12+

Алексей Рахманов

IT-шишка

Записки СТО о людях, проектах и взрослении в IT



Алексей Рахманов

IT-Шишка

«Издательские решения»

Рахманов А.

IT-Шишка / А. Рахманов — «Издательские решения»,

«IT-Шишка» — честная история пути от разработчика до CTO, рассказанная через реальные ошибки, непростые решения и профессиональные победы.Как собрать сильную команду, научиться управлять людьми, запускать сложные проекты и справляться с кризисами, не теряя себя?Без идеальных рецептов и красивых теорий. Только реальные истории, набитые шишки и выводы, которые помогают двигаться дальше.

© Рахманов А.
© Издательские решения

Содержание

IT-ШИШКА 6
Введение 7
Комната два на три 8
Бизнес, который не выдержал 9
Шаман и Петербург 10
КОММЕНТАРИЙ ИЗ НАСТОЯЩЕГО 12
Восемь дней 13
КОММЕНТАРИЙ ИЗ НАСТОЯЩЕГО 15
Руководитель по подписи 16
Семь заказчиков у одной кнопки 17
Jira не лечит хаос 18
КОММЕНТАРИЙ ИЗ НАСТОЯЩЕГО 19
Человек между двумя языками 20
Игрок или тренер 21
КОММЕНТАРИЙ ИЗ НАСТОЯЩЕГО 22
Доверие и авторитет 23
Команда тоже взрослеет 24
Конец ознакомительного фрагмента. 26

IT-Шишка

Алексей Рахманов

© Алексей Рахманов, 2026

ISBN 978-5-0071-7103-8
Создано в интеллектуальной издательской системе Ridero

IT-ШИШКА

История о том, как я учился руководить людьми, проектами и собой
Алексей Рахманов · Авторская редакция, 2026

© Алексей Рахманов, 2026. Все права защищены.

Книга основана на личном опыте автора. Некоторые имена и узнаваемые детали изменены; отдельные диалоги восстановлены по памяти. Авторские комментарии из настоящего отражают взгляд на описанные события в 2026 году.

Введение

Четыре года назад я написал эту книгу. А потом убрал её в тумбочку.

Не было торжественного решения отложить публикацию. Просто работа не заканчивалась: появлялись новые проекты, люди, задачи, проблемы, которые нужно было решать прямо сейчас. Книга терпела. Так прошло четыре года.

Когда я перечитал рукопись, то заметил странную вещь: некоторые мысли по-прежнему готов защищать, а с некоторыми хочется спорить. И спорить не с редактором или читателем, а с самим собой — тем, который писал эти страницы.

Тогда мне казалось, что я уже немало понял об управлении. За прошедшие годы путь продолжился: от самостоятельных попыток написать первую программу и руководства разработкой и тестированием — до роли CTO. Ошибался, исправлял, снова ошибался. Но чем дальше двигался, тем яснее видел: руководитель не получает диплом о том, что теперь умеет руководить. Его каждый день проверяют люди и обстоятельства.

Это не учебник идеального менеджера. Здесь будут долги, несостоявшиеся проекты, переезды, ночи за кодом, споры с коллегами и решения, которые сегодня я принял бы иначе. Будут и инструменты: не как список функций из документации, а как ответы на конкретные проблемы, с которыми мы столкнулись.

Я расскажу, что делал тогда, почему считал это правильным и чему научился позже. Где-то оставлю прошлому право на ошибку. Где-то признаю, что всё ещё не знаю единственно верного ответа.

В IT шишки набивают быстро. Особенно если привык сначала делать, а потом разбираться. Я из таких. Ну что, поехали.

Шишка 1. Работать на дядю — можно
Не получилось одно — пробуй другое

Сколько себя помню, я хотел двух вещей: писать программы и работать на себя. В моей голове они прекрасно сочетались. В реальности — не очень.

