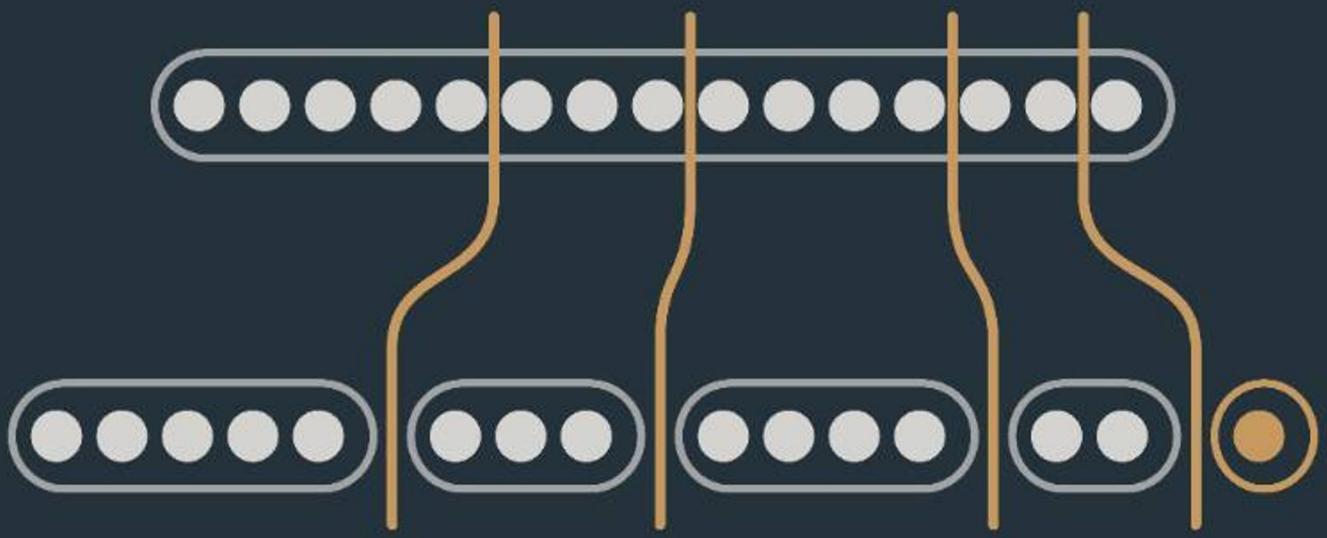
ИГОРЬ ЧЕПЕЛЕВ

Как разрезать команду

Разработка в эпоху ИИ:
известные подходы и новые рамки



Игорь Чепелев

Как разрезать команду. Структура разработки в эпоху ИИ

«Автор»
2026

Чепелев И.

Как разрезать команду. Структура разработки в эпоху ИИ / И. Чепелев — «Автор», 2026

Когда код пишут ИИ-агенты, продуктов становится больше, чем людей, которые могут за них отвечать. Дать каждому продукту свою команду не получится: людей не хватит.Книга для руководителя разработки или продукта в компании на 10–50 инженеров. Она показывает, как делить разработку на команды, пары и владельцев и где резать нельзя.Внутри — пять решений, из которых складывается любое разделение. Законы, которые не обойти: связи, очереди, отпуска и дежурства, с простым счётом. Двенадцать известных подходов, от Scrum до Team Topologies. Пять новых рамок автора. Разбор на компании из пятнадцати инженеров: шесть планов с числами и критикой. Отдельно посчитано, что меняется, если ревью кода отдать боту, а тесты и выпуск — автоматике.В конце — три разговора с людьми, проверка через цикл, шпаргалка и бланки. Свои рамки автор называет гипотезами и про каждую говорит, где она не работает.

© Чепелев И., 2026
© Автор, 2026

Содержание

1. Зачем эта книжка 5
2. Пять решений, из которых состоит любое разделение 7
3. Законы, которые не отменить 8
    3.1. Закон Конвея 9
    3.2. Арифметика связей 11
    3.3. Предел головы 15
    3.4. Арифметика отсутствия 18
    3.5. Закон узкого места 22
    3.6. Проверочный список 25
4. Известные подходы 26
    Функциональные команды 27
    Развитие и эксплуатация (Change / Run) 28
    Scrum, LeSS и SAFe 29
    Модель Spotify 30
    Team Topologies 31
    Amazon, две пиццы и единоличный владелец 32
    Продуктовые команды с полномочиями 33
    Shape Up 34
    Пионеры, поселенцы, градостроители 35
Конец ознакомительного фрагмента. 36

Игорь Чепелев
Как разрезать команду. Структура разработки в эпоху ИИ

1. Зачем эта книжка

Разделить команду — значит принять пять разных решений, а почти каждый известный фреймворк отвечает только на одно-два из них. Эта книжка собирает ответ целиком: что нельзя обойти, что уже придумано, что изменил ИИ и в каком порядке действовать.

Она написана для руководителя разработки или продукта в компании, где от десяти до пятидесяти инженеров. Одна-две команды выросли до семи-восьми человек, проектов хочется больше, а код уже в основном пишут ИИ-агенты.

Главная мысль. Режьте ответственность только там, где уже есть проверка, подмена и право решать. В остальных местах назначайте владельцев и стройте недостающее. Вот как это выглядит на первой из пяти рамок. Команда — это единица, у которой «решать», «делать» и «везти» переживают отсутствие одного человека. Вычеркните каждого по очереди на три недели и посмотрите, кто несёт каждую ношу. Если клетка осталась пустой, ноша держится на одном человеке. Ей нужен дублёр, общее дежурство или точное слово: «владелец» вместо «команды». Агенты этого правила не отменяют. Проект теперь ведут один-два человека. Подменять их, проверять в опасной зоне и дежурить по-прежнему должна команда.

О времени в примерах. Код в примерах пишут агенты, поэтому сама работа занимает часы. Дни уходят на ожидание: очередь, проверку, согласование, выпуск. Там, где для сравнения показано, как было до агентов, это сказано прямо. Числа условные, подставляйте свои.

Что здесь чужое и что новое. Главы 3–6 опираются на опубликованные работы, ссылки собраны в конце. Пять рамок из главы 7 — мои рабочие гипотезы. Их никто не проверял на выборке компаний, поэтому пробуйте их на одном пилоте, прежде чем перестраивать всё. Счёт цены разреза в главе 3 — моя упрощённая модель, вес стыка в ней выбран по одному исследованию. Аналогии из приложений А и Б подсказывают, куда смотреть; доказательств в них нет. Раздел «Продуктов больше, чем хозяев» в главе 6 и три стопки в главе 8 тоже добавил я: это мои наблюдения, исследований за ними нет. Шесть планов в главе 9 посчитаны на условном примере. За самым смелым из них стоят одно описание подхода без цифр и один доклад, поэтому считайте его гипотезой. Главы 10 и 11 собраны из рамок и счёта самой книжки. Новых исследований за ними нет.

Как устроена. Глава 2 раскладывает разделение на пять решений. Глава 3 напоминает законы, которые действуют при любой схеме: каждому отведена подглава с примером на пальцах, счётом и рисунком. Главы 4 и 5 пересказывают известное: двенадцать способов провести границы и пять способов договориться, кто что решает. Глава 6 показывает, что сдвинул ИИ. Глава 7 предлагает пять новых рамок, глава 8 — порядок действий, глава 9 — разбор на примере. В главе 9 рядом стоят шесть планов для одной компании, от спокойного до самого смелого, с числами и критикой каждого. Глава 10 разбирает три разговора с людьми. Глава 11 показывает, как через цикл понять, удалось ли разделение, глава 12 — шпаргалка. В конце четыре приложения: три для тех, кто хочет копнуть глубже, и одно с бланками. Приложение А — математика разделения: графы, разбиения, очереди и законы масштабирования. Приложение Б — что о разделении знают социология, психология, экономика, биология и другие науки. Приложение В — исследования и фреймворки со ссылками. Приложение Г — пустые

формы для таблиц из глав 3–11. Слова, которые в книжке значат что-то своё, собраны в словаре после главы 12.

