Весь контекст, один процесс, предметная структура или архитектура?
- 4. Обозначьте границы.** Что включено, что исключено и какие допущения пока не проверены?
- 5. Подберите диаграмму.** Нужны цели акторов, последовательность действий, структура понятий или программные зависимости?
- 6. Проверьте черновик на читателях.** Попросите объяснить его смысл, пройти важный случай и назвать то, что осталось за рамками.

На встрече по «Тёплому маршруту» команда не стала рисовать все виды UML сразу. Диспетчерам предложили модель процесса, разработчикам — модель предметной структуры, а для обсуждения границ продукта оставили отдельный ракурс. И главное: найденное на схеме подозрение о причинах задержек команда решила проверить по данным, а не объявлять доказанным только потому, что так нарисовано.

Перед первым значком стоит сделать паузу. Точный вопрос, понятный читатель, выбранный масштаб и честно обозначенные границы превращают диаграмму из украшения документа в рабочий инструмент. Сначала решите, кому и в чём должна помочь модель, — и только потом беритесь за карандаш.

Сценарии использования: акторы, границы системы и функциональные требования

На совещании в службе доставки лекарств спор начался с простого вопроса: «Кто нажимает кнопку?»

— Актор — «Тёплый маршрут», — сказал заказчик. — Сервис же обеспечивает доставку.

— Нет, актор — житель. Он выбирает лекарство и оставляет заявку, — возразил диспетчер.

— Я уже добавил базу данных, — сообщил разработчик. — Без неё система ничего не запомнит.

Каждый говорил о реальной части задачи, но все трое смешивали разные вещи: цель бизнеса, роль человека и внутреннее устройство программы. На доске росла диаграмма, а ясности не прибавлялось. В этом и состоит ловушка расплывчатых требований: слова знакомы всем, но каждый мысленно дорисовывает свою систему.

Чтобы начать проектировать сценарии использования, нужно не спрашивать «какие значки поставить?», а выяснить три вещи: кто взаимодействует с системой, зачем он это делает и где проходит граница рассматриваемой системы.

Сначала договоримся, что именно изображаем

Фраза «обеспечить доставку лекарств» описывает большую цель организации. В ней могут участвовать житель, диспетчер, фармацевт, курьер, аптека, транспорт, склад и несколько программ. Но UML-диаграмма вариантов использования не обязана изображать всю эту вселенную. Она показывает, как внешние роли взаимодействуют с конкретной системой, чтобы достичь своих целей.

Команда уточнила границу: в первой версии моделируется программный сервис «Тёплый маршрут», который принимает заявки жителей и помогает диспетчеру назначать курьеров. Физическая доставка — работа людей и транспорта — в систему не входит. Курьер пока не пользуется приложением, поэтому в этой модели он не актор, хотя в реальном процессе, конечно, участвует. Если позднее появится приложение курьера, границу и диаграмму пересмотрят.

Внутри границы — само программное обеспечение: мобильный интерфейс жителя, рабочее место диспетчера, правила проверки заявок и хранилище данных. Внешними по отношению к системе остаются люди и другие программы, которые с ней взаимодействуют. Поэтому база данных, принадлежащая «Тёплому маршруту», не актор: это внутренняя часть реализации. А вот отдельная система аптечных остатков может быть актором, если она находится за границей и обменивается с сервисом данными.

Граница — не декоративный прямоугольник. Она отвечает на практический вопрос: за какое поведение отвечает проектируемая система, а какое только наблюдает или использует? Если этот ответ не дан, команда легко начинает приписывать программе физические действия («доставить лекарство») или, наоборот, превращать каждую внутреннюю деталь в участника взаимодействия.

Актор — это роль, а не табличка на двери

В UML актор — внешняя по отношению к системе роль, которая взаимодействует с ней. Это может быть человек, другая программа или устройство. Актор определяется не названием должности, а тем, какую роль он играет в конкретном взаимодействии.