После школы я решил поступать на информационную безопасность в Бауманку. Родители помогли найти преподавателей, я готовился, но экзамены провалил. Тогда подал документы в Московский институт стали и сплавов — МИСиС, на технологический факультет. Так человек, мечтавший об IT, оказался на пути к диплому инженера-металлурга.

Единственным предметом, на котором я чувствовал себя на своём месте, была информатика. Мы изучали BASIC, и меня затянуло. По информатике у меня были автоматы во всех семестрах, а в дипломе — единственная пятёрка. Параллельно я подрабатывал промоутером, грузчиком, оператором ПК. Тогда это казалось набором случайных занятий. Позже выяснилось, что общаться с людьми и не бояться непривычной работы тоже полезные навыки.

После института была армия. На военную кафедру я сознательно не пошёл: не нравилось то, что видел и слышал о её порядках. Служил в ВДВ. Мне повезло с частью и с людьми: рядом были спортсмены, ребята с высшим образованием. Службу вспоминаю с удовольствием. Но мысль о программировании никуда не делась.

Комната два на три

После армии я собрал свой первый сайт — цветочный интернет-магазин. Учился по книгам и видео, разбирался по ходу дела. Магазин не стал бизнесом, зато я увидел: можно придумать вещь и своими руками заставить её работать.

Потом мы с товарищем занялись информационными терминалами. Один отправился в детский патриотический центр, другой — в Русский археологический музей в Иерихоне. Работали в комнате размером примерно два на три метра. Один готовил материалы в графическом редакторе, второй собирал интерактивные области в Adobe Flash; программировали на ActionScript. Сегодня такой стек вызывает ностальгию, а тогда это были доступные нам инструменты.

В патриотическом центре мы предложили показывать материалы о первой помощи и военной истории. Позже на празднике представили идею электронного тира: лазерная указка вместо выстрела, попадание отмечается на экране. Мы назвали цену, которая теперь кажется смешной. Заказа не получили: всё нужное уже закупили.

У меня были технические идеи, желание работать и полное непонимание того, как продают продукт. Я думал: если сделать полезную вещь, её обязательно купят. Оказалось, полезность нужно ещё суметь объяснить — причём тому, кто принимает решение.

Следом мы сделали электронный дневник — задолго до того, как такие системы стали привычными в школах. Ходили к администрациям в пиджаках и галстуках, предлагали пользоваться бесплатно. В ответ слышали вполне земные вопросы: кто будет вносить данные, зачем это школе и кто оплатит дополнительную работу? Мы приходили с технологией, а люди спрашивали про процесс. И были правы. Тогда я этого не понимал.

Бизнес, который не выдержал

Магазин не взлетел. Терминалы оказались разовыми продажами. Электронный дневник никому не понадобился. Устроиться разработчиком без профильного диплома и официального опыта не получалось.

Мы с друзьями открыли автосервис по покраске автомобилей. Я снова занимался тем, что умел: договаривался со страховыми компаниями, привлекал заказы. И, конечно, написал программу — для заказ-нарядов и расчёта зарплат. Даже когда жизнь уводила меня от IT, я всё равно возвращался к коду.

Дела шли неплохо, пока один из контрагентов не начал задерживать оплату. Машины нужно было ремонтировать, зарплаты — платить, деньги — вкладывать в оборот. Мы взяли крупный кредит. Потом контрагент обанкротился, так и не расплатившись.

Я остался с долгами почти в пять миллионов рублей. Жена была беременна. Бизнес закрывался. В такие моменты слова «предпринимательская свобода» звучат иначе. Я хотел работать на себя — и вдруг обнаружил, что не управляю почти ничем.

Шаман и Петербург

Когда сервис уже закрывался, знакомый отвёз меня к человеку, которого называли шаманом. В шаманов я не верил тогда и не верю сейчас. Но одну его мысль запомнил: хватит заниматься машинами, займись наконец разработкой.

Не знаю, что именно меня задело в этих словах. Возможно, он просто вслух произнёс то, что я давно знал сам.