Если есть только полчаса, прочитайте главы 2, 7 и 8, а потом шпаргалку.

2. Пять решений, из которых состоит любое разделение

Разделение — это пять решений: кого считать командой, по чему резать, как команды стыкуются, кто что решает и как часто схема пересматривается.

Обычно всё начинается с доски. На ней рисуют прямоугольники с названиями и стрелки между ними, и через час схема выглядит готовой. Через месяц выясняется, что прямоугольники были самой лёгкой частью. Никто не договорился, кто чинит ночью, кто утверждает тариф и что делать, когда двум командам нужен один дизайнер.

Единица. Кого вы называете командой. Сколько в ней людей, какие роли обязательны, может ли командой быть один человек с агентами. От ответа зависит, переживёт ли она отпуск, ревью и ночной инцидент.

Граница. По чему проходит разрез. Вариантов больше, чем кажется: часть продукта, сегмент клиента, этап пути пользователя, стадия зрелости, тип работы, технология, метрика. Каждый вариант что-то склеивает и что-то рвёт.

Стык. Как команды работают друг с другом и кому принадлежит общее. Услуга с понятным входом, временная совместная работа, обучение и очередь заявок — четыре разных стыка. Сюда же относится всё, что видит каждый пользователь: навигация, оплата, вход, дизайн-система.

Права на решения. Что команда решает сама, о чём обязана сообщить, где нужен совет и где согласование. И кто судья, когда вопрос упал между командами.

Ритм. Как часто вы пересматриваете приоритеты, границы и сам состав. Схема, которую нельзя менять, устаревает за два квартала.

Пример. Команду из восьми человек делят на две. Единица: две команды по четыре. Граница: одна ведёт ассистента, другая — оплату и тарифы. Стык: у них общий вход в продукт и один дизайнер на двоих. Права: каждая сама выпускает изменения, а цену тарифа утверждает руководитель. Ритм: схему пересматривают раз в квартал. Пять строк, и ни одну нельзя нарисовать прямоугольником на доске, кроме первых двух.

Единица. *Вопрос:* Кто считается командой? *Что бывает, если его пропустить:* Владелец ушёл в отпуск, и направление встало.

Граница. *Вопрос:* По чему режем? *Что бывает, если его пропустить:* Одна задача проходит через три очереди.

Стык. *Вопрос:* Как команды работают друг с другом? *Что бывает, если его пропустить:* Общий дизайнер превращается в узкое место.

Права. *Вопрос:* Кто что решает? *Что бывает, если его пропустить:* Ответственность раздали, согласования оставили.

Ритм. *Вопрос:* Когда пересматриваем? *Что бывает, если его пропустить:* Пилот идёт год, и никто не знает, удался ли он.

Известные подходы сильнее всего в границах и стыках. Единица, права и ритм чаще остаются непроговорёнными, и разделения ломаются именно на них. В сводной таблице главы 4 у каждого подхода отмечено, какие из пяти решений он закрывает.

3. Законы, которые не отменить

Пять законов действуют при любой схеме. Система копирует структуру общения. Связи растут быстрее людей. Команда отвечает за ограниченный объём. Люди уходят в отпуск. А ИИ переносит очередь туда, где проверяют и решают. Каждому закону ниже отведена своя подглава: сцена из жизни, счёт на пальцах, рисунок и что с этим делать.

3.1. Закон Конвея

Система получается такой, как общаются люди, которые её делают: где проходит граница между командами, там пройдёт и граница в продукте.

Возьмём одну кнопку: «Скачать счёт в PDF». Для неё нужны три вещи: сама кнопка на экране, метод в API и шаблон документа. В компании три команды: фронтенд, бэкенд и документы. Работы у каждой на пару часов: код пишет агент, человек ставит задачу и проверяет результат. Но у каждой команды свой план, и задача встаёт в три очереди. Пусть в каждой она ждёт четыре дня; числа условные. Выходит шесть часов работы и двенадцать дней ожидания. Почти тринадцать дней на кнопку.

Теперь та же кнопка у команды оплаты, в которой есть все три умения. Очередь одна, четыре дня. Потом те же шесть часов работы. Меньше пяти дней вместо почти тринадцати, и никто не работал быстрее.

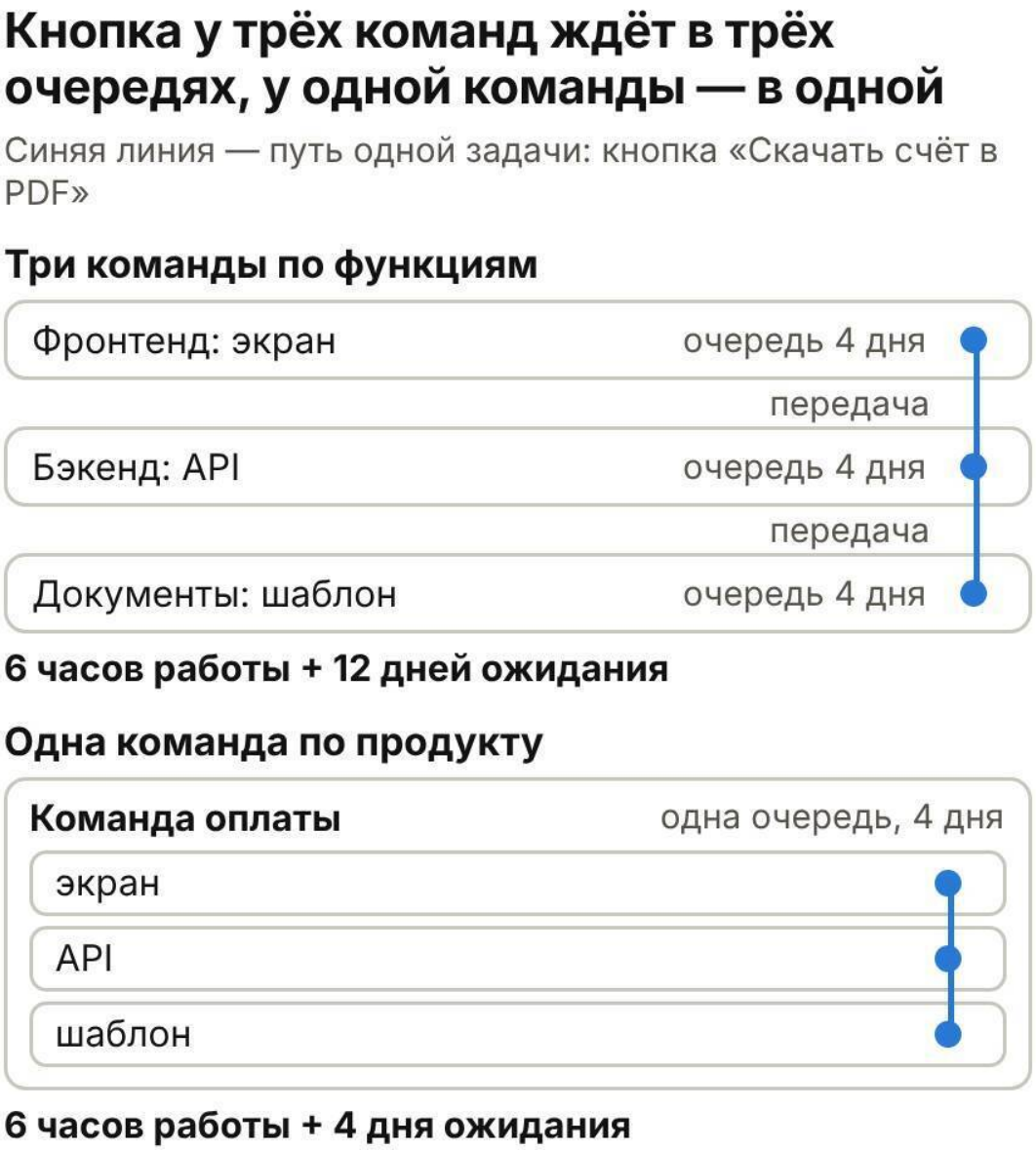


схема · одна задача у трёх функциональных команд и у одной продуктовой, числа условные