Один и тот же человек может быть жителем в одном сценарии и диспетчером в другом. Важна не его личность, а цель, ради которой он обращается к системе. И наоборот: один актор

«Диспетчер» может обозначать целую группу сотрудников, если для системы они выполняют одинаковые действия.

У акторов часто различают две практические роли:

- **Первичный актор** начинает взаимодействие ради собственной цели. В сценарии оформления заявки это житель: он хочет зарегистрировать запрос на доставку.

- **Вспомогательный актор** помогает системе выполнить сценарий или предоставляет ей услугу. Например, внешняя система аптечных остатков сообщает, есть ли нужный препарат.

Иногда первичным актором в другом варианте использования становится диспетчер: его цель — назначить курьера на заявку. Один и тот же участник может быть первичным в одном сценарии и вспомогательным в другом — классификация зависит от конкретного взаимодействия.

Фраза заказчика «актор — сервис» тоже требует уточнения. Если под сервисом он имеет в виду программный продукт, который мы сейчас проектируем, то сервис находится внутри границы и актором быть не может. Но другая, уже существующая платформа доставки, с которой «Тёплый маршрут» обменивается заявками, вполне может выступать внешним актором. Актор — не то, что звучит технически; это то, что находится за установленной границей и взаимодействует с системой.

От кнопки к цели

Житель нажимает кнопку «Новая заявка». Но нажатие — не цель. Это лишь действие на экране, один из способов начать взаимодействие.

Полезный вариант использования обычно называют глаголом и объектом: «**Оформить заявку на доставку**», «**Отменить заявку**», «**Назначить курьера на заявку**». Название должно подсказывать, какой наблюдаемый результат получит актор.

Сравните:

Неудачное название

Что в нём не так

Более полезное название

«Обработать данные»

Неясно, кто действует и ради чего

«Оформить заявку на доставку»

«Работать с системой»

Описывает сам факт использования, а не цель

«Назначить курьера на заявку»

«Экран заявки»

Называет интерфейс, а не результат

«Просмотреть статус заявки»

«Проверить адрес»

Может быть внутренним шагом системы

«Указать адрес доставки» — если это цель пользователя

Вариант использования не обязан начинаться с клика. Житель может позвонить в службу, а оператор внесёт заявку от его имени; цель всё равно может оставаться «Оформить заявку», а актором в этом взаимодействии будет оператор. Если же система проектируется так, что чело-

век регистрирует запрос самостоятельно, актором будет житель. Разные каналы взаимодействия могут потребовать разных сценариев — это зависит от того, что именно мы моделируем.

Есть и полезная мера масштаба. Вариант использования должен описывать цель, достаточно значимую для актора, но не превращать диаграмму в роман. «Заполнить поле с улицей» — слишком мелкий шаг. «Получить лекарство» может оказаться слишком большим результатом, если программная система умеет лишь зарегистрировать заявку и передать её диспетчеру. В таком случае точнее назвать цель «Оформить заявку на доставку».

Диаграмма «Тёплого маршрута»

Зафиксируем границу: **программный сервис «Тёплый маршрут» для регистрации и диспетчеризации заявок**. В таблице ниже показаны связи, которые должны быть видны на диаграмме вариантов использования: акторы находятся вне границы, варианты использования — внутри.

Актор вне границы

Вариант использования внутри границы

Роль в сценарии

Житель

Оформить заявку на доставку

Первичный актор; начинает взаимодействие ради регистрации заявки

Житель

Отменить заявку

Первичный актор; отзывает ранее созданную заявку, если правила это позволяют

Диспетчер

Назначить курьера на заявку

Первичный актор; выбирает исполнителя и фиксирует назначение

Внешняя система аптечных остатков

Оформить заявку на доставку

Вспомогательный актор; передаёт сведения о наличии препарата

На обычной UML-диаграмме эти варианты использования были бы овалами внутри прямоугольника с названием системы, а акторы — снаружи; связи показывали бы, кто участвует в каждом сценарии. Таблица делает те же отношения явными, не притворяясь, что знает порядок шагов. Именно этого диаграмма вариантов использования и не показывает: она не отвечает на вопрос, что происходит первым, вторым и третьим.