Жене не нравилось в Москве, а меня там уже мало что держало. Мы переехали в Санкт-Петербург. Мне было двадцать семь. Скоро должен был родиться сын. Работы не было, долги оставались. Иногда звонили кредиторы; случались и угрозы. Я не видел красивого плана выхода из ситуации. Нужно было просто начинать зарабатывать.

Я устроился мастером-консультантом кузовного участка. Принимал автомобили, передавал их в ремонт. После собственных проектов и бизнеса это было болезненно для самолюбия. Я ведь столько лет повторял, что никогда не стану работать «на дядю». Теперь работал — и был рад, что вообще есть зарплата.

Но старые навыки никуда не делись. Я написал программу, которая сообщала клиентам о ходе ремонта, и связался со знакомыми страховыми компаниями, чтобы они направляли автомобили ко мне. Через три месяца подошёл к директору и заявил, что готов занять место уходящего руководителя. Должность мне не дали. Зато предложили заниматься привлечением корпоративных клиентов.

Я ещё не умел отличать уверенность от самоуверенности. Мне казалось: если хорошо работаю, следующий шаг мне обязаны предоставить. Реальность оказалась сложнее.

Код всё равно находил меня

Позже я сделал программу для учёта автомобилей на мойке, поссорился с руководством и ушёл. Несколько месяцев перебивался фрилансом. Затем пришёл в IT-компанию, которая собиралась делать софт для автомастерских. Я рассчитывал программировать, но, по моим ощущениям, от меня прежде всего хотели получить знания об автомобильном бизнесе. До разработки дело так и не дошло.

Я снова вернулся в автомобильную отрасль — уже менеджером. Через год стал лучшим в группе компаний и добился появления новой должности: руководителя отдела по работе с корпоративными клиентами. Доход был хорошим, работа знакомой, а удовлетворения всё равно не хватало.

И там я снова написал программу: она рассчитывала пробег автомобилей и формировала списки клиентов, которым пора было напомнить об обслуживании. Я мог сменить должность, компанию и город, но продолжал искать повод что-нибудь автоматизировать.

Однажды рядом с одним из дилерских центров, за которые я отвечал, подвернулась возможность поговорить с финансовым директором медицинского центра. Я предложил написать систему управления клиникой. Она согласилась.

Днём я занимался дилерскими центрами. Ночью писал код. Так продолжалось восемь месяцев. Спал мало, работал много. Тогда я воспринимал это как необходимую цену за возможность вернуться в IT. Сейчас я бы куда осторожнее советовал кому-либо повторять такой режим.

Систему запустили в сети клиник Санкт-Петербурга. Она работает и сейчас, а я до сих пор участвую в её поддержке и развитии. После этого мне предложили возглавить веб-проекты. Я ушёл из автомобильного бизнеса. Впервые за долгое время работа и то, чем я действительно хотел заниматься, совпали.

Постепенно удалось рассчитаться с долгами. Компания оплатила мне обучение; я получил подтверждение профессиональной квалификации. Затем были другие проекты — система

для биобанка и система управления автопарком. Они тоже продолжают работать; я по-прежнему поддерживаю и развиваю их. Казалось, наконец всё складывается.

КОММЕНТАРИЙ ИЗ НАСТОЯЩЕГО

Сегодня, уже в роли CTO, я особенно ясно вижу: релиз — не конец проекта, а начало его взрослой жизни. Система становится частью бизнеса, обрастает пользователями, интеграциями и ограничениями. Сделать её однажды — только половина работы. Гораздо сложнее годами менять её, не разрушая то, на что уже опираются люди.

Восемь дней

Потом началась пандемия. Команду сократили, а я остался руководителем без сотрудников. Зарплату уменьшили на тридцать процентов. Мы с женой и двумя детьми уехали в деревню в Псковской области. Мобильная связь там почти не ловила, поэтому для работы я купил спутниковую антенну. Деревня, старенький компьютер, интернет через спутник — и снова код.