В первом случае работа заняла шесть часов из почти тринадцати дней, остальное съело ожидание на границах. До агентов та же кнопка стоила бы три дня работы, и ожидание занимало четыре пятых срока. Агенты сжали работу до часов, а очереди остались прежними. Теперь срок почти целиком состоит из ожидания, и граница между командами обходится дороже, чем раньше. И это только начало. Через год таких задач в продукте сложатся три отдельных мира: у экранов свои соглашения, у API свои, у документов свои, а между ними формальные передачи. Продукт стал похож на оргсхему.

Что сказал Конвей. В 1968 году программист Мелвин Конвей заметил, что устройство системы повторяет то, как общаются её создатели. Причина бытовая. Договориться внутри команды — это повернуться к соседу. Договориться между командами — это встреча, заявка и очередь. Поэтому всё, что требует частых договорённостей, оседает внутри одной команды, а на границе вырастает интерфейс: API, формат файла, регламент. Три команды напишут три модуля, даже если продукту нужен один.

Что показали исследования. В Microsoft проверили это на Windows Vista: показатели оргструктуры предсказывали сбойные модули точнее, чем объём изменений, сложность кода, покрытие тестами и зависимости (Nagappan, Murphy, Basili, 2008). Обзор 142 исследований подтвердил, что в компаниях устройство продукта обычно повторяет устройство организации, хотя и не всегда; открытые проекты из правила выбиваются (Colfer, Baldwin, 2016). Это сильная тяга, и законом природы её не назовёшь. Но ставить против неё дорого.

Как проверить у себя. Возьмите двадцать последних законченных задач. У каждой отметьте, сколько команд к ней прикасалось. Пример: одиннадцать задач сделала одна команда, шесть задач делали две, три задачи — три. Девять из двадцати, то есть 45%, пересекли границу. Если таких задач больше трети, граница стоит не там: команды режут то, что продукт хочет делать целиком. Треть — мой ориентир, общепринятой нормы тут нет.

Вторая проверка занимает пять минут. Положите рядом оргсхему и карту модулей. Если они совпали, закон у вас уже сработал. Остаётся вопрос, такую ли систему вы хотели.

Обратный манёвр. Раз система повторит оргсхему, начните с системы. Нарисуйте, из каких частей должен состоять продукт. Потом соберите команды под эти части. Хотите, чтобы оплату можно было менять отдельно от всего остального, — дайте оплате команду, которая владеет ею целиком: экран, API, данные. Часть продукта, у которой нет команды, не появится. Команда, у которой нет своей части, придумает её сама.

Три ловушки.

1. После реорганизации код ещё долго хранит старую форму. Команды уже новые, модули прежние, и каждая задача идёт через границу. Закладывайте на это время.

2. Временная проектная группа оставляет после себя модуль без хозяина. Назначьте хозяина до того, как группа разойдётся.

3. Восемь единоличных владельцев построят восемь маленьких систем, каждую со своими привычками. Общие соглашения придётся держать отдельным усилием.

С агентами так же. Репозиторий, данные и права, которые вы выдали агенту, тоже станут границей в коде. Агент с доступом только к экранам, скорее всего, решит задачу на экране, даже если правильное место для неё в API.

3.2. Арифметика связей

Пар в команде прибавляется быстрее, чем людей: людей вдвое больше — пар примерно вчетверо больше.

Представьте встречу, где каждый здоровается с каждым за руку. Втроём это три рукопожатия. Четвёртый входит и жмёт руку троим: стало шесть. Пятый добавляет четыре, шестой — пять. Каждый новый человек приносит столько связей, сколько людей уже в комнате. Восьмой добавит семь. Пятнадцатый — четырнадцать.

Рукопожатие длится секунду. Рабочая пара живёт весь год: эти двое должны знать, чем занят другой, договариваться о порядке работы и не ломать сделанное друг другом.

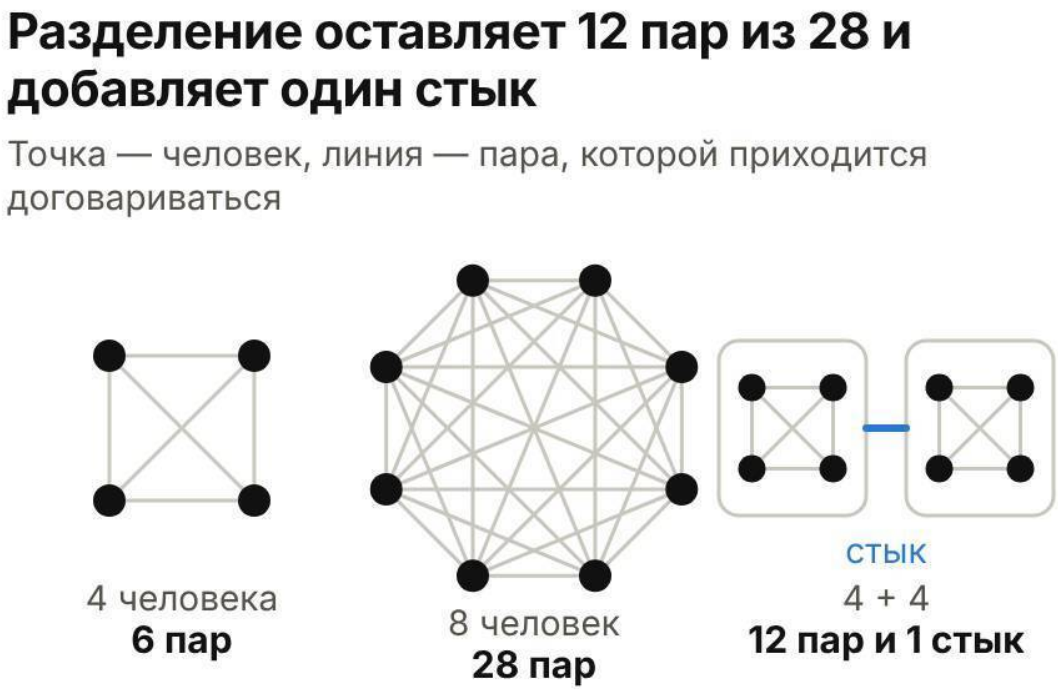
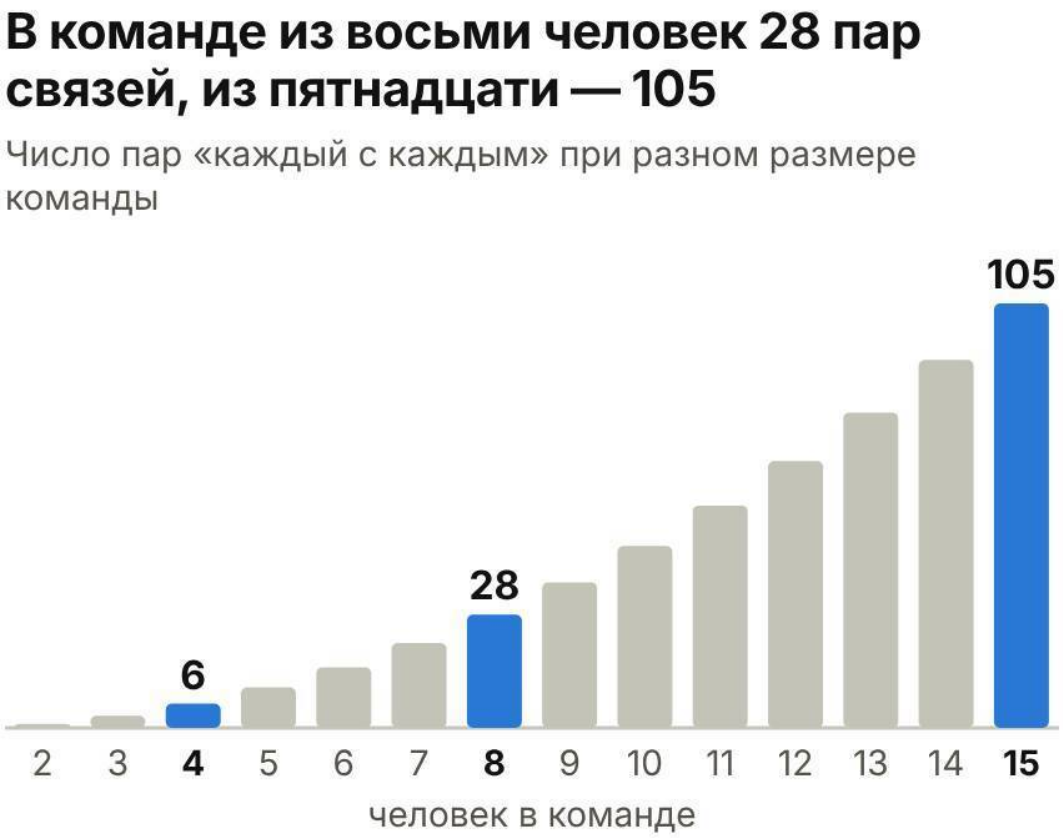