Граница помогает исключить ложных акторов. База данных внутри программного сервиса не появляется снаружи прямоугольника. Рабочее место диспетчера — тоже часть решения, а не актор: актор здесь сам диспетчер. Экран «Оформление заявки» — элемент интерфейса, но не цель пользователя. Напротив, внешний сервис аптечных остатков может быть актором, потому что располагается за границей и участвует в обмене данными.

Диаграмма даёт обзор, сценарий раскрывает детали

Диаграмма помогает быстро увидеть состав ролей и их цели. Но по ней нельзя надёжно реализовать поведение: она не говорит, какие данные вводит житель, что происходит при недоступном препарате и когда именно можно отменить заявку. Для этого нужен текстовый сценарий — последовательное описание взаимодействия.

Такое разделение полезно и для мышления. В когнитивной психологии хорошо изучены ограничения рабочей памяти: человеку трудно одновременно удерживать много несвязанных деталей. Диаграмма выносит наружу общую структуру, а нумерованный сценарий — последовательность и условия. Если попытаться вместить всё в одну схему, она перегружается. Если оставить только схему, критические ветви остаются в головах участников — и легко теряются.

Для нашего примера карточка сценария выглядит так.

Карточка UC-01: Оформить заявку на доставку

Цель: житель регистрирует заявку на доставку выбранного лекарства.

Первичный актер: житель.

Вспомогательный актер: внешняя система аптечных остатков.

Триггер: житель начинает оформление новой заявки.

Предусловие: житель вошёл в систему. Заявка ещё не создана.

Основной поток:

1. Житель начинает оформление заявки.
2. Система предлагает указать препарат, количество, адрес и предпочтительный интервал доставки.
3. Житель вводит нужные сведения и подтверждает адрес.
4. Система определяет, находится ли адрес в зоне обслуживания.
5. Система запрашивает у внешней системы аптечных остатков сведения о наличии препарата.
6. Система показывает жителю результат проверки и доступные условия доставки.
7. Житель подтверждает оформление.
8. Система сохраняет заявку со статусом «Новая», присваивает ей номер и показывает подтверждение.

Альтернативный поток A1: адрес вне зоны обслуживания. После шага 4 система сообщает, что доставить по указанному адресу нельзя. Житель может изменить адрес; тогда сценарий возвращается к проверке зоны обслуживания. Если он прекращает оформление, заявка не создаётся.

Альтернативный поток A2: препарата нет в наличии. После шага 5 система сообщает об отсутствии препарата. Житель может выбрать доступную замену, если такая возможность предусмотрена, или завершить оформление без отправки заявки. Правило о заменах нужно согласовать с заказчиком: его нельзя незаметно додумывать за пользователя.

Альтернативный поток A3: житель передумал до отправки. До шага 7 житель прекращает оформление. Система удаляет черновик или сохраняет его по установленному правилу; новая заявка не создаётся. Команда должна выбрать одно поведение, а не оставлять его на усмотрение разработчика.

Результат: либо создана заявка со статусом «Новая», либо заявка не создана. Если сценарий завершился отказом, житель получает понятное объяснение причины.

Это уже не просто рассказ о «доставке». В сценарии появились условия, точки проверки и ожидаемые результаты. Теперь заказчик может ответить на конкретные вопросы: должна ли система разрешать замену препарата? Сохраняется ли черновик? Каковы правила зоны обслуживания? Чем точнее такие решения зафиксированы, тем меньше места остаётся для несовместимых трактовок.

Отмена уже созданной заявки — отдельная цель и поэтому отдельный вариант использования. Для неё тоже нужны правила: может ли житель отменить заявку после назначения курьера? Нужно ли указывать причину? Как меняется статус? Например, предусловием может быть статус «Новая», а результатом — статус «Отменена». Если разрешена отмена и после назначения, это следует описать отдельно и согласовать с операционными правилами.