В это время мне предложили сделать корпоративный портал для большого мероприятия. Нужны были новости, опросы, голосования, фотографии, комментарии и интерактив для сотрудников в зале и на удалёнке. Я оценил работу в четыре-пять месяцев. Заказчик ответил, что у них неделя: предыдущий подрядчик не справился.

Согласиться на такое — сомнительная идея. Но мне нужна была работа, а ещё хотелось доказать, что я могу. Мы договорились использовать готовую вёрстку и сосредоточиться на том, без чего мероприятие не состоится.

Я закончил за восемь дней. В какой-то момент не спал трое суток подряд и буквально отключался от усталости. Портал выполнил задачу заказчика. Код был далёк от образцового: монолит, жёстко зашитые решения, временные обходы. Это был не эталон архитектуры, а продукт под конкретный срок и конкретную потребность.

Тогда я гордился тем, что справился. И до сих пор горжусь результатом. Но теперь вижу и другую сторону: героизм одного человека не должен становиться нормальным способом разработки. Если система держится на чьём-то недосыпе, это риск, а не управленческая технология.

После проекта мы ещё работали с этим заказчиком. А вскоре написал старый товарищ — тот самый, с которым мы когда-то делали терминалы и электронный дневник. Он позвал меня в стартап разрабатывать сайт.

Я согласился. Стартап вырос, вместе с ним росли задачи и команда. Со временем я возглавил разработку и тестирование в крупной туристической компании. На момент написания первой версии книги в моей команде было больше шестидесяти человек. Теперь я смотрю на тот период уже из роли CTO — с другим масштабом ответственности и совсем другим отношением к управленческому героизму.

Когда-то я хотел любой ценой не работать «на дядю». В итоге именно работа в найме дала мне проекты, людей, обучение и возможность заниматься любимым делом. Проблема была не в форме занятости. Проблема была в моей уверенности, будто я заранее знаю единственный правильный путь.

Что я забрал из этой шишки

Первое: технически хороший продукт не обязательно кому-то нужен. История электронного дневника научила меня спрашивать не только «можем ли мы сделать?», но и «кто будет этим пользоваться и зачем?».

Второе: прошлый опыт не пропадает. Автомобильный бизнес научил меня продажам, переговорам и ответственности за деньги. Позже это пригодилось в IT не меньше, чем знание языков программирования.

Третье: разовый подвиг не заменяет процесс. Портал за восемь дней был удачным исключением. Команду, способную стабильно делать сложные продукты, на исключениях не построишь.

И главное: карьерный путь редко выглядит как прямая линия. Мой оказался похож на череду поворотов, тупиков и возвращений. Но именно он привёл меня к людям, с которыми предстояло учиться самому трудному — не писать программы, а руководить теми, кто их пишет.

Шишка 2. Один в поле не воин

Один человек может запустить продукт

Когда я пришёл в стартап фронтенд-разработчиком, мне казалось, что главная сложность уже позади. Я наконец занимался тем, к чему столько лет пробивался: писал код, видел результат и понимал, зачем нужна моя работа.

Дальше, как мне тогда представлялось, всё было просто. Нужно быстрее разрабатывать, исправлять ошибки и постепенно добавлять новые функции. Если задач станет больше, будем работать интенсивнее.

Это заблуждение держалось недолго.

Один человек действительно может собрать сайт или первую версию продукта. Иногда он способен поддерживать её довольно долго. Я сам так работал в медицинской клинике: после сокращения команды на мне остались система, телефония, 1С и всё, что внезапно переставало работать. Но это была уже не разработка в полном смысле слова. Я поддерживал жизнедеятельность. На развитие не оставалось ни времени, ни головы.

Когда продукт становится важным для бизнеса, одного универсального специалиста почти всегда оказывается мало. Кто-то должен разбираться с интерфейсом, кто-то — с серверной частью и данными. Кто-то должен проверить, что новая функция не сломала старую. Нужен человек, который переведёт пожелание бизнеса на язык конкретной задачи. По мере роста появляются инфраструктура, безопасность, дизайн, управление релизами и десятки других обязанностей.