схема · 4 человека, 8 человек и две команды по 4

Рисунок показывает, что делает разделение. У восьми человек 28 линий. У двух команд по четыре внутри остаётся 12, а всё, что шло через границу, заменяет один стык. Ниже разберём это по шагам и посчитаем, сколько стоит сам стык.

Формула общая: в команде из n человек $n(n-1)/2$ пар, и каждая пара — это канал, по которому приходится договариваться.



расчёт по формуле n(n − 1)/2 · команды от 2 до 15 человек

Фред Брукс вывел из этой арифметики правило, известное по книге «Мифический человеко-месяц»: новые люди в опаздывающем проекте отодвигают срок ещё дальше.

Формула получается простым сложением. Первый человек ни с кем не связан. Второй добавляет одну связь, третий — две, девятый — восемь. Сумма 1 + 2 + … + (n − 1) и даёт n(n − 1)/2. Поэтому каждый новый человек обходится дороже предыдущего: девятый в команде добавляет восемь связей, шестнадцатый — пятнадцать.

В часах это выглядит так. Пусть каждой паре нужно полчаса в неделю, чтобы оставаться в курсе дел друг друга. У каждого n − 1 напарник. В команде из четырёх это полтора часа в неделю на человека, из восьми — три с половиной, из пятнадцати — семь, почти рабочий день. Полчаса здесь допущение. Подставьте своё число, пропорция останется.

Разделение лечит это наполовину. Разрежьте восьмерых на две команды по четыре. Внутри каждой шесть пар, всего двенадцать вместо двадцати восьми. Остальные шестнадцать пар теперь разделены границей. Их разговоры никуда не делись: они сжались в один стык между командами — общий канал, встречу лидов, договор о том, кто что кому отдаёт.

Стык дешевле шестнадцати пар, но дороже одной. Через него идёт всё, что раньше расходилось по шестнадцати каналам, и идёт медленнее: с очередью, с пересказом, с потерей подробностей. Джеймс Хербслеб и Аудрис Мокус измерили это на распределённой разработке. Задачи, в которых участвовало несколько площадок, занимали примерно в два с половиной раза больше времени, чем похожие задачи внутри одной (Herbsleb, Mockus, 2003). Соседние команды ближе, чем разные города, но порядок величины полезен. В расчётах ниже один стык считается за три пары.

Как это считать. Счёт идёт в три шага.

1. Пары внутри команд. Для каждой команды возьмите n(n − 1)/2 и сложите.

2. Стыки между командами. Нарисуйте команды кружками и соедините те, которые ждут друг от друга работу, ревью или решения. Число линий и есть число стыков. Потолок для k команд равен k(k − 1)/2: это случай, когда каждая зависит от каждой.

3. Цена. Сложите пары и стыки, умножив стыки на их вес. Вес три означает, что стык обходится как три пары.

Вот этот счёт для пятнадцати человек, если стыков столько, сколько возможно.



расчёт: пары внутри команд + 3 × стыки · 15 человек, от 1 до 8 команд

Одна команда из пятнадцати стоит 105: все пары внутри. Пятнадцать одиночек стоили бы втрое больше, 315: пар внутри нет, зато 105 стыков. Минимум приходится на три команды по пять и на четыре команды по четыре. Проверьте четвёртую строку руками. Четыре команды — это 6 + 6 + 6 + 3 = 21 пара внутри и 4 × 3 / 2 = 6 стыков, итого 21 + 6 × 3 = 39.

Вес меняет ответ. Если считать стык за одну пару, выгоднее всего пять троек. Если за пять пар — три пятёрки. Чем дороже вам обходится стык, тем меньше должно быть команд.

Тот же счёт для схемы «восемь владельцев-одиночек и две общие команды». Единиц десять. Внутри общих команд из четырёх и трёх человек девять пар. Каждый владелец зависит от обеих команд — это шестнадцать стыков, и ещё один между самими командами. Цена: 9 + 17 × 3 = 60. У трёх пятёрок 39.

Число стыков зависит от формы. Четыре продуктовые команды и платформа дают десять стыков, если каждая работает с каждой, и всего четыре, если продуктовые команды общаются только через платформу. Отсюда вся польза платформы и весь её риск: четыре очереди сходятся в одну точку.

Если команды в разных городах. Вес три вырос из измерения на распределённой разработке. Задачи между площадками шли там примерно в два с половиной раза дольше, и людей в них было занято больше (приложение В). Для команд в разных городах это измерение прямое, и вес три для них не завышен. Отсюда мой ориентир: границу команд ведите по границе площадок. Тогда стык между городами и стык между командами совпадают, и платите вы за один. У дежурства на двух площадках в разных часовых поясах свой порог. Google называет по шесть человек на каждой (глава 3.4).

Цена здесь условная. Она годится, чтобы сравнивать варианты между собой, и ничего не говорит о часах или деньгах. Число команд — такая же цена, как число людей. Вывод формулы, оптимум для других размеров и расчёт очередей собраны в приложении А.

3.3. Предел головы

Команда уверенно отвечает за ограниченный объём, и каждая область сверх него замедляет все остальные.