Связь вариантов использования с функциональными требованиями

Сценарий помогает обнаружить требования, но сам по себе не заменяет их. Для разработки и проверки полезно сформулировать требования так, чтобы их можно было подтвердить тестом или наблюдением.

Идентификатор
Функциональное требование
Проверка

FR-01

Система должна позволять вошедшему жителю указать препарат, количество и адрес доставки

Житель вводит данные, система принимает их

FR-02

Если адрес находится вне зоны обслуживания, система должна сообщить об этом и не создавать заявку

Для адреса за пределами зоны заявка не появляется в очереди

FR-03

Если препарат отсутствует, система должна показать результат проверки и не отправлять заявку без подтверждения жителя

При отсутствии препарата создаётся либо не создаётся заявка согласно выбранному правилу

FR-04

После подтверждения система должна присвоить заявке номер и установить статус «Новая»

Житель получает номер, диспетчер видит заявку в очереди

FR-05

Житель должен иметь возможность отменить заявку, если её статус допускает отмену

Статус меняется на «Отменена» при выполнении условий

FR-06

Диспетчер должен иметь возможность назначить курьера на заявку

После назначения в системе сохранены заявка и выбранный курьер

Такая связка позволяет проследить путь от цели до проверки: вариант использования показывает, чья это цель; шаги и альтернативы описывают поведение; функциональные требования фиксируют обязательные свойства системы. При этом сценарии не исчерпывают все требования. Например, безопасность персональных данных, время ответа или доступность сервиса тоже важны, но обычно относятся к другим категориям требований.

Когда include и extend добавляют путаницу

В UML есть отношения include и extend, и пользоваться ими стоит только тогда, когда их смысл действительно нужен.

include означает, что один вариант использования обязательно включает поведение другого. Это удобно для повторно используемого обязательного фрагмента. Например, если несколько сценариев всегда выполняют одну и ту же самостоятельную проверку полномочий,

её можно описать отдельно и включать в эти сценарии. Но `include` — не синоним слова «после этого».

`extend` обозначает дополнительное условное поведение, которое при определённых условиях расширяет базовый вариант использования. Для этого должны быть ясны условие и точка расширения. `extend` — не просто способ вынести любую альтернативную ветку из текста на диаграмму.

В «Тёплом маршруте» отсутствие лекарства — обычная ветвь сценария оформления заявки. Само по себе оно не становится вариантом использования `extend`: житель не приходит в систему с самостоятельной целью «расширить оформление случаем отсутствия». Если команда без ясной причины соединит сценарии `include` и `extend`, диаграмма станет похожа на карту метро: линий много, но куда ехать — непонятно.

Проверка перед публикацией модели

Перед тем как отдавать диаграмму разработчикам, заказчикам или тестировщикам, задайте несколько вопросов.

1. Названа ли система и ясна ли её граница? Понятно ли, что входит в программный сервис, а что находится снаружи?

2. Каждый ли актер внешняя роль? Не оказались ли внутри диаграммы база данных, экран, внутренний модуль или функция?

3. У каждого ли варианта использования есть актер и осмысленная цель? Можно ли сказать, какой результат получит роль?

4. Не назван ли вариант использования слишком мелко или слишком широко? «Нажать кнопку» — шаг, а «обеспечить доставку» может быть целью целой организации.

5. Описаны ли значимые альтернативы? Что будет при неверном адресе, отсутствии препарата или отмене?

6. Проверяемы ли связанные требования? Можно ли определить, выполнено ли требование, не угадывая намерения автора?

Небольшое упражнение: выберите любой знакомый сервис и выпишите три роли, которые с ним взаимодействуют. Для каждой сформулируйте по одной цели в виде «глагол + объект». Затем проведите воображаемую границу вокруг именно того продукта, который моделируете, и перечеркните внутренние компоненты, ошибочно попавшие в список акторов. Если цель нельзя проверить по наблюдаемому результату, переформулируйте её.