Не обязательно сразу нанимать на каждую обязанность отдельного сотрудника. В небольшой команде один человек неизбежно совмещает несколько ролей. Важно другое: увидеть эти роли и честно признать, какие из них пока некому выполнять.

Однажды ко мне обратились представители компании, работающей с промышленными кондиционерами. Они хотели создать систему, которая позволяла бы получать данные с оборудования и управлять температурой на объектах в разных регионах. Сделать первую версию вдвоём с разработчиком было возможно. Сделать устойчивый коммерческий продукт силами двух человек — уже гораздо сложнее.

Прототип может держаться на энтузиазме. Продукту нужны качество, эксплуатация и ответственность. Чем больше денег и процессов бизнеса завязано на системе, тем опаснее надежда на одного героя.

КОММЕНТАРИЙ ИЗ НАСТОЯЩЕГО

Сегодня, уже в роли CTO, я смотрю на одиночный героизм прежде всего как на риск. Если один человек знает систему, выпускает релизы и в случае аварии единственный способен её поднять, это не признак его незаменимости. Это признак того, что организация построила зависимость от одного человека. Когда-то мне нравилось быть тем самым человеком. Теперь моя задача — сделать так, чтобы компания без него не остановилась. Даже если этот человек — я сам.

Инструмент: пять вопросов о составе команды

Когда мне нужно понять, действительно ли команде не хватает людей, я смотрю не только на количество сотрудников. Полезнее ответить на пять вопросов:

1. Кто превращает запрос бизнеса в понятную задачу?
2. Кто отвечает за техническое решение и качество кода?
3. Кто проверяет результат независимо от автора изменения?
4. Кто отвечает за выпуск, инфраструктуру и работу системы после релиза?
5. Какие знания сосредоточены у одного человека и что произойдёт, если он на время выпадет?

Ответы быстро показывают, где реальная нехватка компетенций, а где проблема не в численности, а в неразделённой ответственности.

Руководитель по подписи

Наша первая команда была скорее инициативной группой. В ней были мой товарищ — идейный вдохновитель, представитель компании-заказчика, бэкенд-разработчик, специалист по базам данных и я, отвечавший за фронтенд.

Дизайнера не было. Аналитика не было. Системного анализа тоже не было. Зато была энергия людей, которые хотели что-то запустить и пока не представляли, сколько разных специалистов обычно участвует в создании продукта.

Первую концепцию мы обсуждали с заказчиком по Skype. Никакого подробного технического задания нам не дали. Во время разговора рисовали прямоугольники: здесь будет поиск, здесь — личный кабинет, здесь — ещё что-нибудь, что мы пока не до конца понимаем. По существу это был первый грубый прототип, собранный прямо в разговоре.

Сейчас я бы начал с целей продукта, пользователей, ограничений и критериев успеха. Тогда мы начали с квадратов на экране. И, что характерно, какое-то время этого хватало.

На одной из встреч с представителями заказчика меня представили коротко:

— Это Алексей. Он руководитель.

Формально руководителем стартапа был другой человек. Но после этих слов я решил, что вопрос закрыт, и поменял подпись в корпоративной почте.

Так я впервые стал руководителем — сначала в подписи, а уже потом в реальности.

Сегодня эта история меня смешит. Название должности можно получить за одну минуту. На то, чтобы научиться руководить, уходят годы. В тот момент я ещё не умел выстраивать процесс, делегировать, давать обратную связь или защищать команду от хаоса. Зато был готов брать ответственность и не прятаться, когда что-то не работало. Видимо, с этого всё и началось.

Начальный состав продержался недолго. Один из разработчиков почти сразу ушёл. Тестировщиков у нас не было — да и тестировать поначалу было особенно нечего: задачи существовали в разговорах, процессы — в нашей голове, а решения принимались на ходу.

В компании-заказчике уже был собственный IT-отдел. Часть коллег оттуда подключили к нашему проекту. Другим источником людей стала Telegram-группа, которую я создал во время обучения программированию. В неё входили ребята с похожим стеком. Позже я сам стал наставником в онлайн-школе и приглашал туда сильных студентов с новых потоков.