Утро понедельника в команде из пяти человек. Они ведут ассистента, оплату, вход, админку и бота в мессенджере. За выходные пришло три сбоя в трёх разных областях. Вера собиралась доделать поиск в ассистенте, но до обеда разбирается с ботом, потому что Денис в отгуле. После обеда её зовут в оплату: там нужен второй взгляд. К поиску она возвращается в четверг и первые два часа вспоминает, на чём остановилась.

Вера работает хорошо. Просто в её голове лежит больше, чем туда помещается.

Переключение стоит времени. Джеральд Вайнберг в начале 1990-х предложил прикидку: каждое дополнительное дело, которое человек ведёт одновременно, съедает около пятой части его времени. При двух проектах на каждый остаётся по 40%, а 20% уходит на то, чтобы вспомнить, где остановился. При трёх — по 20% на проект и 40% на переключение (пересказ на сайте организации Scrum).



таблица Вайнберга в пересказе · от 1 до 5 проектов одновременно

Измерением это не назовёшь: Вайнберг дал прикидку, и точные проценты у вас будут свои. Но направление проверяется за неделю. Посчитайте, сколько раз за день человек меняет область, и спросите, сколько у него уходит на то, чтобы войти обратно.

Память команды тоже ограничена. Джордж Миллер в 1956 году описал предел примерно в семь единиц, которые человек удерживает одновременно. Нельсон Коуэн в 2001 году уточнил его до четырёх. Команда — не один мозг, но и у неё есть предел того, что она помнит без записей. Авторы книги Team Topologies называют это когнитивной нагрузкой команды и дают ориентир: команде по силам одна сложная область или две-три простые.

Матрица знаний. Самый быстрый способ увидеть предел — нарисовать таблицу. По строкам люди, по столбцам области. В клетке — что человек может сделать с этой областью ночью, без подсказки: починить, разобраться за день или ничего.



условный пример · 5 человек, 5 областей

В нашей пятёрке у оплаты, входа и бота по одному человеку, который починит их без подготовки. Это три очереди к трём людям. Любое изменение в оплате ждёт Анну. Любой сбой в боте ждёт Дениса. А у Веры и Дениса по две области сразу, и переключаются они чаще всех.

Счёт за полчаса. Выпишите всё, за что отвечает команда: части продукта, сервисы, интеграции, типы пользователей, ночные сценарии. Для каждой строки спросите, сколько человек починят это без подготовки. Строки с ответом «один» или «никто» означают перегруз.

Потом оцените сложность: простая область, где всё понятно из кода, или сложная, где нужно помнить историю решений. В примере пять областей, три из них сложные. По ориентиру Team Topologies это работа для двух-трёх команд. Одна команда её не удержит.

Признаки, что предел пройден.

1. Ревью ждёт одного и того же человека.
2. На планировании половину задач может оценить только один.
3. Новичок через три месяца всё ещё боится выпускать изменения сам.
4. Про часть кода говорят «туда лучше не лезть».
5. Сбои в одной области всегда чинит один и тот же человек, и в отпуск он уходит с ноутбуком.

Что с этим делать. Способов три, и все они об одном: привести объём в соответствие с головами.

• Убрать область: закрыть, передать другой команде или купить готовое. Самый честный способ и самый редкий.

• Упростить область. Общая платформа, которая берёт на себя выкладку, мониторинг и доступы, снимает с команды часть того, что нужно помнить.

• Добавить знающих: парная работа над задачами в чужой области, дежурства по очереди, записанные сценарии сбоев.

Между крайностями есть рабочая середина. Один владелец на область — это минимум переключений и ноль запаса. «Все знают всё» — это запас, за который платят переключениями весь день. Середина такая: у каждой области двое-трое знающих, у каждого человека не больше двух областей. Для пятерых с пятью областями это десять тёмных клеток в матрице, по две в каждой строке и в каждом столбце.

Что меняет ИИ. ИИ пишет код быстрее, но отвечать за написанное по-прежнему приходится людям. Читать и проверять теперь нужно больше: поток изменений в коде вырос, цифры собраны в главе 3.5. Нагрузка на головы от этого только растёт. Авторы Team Topologies переносят то же правило на самого агента: его окно контекста работает как предел нагрузки, поэтому агенту нужна узкая и ясная зона ответственности (Team Topologies).

3.4. Арифметика отсутствия

Человека нет на работе пять-шесть недель в году, поэтому область с единственным владельцем примерно каждую десятую неделю остаётся без хозяина.

В понедельник Анна улетает в отпуск на две недели. Оплату в компании ведёт она одна. Во вторник платёжный провайдер меняет формат уведомлений, и часть платежей перестаёт подтверждаться. Борис открывает код оплаты впервые за год. К вечеру он находит место, где всё ломается, но выпускать исправление не решается: непонятно, что ещё на это завязано. Поддержка отвечает клиентам вручную. Анна в это время идёт по горной тропе без связи.

Никто здесь не ошибся. Анна имеет право на отпуск, Борис честно старался. Ошиблась схема: на бумаге «команда оплаты» есть, а в отпускную неделю её нет.

Счёт первый: один человек. Четыре недели отпуска и неделя-другая болезней дают пять-шесть недель, когда человека нет. Цифра оценочная, подставьте свою. В году 52 недели, и 5,5 из них — это 10,6% времени, каждая десятая неделя. Для «команды» из одного человека это недели, когда не выходит ни одно изменение, некому разобрать сбой и некому ответить соседям.

Счёт второй: двое. Дадим Анне дублёра — Бориса, который знает оплату так же, как она. Каждого из них нет примерно каждую десятую неделю. Шанс, что в случайную неделю нет обоих, — десятая от десятой, около одной сотой. За год это чуть больше половины недели: три рабочих дня вместо пяти с половиной недель.



условный пример · 52 недели, два человека по 5,5 недели отсутствия

На рисунке условный год. У Анны и Бориса по 5,5 недели отсутствия, а совпала половина одной недели на стыке февраля и марта, когда оба заболели.

Простой можно сократить ещё вдвое одним правилом: владелец и дублёр не уходят в отпуск одновременно. Тогда совпасть могут только болезни или болезнь одного с отпуском

другого, и пустыми остаются полтора рабочих дня в году. Правило ничего не стоит, нужен только общий календарь отпусков.

Расчёт считает отсутствия независимыми. В сезон гриппа люди болеют разом, поэтому настоящая цифра будет выше расчётной. Вывод остаётся прежним: второй знающий превращает недели простоя в дни.

Обратная сторона: большая команда редко в сборе. Та же арифметика бьёт и по большим командам. Четверо бывают в полном составе 33 недели из 52. Остальные 19 недель кого-то нет. Восемь человек собираются все вместе 21 неделю в году, меньше половины времени.



расчёт: $52 \times (1 - p)^n$ и $52 \times p^n$ при $p = 5{,}5/52$ · команды от 1 до 8 человек

Отсюда правило для планов: считайте каждого человека за 0,9. Команда из четырёх — это 3,6 человека. План, который сходится только при полном составе, развалится каждую третью неделю.

Счёт третий: дежурства. С ночными вызовами арифметика строже. В неделе 168 часов, рабочих из них 40. Остаётся 128 часов, когда сервис работает, а люди отдыхают. Если дежурит один человек, все 128 часов каждую неделю достаются ему. Вдвоём каждый дежурит 26 недель в году, а когда напарник уходит в отпуск на месяц, второй дежурит месяц подряд. Вчетвером выходит 13 недель в году, каждая четвёртая.