В начале совещания участники спорили, кто нажимает кнопку, и пытались добавить в диаграмму всё, что помогает программе работать. К концу у них появилась граница системы, роли жителя и диспетчера, внешний поставщик сведений об остатках и сценарий с понятными исходами. Это не означает, что вопросов больше нет. Хорошая модель не устраняет необходимость договариваться — она делает спор точным и проверяемым.

Вариант использования описывает цель внешней роли во взаимодействии с системой. Когда граница проведена ясно, пользователи, функции и внутренние компоненты перестают притворяться друг другом, а общая фраза превращается в сценарий, который можно реализовать и проверить.

Поэтому при описании сценария важно держаться цели пользователя, а не пересказывать устройство программы. Актер показывает, кто взаимодействует с системой; граница отделяет её ответственность от внешней; вариант использования формулирует, какого результата добивается внешняя роль. Вместе они помогают выявить функциональные требования и договориться, что именно система должна делать, не смешивая это с тем, как она будет устроена внутри.

Но одного перечня целей недостаточно, когда нужно понять ход взаимодействия: какие действия следуют друг за другом, где возможны разные условия, что происходит параллельно

и кто за что отвечает. Для этого сценарии раскрывают на следующем уровне — с помощью диаграмм деятельности.

Диаграммы деятельности: процессы, условия, параллельность и ответственность

В 9:06 Марина отправила заявку на доставку назначенного ей лекарства. Через двадцать минут диспетчер сказал, что заказ проверен. На складе тем временем ждали подтверждения, можно ли собирать покупку, а курьер не видел её в списке доставок. В 11:10 Марина позвонила сама. Никто не пытался переложить на коллег ответственность: каждый честно считал, что свою часть выполнил. Просто слово «проверить» означало для каждого разное.

Так часто и рождается задержка: не в большом сбое, а между двумя вполне разумными действиями. Диаграмма деятельности помогает увидеть этот зазор. Она показывает не только, что происходит с заявкой, но и кто выполняет работу, какие решения меняют маршрут, что можно делать одновременно и где процесс оставляет людей без ясного следующего шага.

Сначала восстановим историю

Команда не начинает с красивых символов. Она разговаривает с участниками процесса — отдельно, а затем вместе. Марина вспоминает, что указала адрес и пожелала получить заказ утром. Диспетчер говорит: «Я проверил заявку». На вопрос, что именно он проверял, уточняет: адрес, контактный телефон и удобное время. Сотрудник склада тоже говорит: «Мы проверили заказ», имея в виду наличие лекарства и возможность собрать остальные позиции. Курьер понимает под проверкой сверку уже собранного пакета перед выездом.

Это не спор о словах. Это три разных действия, которые команда поначалу сжала в одно. А пока шаги не названы точно, трудно понять, где заявка ждёт, кто должен принять решение и что вообще означает статус «проверено».

При реконструкции полезно разделить три слоя:

1. Наблюдаемые факты. Что участники действительно сделали в случае Марины и в каком порядке?

2. Правила. Что должно происходить согласно действующей политике?

3. Предложения. Что изменить, чтобы следующий заказ не повторил этот путь?

Если смешать эти слои, диаграмма незаметно превратится из описания процесса в изображение желаемого будущего. Это не всегда плохо, но команда должна понимать, какую именно модель строит: **as-is** — процесс как он есть, или **to-be** — процесс, каким его хотят сделать.

Для разговора об исполнителях мы выберем дорожки ответственности, или swimlanes. В нашем случае это четыре дорожки: **Житель, Диспетчер, Склад, Курьер**. Они группируют действия по тому, кто за них отвечает. Это не организационная схема: дорожка не утверждает, что у сотрудника есть определённая должность, что он подчиняется конкретному руководителю или что вся работа происходит в одной системе. Она отвечает на более узкий и полезный вопрос: **кто выполняет этот шаг?**

От заявки к действиям

Диаграмма деятельности читается как маршрут через работу. Начальный узел показывает, где начинается моделируемая деятельность; обычно его изображают закрашенным кружком. За ним располагаются действия — конкретные единицы работы, часто обозначаемые скруглёнными прямоугольниками. Потоки управления, то есть направленные линии между узлами, показывают, какой шаг может выполняться следующим.