Когда понадобилось усиление, я написал знакомым. Несколько человек согласились присоединиться. Так вокруг проекта собрались не случайные исполнители, а люди, с которыми уже существовало доверие и общий профессиональный язык.

Нас стало двенадцать.

Из этой истории я вынес очень практичный вывод: контакты хороших специалистов нельзя вспоминать только в тот момент, когда открылась вакансия. Профессиональное окружение формируется задолго до найма.

Семь заказчиков у одной кнопки

Пока команда была маленькой, мы работали почти по-дружески. Формально задачи никто не назначал. Мы обсуждали, кто что возьмёт, и двигались дальше. Для старта этого оказалось достаточно.

Проблемы начались, когда проект стал заметен внутри компании.

К обсуждениям подключались новые отделы. У каждого были свои цели, сроки и представление о том, каким должен быть продукт. Один просил изменить кнопку, второй хотел вернуть её обратно, третий считал, что сначала нужно переделать весь экран. Иногда требования прямо противоречили друг другу.

У каждого отдела появлялся собственный чат с разработчиками. Чатов становилось всё больше. Задачи жили в сообщениях, устных договорённостях и личной переписке. Пока команда делала работу для одного подразделения, другое уже спрашивало, почему его просьбу опять отложили.

Мы старались никого не расстроить и в результате всё чаще расстраивали всех.

Это была моя первая встреча с простой управленческой истиной: если у продукта много заказчиков и нет единого приоритета, команда не может принять правильное решение. Она может только угадывать, кто сегодня громче.

Параллельно рос сам продукт. До нашего появления в компании уже создавалась административная система с информацией о странах, городах, отелях и номерах. Ею занимался один разработчик, и объём явно превышал возможности одного человека.

Я позвал коллегу поговорить.

— Либо ты успеваешь закончить к сроку, либо дальше проект забираем мы.

— Я не успею, — честно ответил он.

Мы забрали систему и продолжили её развитие. Объём данных оказался огромным. Отдельная команда собирала сведения об отелях и номерах настолько подробно, что можно было узнать даже, есть ли в конкретном номере детский горшок. Добавлялись фотографии, фильтры и всё, что помогало человеку выбрать не абстрактный «номер такой-то», а подходящий вариант отдыха.

Постепенно мы переставали быть маленькой группой, которая делает что-то непонятное на периферии компании. На нас ложилась всё более значимая часть продукта. Вместе с влиянием росла цена нашего хаоса.

Jira не лечит хаос

В какой-то момент стало очевидно: управлять задачами через десятки чатов больше нельзя. Мы внедрили Jira.

Само решение выглядело зрелым. Теперь у задачи могло быть описание, ответственный, статус и история изменений. Появилось одно место, где можно было увидеть работу команды. Нам казалось, что вместе с Jira мы внедрили процесс.

На деле мы просто перенесли хаос из чатов в систему управления задачами.

Мы не оценивали трудоёмкость, не ограничивали количество одновременной работы, не договорились, кто расставляет приоритеты. В Jira быстро накопился большой список задач, но бизнес продолжал приходить с новыми пожеланиями в прежнем режиме.

Мы убрали разработчиков из общих чатов, чтобы они могли сосредоточиться. Тогда представители бизнеса начали писать им напрямую:

— Слушай, а можешь быстро поменять цвет у кнопки?

Формальный процесс существовал, но обходился одним личным сообщением.

Главной проблемы Jira не решала: у нас не было человека, который отвечал бы за приоритет продукта. Позже я узнал для этого роль Product Owner — владелец продукта. Его задача не просто собирать пожелания, а принимать решение, что действительно нужно делать, в какой последовательности и ради какого результата.

Пока такого центра решения нет, бэклог становится складом чужих надежд. В нём может лежать тысяча задач, но это не означает, что команда понимает, куда движется.