расчёт: 52 недели / число дежурных · команды от 1 до 8 человек

Ханна Фоксвелл называет четырёх разработчиков минимумом для устойчивого графика дежурств; командам меньшего размера, по её словам, остаётся отдавать поддержку отдельной группе (LeadDev, апрель 2026). У Google порог выше: восемь человек для команды на одной площадке или по шесть на каждой из двух площадок в разных часовых поясах (Google SRE, приложение B). Противоречия тут нет. В Google одновременно дежурят двое, основной и запасной, и никто не должен дежурить больше четверти времени (глава о дежурствах). Один дежурный при том же правиле — это четыре человека.

Счёт четвёртый: люди уходят насовсем. Отпуск заканчивается, увольнение нет. Пусть из компании за год уходит от 10 до 15% людей; подставьте свою цифру. Для одной области с одним владельцем это терпимый риск. Но если таких областей восемь, шанс, что за год уйдёт хотя бы один владелец, — от 57 до 73%.

Областей с единственным владельцем: 1. *Уходит 10% людей в год:* 10%. *Уходит 15% людей в год:* 15%.

Областей с единственным владельцем: 4. *Уходит 10% людей в год:* 34%. *Уходит 15% людей в год:* 48%.

Областей с единственным владельцем: 8. *Уходит 10% людей в год:* 57%. *Уходит 15% людей в год:* 73%.

Областей с единственным владельцем: 12. *Уходит 10% людей в год:* 72%. *Уходит 15% людей в год:* 86%.

Числа показывают вероятность, что за год уйдёт хотя бы один из владельцев. После ухода область стоит без хозяина, пока вы ищете человека и пока он входит в курс дела, а это обычно

месяцы. Схема с единственными владельцами выглядит экономной до первого заявления об уходе.

Что с этим делать.

1. У каждой области двое знающих: владелец и дублёр. Проверка простая: дублёр сам выпускал изменение в этой области за последние три месяца. Если нет, он дублёр только на бумаге.

2. Владелец и дублёр не уходят в отпуск одновременно. Календарь отпусков общий и виден всем.

3. Дежурят минимум вчетвером. Если в одной области четверых нет, соберите общую дежурную смену на несколько областей и напишите для неё инструкции по каждому ночному сценарию.

4. Если четверых не набрать, скажите об этом вслух: поддержка работает в рабочие часы, ночные сбои ждут утра. Честное «ночью не чиним» надёжнее графика, который держится на одном человеке.

5. В планах считайте каждого за 0,9 человека.

6. Агент помогает дублёру, но дублёром не становится. Он может найти место в коде и подготовить исправление. Решение выпускать и ночной звонок остаются за человеком.

Три вопроса к вашей схеме. Сколько у вас областей, где изменение может выпустить только один человек? Кто дежурит, когда дежурный в отпуске? Что будет с областью, если её владелец уйдёт через месяц?

Из этой арифметики получается нижняя граница для единицы разделения. Решать за область может один человек. Выпускать в ней изменения должны уметь минимум двое. Дежурить — минимум четверо.

3.5. Закон узкого места

Работа доходит до пользователя со скоростью самого медленного шага, и ускорять остальные шаги значит растить очередь перед ним.

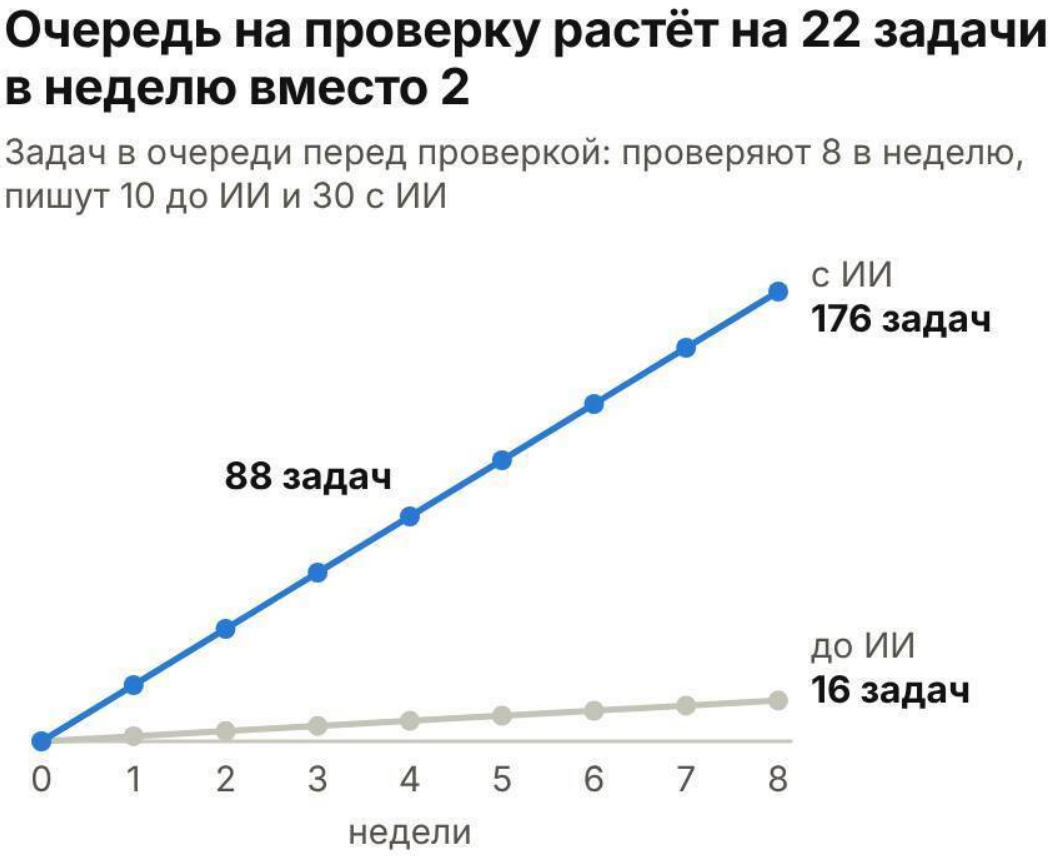
В ресторане новый повар. Он готовит втрое быстрее прежнего. Официант в зале по-прежнему один, и за час он выносит столько же тарелок, сколько раньше. Гости быстрее не едят. Зато на раздаче стынут блюда, а повар недоумевает, почему выручка не выросла.

В разработке всё так же. Возьмём условную команду, у которой три шага: написать, проверить, выложить. Пишут 10 задач в неделю, проверяют 8, выкладывать успевают 12. До пользователя доходит 8: столько пропускает проверка. Теперь команда получает ИИ-помощников и пишет 30 задач в неделю. Проверяют по-прежнему 8. До пользователя доходит 8.



схема · условная команда, три шага, задач в неделю до ИИ и с ИИ

Разница есть, но совсем в другом месте. Раньше перед проверкой копилось две лишние задачи в неделю. Теперь двадцать две. Через месяц в очереди 88 задач, и та, что встала последней, дождётся проверки через одиннадцать недель.



расчёт: (написано − проверено) × число недель · проверяют 8 в неделю, пишут 10 и 30

Очередь тоже стоит денег. Задача, которая ждёт два месяца, устаревает: код вокруг уже изменился, автор забыл детали, заказчик передумал. Чем длиннее очередь, тем больше работы уходит в переделку.