Начальный узел — не то же самое, что действие «Оформить заявку». Первый задаёт старт процесса, второе описывает работу участника. В нашем случае стартом может быть поступление заявки, а первым действием в дорожке жителя — «Отправить заявку».

Когда заявка обработана, процесс завершается конечным узлом деятельности, который обычно изображают как кружок с заполненным центром внутри внешнего круга. Он означает завершение всей деятельности. Например, доставка вручена и зарегистрирована либо заявка отменена и об этом сообщено жителю.

Есть и другой элемент — конечный узел потока, обычно кружок с крестом. Он завершает только отдельную ветвь, а не всю деятельность. Это важно, когда работа идёт параллельно: окончание одной ветви не должно случайно выглядеть как отмена остальных.

Для первого разбора запишем путь заявки Марины без попытки уместить каждую мелочь:

Этап

Дорожка

Назначение

Заявка поступила

Начало процесса

Зафиксировать старт обработки

Отправить заявку

Житель

Передать сведения и пожелания по доставке

Проверить полноту данных

Диспетчер

Убедиться, что есть нужные контакты, адрес и другая обязательная информация

Уточнить недостающие сведения, если они нужны

Диспетчер и житель

Получить данные, без которых нельзя продолжить

Проверить наличие и подготовить заказ

Склад

Установить, что можно собрать и что отсутствует

Подготовить черновой маршрут

Диспетчер

Проверить адрес, временное окно и возможную последовательность доставок

Передать маршрут и заказ на доставку

Диспетчер и курьер

Зафиксировать передачу работы исполнителю

Доставить заказ и отметить результат

Курьер

Завершить доставку или зарегистрировать её неудачу

Таблица пока не раскрывает все развилки. Но уже видны два важных вопроса: что происходит, если данных не хватает, и как диспетчер узнаёт, что на складе обнаружено отсутствие

лекарства? Если ответ звучит как «как-нибудь сообщают», перед нами не готовое правило, а пробел, который нужно исследовать.

«Проверить» — не решение

Действие выполняет работу: например, **«Проверить полноту заявки»**. Решение определяет, какой путь будет выбран после работы. Само решение на диаграмме обычно изображают ромбом. Условия — например, [данные полны] и [данных не хватает] — подписывают на исходящих из него потоках.

Это различие практично. Если написать в ромбе «Данные полны?», мы видим вопрос, на который процесс должен ответить. Если поместить туда «Проверить полноту», действие и решение смешаются: непонятно, где именно заканчивается проверка и по какому правилу выбирается дальнейший путь.

Для заявки Марины после проверки полноты возможны два исхода:

- [данные полны] — обработка продолжается;
- [не хватает данных] — диспетчер запрашивает уточнение у жителя.

После уточнения заявка возвращается на проверку. Получается цикл, а не тупиковая ветвь. На практике здесь стоит уточнить и ещё одну деталь: кто следит за ответом и что случится, если его долго нет? Диаграмма не обязана решать этот вопрос сама, но делает его заметным.

Условия на выходах из ромба должны быть сформулированы так, чтобы их можно было проверить. Хорошо, если они покрывают все ожидаемые случаи и не противоречат друг другу. Если возможны «есть в наличии», «нет в наличии» и «данные о наличии пока不可靠», не стоит прятать третий вариант за расплывчатым словом «иначе». Неопределённый поток — это не упрощение, а будущий спор.

После альтернативных путей часто нужен ромб слияния. **Слияние** сводит обратно несколько альтернатив, например путь с полными данными и путь с уточнёнными данными. Оно не ждёт, пока «закончатся обе ветви»: в конкретном проходе заявки обычно была выбрана только одна.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.