КОММЕНТАРИЙ ИЗ НАСТОЯЩЕГО

Сейчас я часто повторяю: инструмент не создаёт процесс, он только делает существующий процесс видимым. Если в компании нет правил приоритизации, Jira покажет хаос аккуратными карточками. Если никто не отвечает за результат продукта, новый статус задачи этого человека не заменит. Начинать нужно не с настройки полей, а с договорённостей о том, кто принимает решения и по каким критериям.

Со временем Jira всё же стала приносить пользу. Пожелания бизнеса начали попадать в единый контур, проходить согласование и анализ. Мы привлекли начинающих тестировщиков, которые помогали разбирать входящий поток и одновременно погружались в продукт. Появились метрики, стало видно состояние бэклога и количество задач, которые слишком долго ждут решения.

Jira не спасла нас. Но дала поверхность, на которой стало возможно строить систему.

Инструмент: минимальные правила рабочего контура

Чтобы система управления задачами не превратилась в дорогой список дел, нам понадобились четыре договорённости:

— у задачи есть единая точка входа, а личное сообщение не считается постановкой;

— до разработки запрос проходит анализ и получает понятный ожидаемый результат;

— приоритет определяет конкретный ответственный со стороны продукта или бизнеса;

— статусы и метрики нужны для принятия решений, а не для имитации контроля.

Программное обеспечение здесь вторично. Эти правила можно нарушить и в Jira, и в таблице, и на бумажной доске.

Человек между двумя языками

Перед нами по-прежнему стояла «тысяча миллионов» задач. Я распределял поток между фронтендом и бэкендом, отвечал на вопросы, сам писал код и пытался одновременно удерживать общую картину.

Затем в команде появилась аналитик. Позже она возглавила группу аналитиков, но тогда для нас важнее всего было другое: впервые появился человек, который мог разговаривать и с бизнесом, и с разработкой, не теряя смысл по дороге.

Она уточняла требования, описывала задачи, сопоставляла желания заказчиков с возможностями команды и помогала договариваться о последовательности работы. По сути, закрывала сразу две потребности: бизнес-анализ и часть проектного управления.

Она же принесла в нашу работу спринты и более осознанный Agile-подход. Не лозунг «теперь мы гибкие», а понятный ритм: выбрать объём работы, договориться о результате, выполнить, показать и скорректировать дальнейшие шаги.

До её прихода мы часто принимали задачу в виде фразы «нужно сделать сайт» или «нужно переделать раздел», а детали выясняли уже в разработке. После — между желанием и кодом появился этап осмысления.

Мне стало легче не только потому, что уменьшился объём работы. Появился человек, с которым можно было разделить неопределённость. Она действительно прикрывала нам спину: не обещала бизнесу невозможное и не позволяла разработке прятаться за техническими формулировками.

Тогда я впервые ясно увидел, насколько сильно окружение влияет на руководителя. Сильный специалист рядом не уменьшает твою значимость. Он увеличивает масштаб задач, с которыми вы способны справиться.

Игрок или тренер

Несмотря на рост команды, я продолжал писать код. Мне нравилась роль играющего тренера: утром распределять задачи, днём обсуждать решения, вечером самому что-нибудь разработать.

Проблема в том, что у такого режима быстро заканчивается вечер.

Чтобы действительно помогать разработчикам, нужно погружаться в контекст. У нас было несколько стеков, множество параллельных задач и постоянный поток запросов от бизнеса. Нельзя одновременно глубоко разбираться во всём, принимать управленческие решения, проводить встречи и оставаться продуктивным разработчиком.

Я даже пошёл учиться на курс для тимлидов, надеясь найти правильную формулу совмещения. Моё мнение после курса не изменилось: руководителю технической команды необходимо понимать разработку, но не обязательно лично писать весь критичный код.

Полностью отрываться от технологии тоже опасно. Если руководитель раздаёт указания, не понимая, о чём говорит команда, авторитет быстро превращается в должностную инструкцию. Но и пытаться каждый раз доказать компетентность, забирая задачу у сотрудника, — другой тупик.