Что видно в данных. LeadDev пишет, что Meta (признана в России экстремистской организацией и запрещена) в 2026 году переводила подразделения с групп по 10–20 человек на группы из трёх—пяти с агентами. Изменений кода во внутренних платформах стало больше на 220% за год, а изменений, которые дошли до пользователей новой или улучшенной функцией, — только на 36%. Крупных инцидентов стало больше на 40%, времени на тушение пожаров — на 70% (LeadDev, сентябрь 2026). Все четыре числа восходят к расследованию Reuters от 26 августа 2026 года. Оно ссылается на внутренние записи компании. Комментировать эти данные Meta отказалась. Компания DX собрала данные более чем 500 организаций: за четыре квартала число запросов на слияние в неделю выросло на 37%, а доля времени на новые возможности почти не изменилась (там же). Отчёт DORA за 2025 год, почти 5 000 опрошенных, видит то же с другой стороны: с распространением ИИ растёт скорость поставки и падает её стабильность, а отдача зависит от качества внутренней платформы (Google Cloud об отчёте DORA).

Узкое место при этом не застревает на проверке, оно едет дальше. Эндрю Ын замечает, что тормозом становится продуктовая работа: собрать отзывы и решить, что строить. Рэйчел Лейкок из Thoughtworks предупреждает о том же про руководителей: если строить стало быстро, решений на них сваливается больше, чем они успевают принять (LeadDev, апрель 2026).

Как найти своё узкое место. Возьмите двадцать недавно законченных задач и для каждой выпишите, сколько дней она провела в каждом статусе. Пример: в работе полдня, ждала

проверки три дня, ждала выкладки один день. Из четырёх с половиной дней работа заняла полдня, то есть девятую часть. Статус, где задачи ждут дольше всего, и есть узкое место. В примере это проверка.

Есть и способ без таблицы. Посмотрите на доску задач: в какой колонке больше всего карточек. И послушайте, кому в команде чаще всего пишут «когда посмотришь?».

Что делать, когда нашли. Элияху Голдратт в книге «Цель» описал порядок из пяти шагов. В пересказе для разработки он выглядит так.

1. Найти узкое место. Одно, самое тесное.

2. Выжать из него всё. Проверяющие начинают день с проверок. Мелкие запросы не ждут в одной очереди с крупными.

3. Подчинить ему остальных. Авторы не начинают новую задачу, пока в очереди на проверку больше условленного числа. Запросы на слияние становятся меньше.

4. Расширить. Больше людей с правом проверять, автоматические проверки, тесты, которые ловят ошибку до человека.

5. Вернуться к первому шагу. Узкое место переехало, ищите его заново.

Третий шаг даётся труднее всего. Автор, который сидит без новой задачи, выглядит бездельником, хотя команде полезнее, чтобы он помог разобрать очередь.

Вывод для разделения. Единица, которая владеет только написанием кода, упрётся в чужую проверку и чужой выпуск. Когда режете команду, смотрите на весь путь до пользователя: кто в этой единице проверяет, кто выкладывает, кто решает, что делать дальше. Если какой-то шаг остаётся общим на несколько единиц, у него должна быть оговорённая ёмкость для каждой. Об этом рамка «Контракт ёмкости» из главы 7.

3.6. Проверочный список

Прогоните через эти пять вопросов любую схему из следующих глав до того, как показывать её людям.

Конвей. *Вопрос к схеме:* Совпадают ли границы команд с частями продукта? *Что посчитать:* Долю задач, которые задели больше одной команды. *Тревожный знак:* Через границу идёт больше трети задач.

Связи. *Вопрос к схеме:* Сколько пар и стыков даёт схема? *Что посчитать:* Пары внутри команд плюс стыки с весом 3. *Тревожный знак:* У соседнего варианта цена заметно ниже.

Предел головы. *Вопрос к схеме:* Сколько областей у команды и сколько знающих у области? *Что посчитать:* Матрицу знаний. *Тревожный знак:* Область с одним знающим; три сложные области у одной команды.

Отсутствие. *Вопрос к схеме:* Кто подменяет владельца и кто дежурит? *Что посчитать:* Недели без хозяина и недели дежурства на человека. *Тревожный знак:* Владелец без дублёра; дежурят меньше четырёх.

Узкое место. *Вопрос к схеме:* Где задачи ждут дольше всего? *Что посчитать:* Дни в каждом статусе у двадцати задач. *Тревожный знак:* Единица не владеет проверкой и выпуском.

Пороги в графе «Тревожный знак» — мои ориентиры. Подставляйте свои, но считайте до того, как объявите схему.

4. Известные подходы

Двенадцать подходов ниже — это двенадцать ответов на вопрос «по чему резать», и у каждого есть условие, при котором он ломается. Карточки одинаковые: суть, пример на компании из пятнадцати инженеров, сильная сторона, место поломки и то, что стоит взять компании на десять—пятьдесят инженеров.

Фреймворк обычно приходит в компанию одним словом. Кто-то возвращается с конференции и предлагает «сделать сквады» или «назначить владельцев потоков». Через месяц у команд новые названия, а задачи ждут в тех же очередях. Слово привезли, а условие, при котором подход работает, осталось в книге. Поэтому в каждой карточке главная строка — «Ломается, когда». Читайте её первой.

Если читать все двенадцать карточек некогда, начните с того, что болит.

Продукты запущены, но их никто не развивает

Где смотреть: Amazon, продуктовые команды с полномочиями.

Что взять: Человека, который не делит внимание, и метрику вместо списка задач.

Любая задача ждёт в трёх очередях

Где смотреть: Функциональные команды, Team Topologies.

Что взять: Проверку: доводит ли команда изменение до пользователя, никого не ожидая.

Новое делают одни, а чинят другие

Где смотреть: Развитие и эксплуатация.

Что взять: Дежурство по очереди для всех, включая авторов изменений.

У человека два начальника и два списка приоритетов

Где смотреть: Модель Spotify, unFIX.

Что взять: Два отдельных вопроса: с кем я работаю и кто мой руководитель.

Список задач бесконечный, цену задачи никто не называет

Где смотреть: Shape Up.

Что взять: Стол ставок и аппетит — сколько времени задача стоит.

Новые продукты и главный продукт мешают друг другу

Где смотреть: Пионеры, поселенцы, градостроители.

Что взять: Разные мерки для ставки и для ядра.

Команда выросла и вязнет в согласованиях

Где смотреть: Dynamic Reteaming.

Что взять: Разделение выросшей команды как обычное событие.

Платформа превратилась в отдел заявок

Где смотреть: Team Topologies, Haier.

Что взять: Внутренний договор — что платформа обещает и что получает.

Ответственность дали, а решать не дали

Где смотреть: Продуктовые команды с полномочиями.

Что взять: Задачу в виде проблемы и метрики. Права разбирает глава 5.

Первый пункт — частая боль компаний, у которых продуктов несколько. Одного назначения владельцев для неё мало: владельцу нужны подмена и права. Об этом главы 6, 8 и 9.

Функциональные команды

Суть. Люди собраны по профессии: фронтенд, бэкенд, тестирование, дизайн, эксплуатация.

Пример. Пятнадцать инженеров сидят в трёх отделах: шестеро в бэкенде, пятеро во фронтенде, четверо в эксплуатации. Кнопка «Скачать счёт» из главы 3 проходит через все три отдела и ждёт в трёх очередях.

Сильная сторона. Глубина мастерства, единые стандарты, понятный рост специалиста.

Ломается, когда любая заметная задача проходит через три-четыре очереди, и за результат для пользователя не отвечает никто.

Что взять. Профессиональные сообщества поверх команд. Фронтендеры из разных команд держат общие стандарты, смотрят код друг друга и учат новичков. Приоритеты каждому по-прежнему задаёт его команда.

Развитие и эксплуатация (Change / Run)

Суть. Одна команда делает новое, другая держит работающее: инциденты, поддержка, мелкие доработки.

Пример. Восемь человек делают новые возможности, семь держат работающий сервис. Новая оплата выходит в четверг, а в ночь на субботу её чинит человек, который видит этот код впервые.

Сильная сторона. Развитие защищено от пожаров, прод — от спешки. Для восьми—двенадцати человек это самый дешёвый разрез.

Ломается, когда команды вырастают до семи-восьми человек. Авторы изменений перестают чувствовать последствия, а Run превращается в склад всего, что никому не интересно. Правило Вернера Фогельса из Amazon «you build it, you run it» (интервью журналу ACM Queue, 2006) придумано как раз против такого разрыва.

Как это выглядит через год. Команда развития запускает продукт и уходит к следующему. Эксплуатация принимает его на поддержку: чинит сбои, но улучшать не обязана. У продукта есть дежурный и нет человека, который знает, зачем он пользователям. Так появляются продукты, которые работают и не растут. Счёт этой истории — в главе 6.

Что взять. Дежурят по очереди все, включая авторов изменений. А команду эксплуатации со временем превращайте в платформенную: вместо того чтобы чинить за других, она делает инструменты, с которыми остальные чинят и выпускают сами.

Scrum, LeSS и SAFe

Суть. Scrum задаёт кросс-функциональную команду, обычно до десяти человек, с одним владельцем продукта (Scrum Guide). LeSS оставляет один бэклог и одного владельца продукта на две—восемь команд (LeSS). SAFe собирает 50–125 человек в «поезд релизов» с общим планированием (SAFe).

Пример. Две команды по семь человек берут задачи из одного бэклога, то есть общего упорядоченного списка, который ведёт один владелец продукта. Раз в две недели обе показывают сделанное на общем демо. Кросс-функциональная значит, что внутри команды есть все умения, чтобы довести задачу до пользователя.

Сильная сторона. Общий ритм, общий язык и идея команды, которая делает задачу целиком.

Ломается, когда на пятнадцать инженеров надевают церемонии, рассчитанные на сто, или когда единственный владелец продукта становится узким местом для всех.

Что взять. Один бэклог продукта, пока команд две-три, и общий день демо.

Модель Spotify

Суть. Сквады владеют частью продукта, трайбы объединяют соседние сквады, чаптеры собирают людей одной профессии, гильдии — всех, кому интересна тема. Хенрик Книберг и Андерс Иварссон описали это в 2012 году.

Пример. Сквад «Оплата» из пяти человек владеет всем, что связано с деньгами. Его бэкендеры раз в неделю встречаются с бэкендерами других сквадов в своём чаптере. Руководитель чаптера отвечает за их рост и зарплату, а задачи им ставит сквад.

Сильная сторона. Автономия команд вместе с общим направлением и словарь, который понимают почти все.

Ломается, когда копируют названия. Лидер чаптера и владелец продукта дают человеку двух начальников, а это уже матричная структура: два руководителя и два списка приоритетов. Книберг сам предупреждал, что описал одну компанию в один момент и фреймворком это не считает (Atlassian).

Что взять. Гильдии как добровольные сообщества практики.

Team Topologies

Суть. Четыре типа команд: команда потока, платформенная, помогающая и команда сложной подсистемы. Три способа взаимодействия: совместная работа на время, «как услуга» и помощь с обучением (Team Topologies). Границы проводятся по потоку ценности и по тому, сколько команда способна удержать в голове.

Типы команд проще понять на примерах.

Команда потока. *Что делает:* ведёт свою часть продукта от идеи до пользователя. *Пример:* команда оплаты.

Платформенная. *Что делает:* делает инструменты, с которыми остальные выпускают сами. *Пример:* выкладка, мониторинг, доступы.

Помогающая. *Что делает:* приходит в команду на время и учит новому. *Пример:* два специалиста налаживают автотесты и уходят.

Команда сложной подсистемы. *Что делает:* держит часть, где нужна редкая экспертиза. *Пример:* модель, которая ранжирует ответы.

Способы взаимодействия тоже. Совместная работа на время: две команды месяц делают одну задачу вместе. «Как услуга»: одна команда пользуется готовым сервисом другой и почти с ней не разговаривает. Помощь с обучением: одна команда учит другую и уходит.

Проверка названия. В оригинале команда потока называется stream-aligned team, и этим словом легко назвать владельца направления. Проверка простая: доводит ли он изменение до пользователя, никого не ожидая? Если он один, выпуск согласует эксплуатация, а макеты ждут общего дизайнера, ответ «нет». Тогда перед вами владелец со второй ступени лестницы владения из главы 7.5, и ему нужен «дом»: команда, которая возьмёт на себя проверку, подмену и дежурство. Командой потока он станет, когда сможет выпускать сам.

Сильная сторона. Редкий подход, который проектирует и команды, и стыки между ними.

Ломается, когда команду называют командой потока, а выпустить изменение сама она не может. Или когда платформа превращается в отдел заявок.

Что взять. Почти всё, начиная с проверки: доводит ли команда изменение до пользователя, никого не ожидая.

Amazon, две пиццы и единоличный владелец

Суть. Маленькая команда целиком владеет своим сервисом. Позже Amazon сместил акцент с размера на руководителя, занятого только одной задачей (single-threaded leader), и на «отделимость» команды от остальных. Об этом рассказывают Колин Брайар и Билл Карр в книге Working Backwards (Inc.).

Пример. Запуск мобильного приложения объявлен главной ставкой года. У неё есть руководитель, который занят только ею, и команда, которая выпускает приложение, не дожидаясь остальных.

Сильная сторона. У каждой большой ставки есть человек, который не делит внимание.

Ломается, когда отделимость недостижима: общий код, общая база, общий релиз. Тогда владелец отвечает за то, чем не управляет.

Что взять. Правило: главная ставка года не бывает чьей-то подработкой.

Продуктовые команды с полномочиями

Суть. Марти Каган предлагает давать команде проблему и метрику вместо списка функций. Исследование и разработка живут в одной команде. Противоположность он называет feature team: ей спускают готовый список (SVPG). Рядом стоит «продуктовая тройка» Терезы Торрес: продакт, дизайнер и инженер вместе решают, что строить (Product Talk).

Пример. Вместо списка «экспорт в PDF, напоминания, тёмная тема» команда получает задачу: поднять долю тех, кто платит второй месяц, с 40 до 50%. Что для этого строить, она решает сама.

Сильная сторона. Решение принимается там, где есть знание о пользователе.

Ломается, когда полномочия объявлены, а дорожную карту по-прежнему пишет руководство.

Что взять. Формулировку задачи: какую проблему решаем и по какой метрике это поймём.

Shape Up

Суть. Работа идёт шестинедельными циклами. До цикла небольшая группа старших придаёт задаче форму и назначает ей аппетит: сколько времени она стоит. Руководители делают ставки за общим столом, и маленькая команда дизайнеров и программистов получает задачу целиком. Не успели за цикл — автоматического продления нет (Shape Up). Между циклами оставлены две недели передышки.

Пример. За столом ставок руководители решают: на оплату по подписке даём шесть недель, не больше. Трое получают задачу целиком и никого не ждут. Если к концу цикла не готово, вопрос возвращается на стол: продолжать или бросить.

Сильная сторона. Фокус и прямой разговор о цене задачи. Бэклог не копится.

Ломается, когда продукту нужен постоянный хозяин метрики и дежурство: команды каждый цикл собираются заново.

Что взять. Стол ставок и правило «не уложились — пересматриваем» для новых продуктов.

Пионеры, поселенцы, градостроители

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.