Мне пришлось учиться говорить не «я знаю, как правильно», а «вот риск, который я вижу; давайте вместе найдём решение». Экспертом в конкретной области может быть разработчик, тестировщик или DevOps-инженер. Моя задача — собрать экспертизу, обозначить ограничения и добиться решения, за которое команда готова отвечать.

Постепенно я отходил от операционного программирования. Чтобы не потерять техническую форму, оставлял себе экспериментальные проекты — те, где можно попробовать новый инструмент и которые не остановят основной бизнес, если я отложу их на неделю.

КОММЕНТАРИЙ ИЗ НАСТОЯЩЕГО

В роли СТО мне уже не нужно доказывать техническую состоятельность количеством написанных строк. Моя работа — повышать качество решений всей системы: помогать командам видеть риски, создавать условия для сильных специалистов и вовремя задавать вопросы, которые не помещаются в границы одного проекта. Техническая глубина по-прежнему важна. Но теперь она нужна мне не для того, чтобы сделать работу за людей, а чтобы понимать цену их решений.

Доверие и авторитет

Когда нас стало двенадцать, я впервые почувствовал: это уже моя команда. Не в смысле собственности, а в смысле ответственности. Я знал людей, понимал, кому что интересно, и хотел, чтобы им было хорошо работать вместе.

На собеседованиях я говорил кандидатам:

— Если примете наш оффер, считайте, что попали на теплоход «Счастье». Спрыгнуть можно, но движется он только в сторону развития.

Фраза была самоуверенной, но честной. Мы действительно много работали и часто горели, однако ощущали, что строим что-то важное.

Мне не приходилось специально изображать авторитет. Я старался быть предсказуемым: выполнять обещания, разбираться в проблеме до решения и не перекладывать ответственность на сотрудника, если сам согласовал невозможный срок.

Большую роль играла личная система работы. После книги Максима Дорофеева «Джедайские техники» я начал жёстче фиксировать входящие задачи, а идеи из книги Барбары Оакли «Думай как математик» помогли лучше понять, как чередовать сфокусированную работу и паузы. Постепенно я собрал собственную систему планирования. В календарь и заметки попадало почти всё — от рабочих встреч до бытовых просьб.

Это не делало меня идеальным руководителем. Но снижало количество обещаний, которые я забывал. А доверие часто складывается именно из таких мелочей.

Команда тоже взрослеет

Позже, уже на управленческом обучении, я познакомился с моделью Брюса Такмана: forming, storming, norming, performing. Оказалось, наш путь давно описан.

На стадии формирования люди знакомятся, ищут своё место и ждут от руководителя направления. Именно такой была наша первая группа: много энергии, мало ясности и почти все решения сходятся в одной точке.

Затем начинается шторм. Появляются разногласия, борьба за подходы и вопросы о границах ответственности. Для нас этим штормом стали противоречивые требования бизнеса, рост команды и попытки понять, кто вообще имеет право принимать решения.

Нормализация начинается, когда команда договаривается о правилах. Возникают роли, ритм встреч, критерии готовности и способы разрешать конфликты. Jira, аналитика и первые процессы были не самой системой, а признаками того, что мы к ней подбираемся.

На стадии результативной работы команда уже не тратит большую часть энергии на внутренние договорённости. Люди понимают общую цель, доверяют друг другу и могут самостоятельно решать значительную часть вопросов.

Важное наблюдение: смена состава возвращает часть вопросов заново. Даже зрелой команде после прихода нового человека приходится снова договариваться о ролях, границах и способах взаимодействия. Руководителю полезно не раздражаться из-за этого, а понимать, какой этап группа проходит сейчас.

Инструмент: короткая диагностика зрелости

Я задаю себе четыре вопроса:

1. Команда ждёт от руководителя ответа почти на каждый вопрос или умеет договариваться сама?
2. Конфликты обсуждаются открыто или уходят в личные чаты и молчание?
3. Правила работы существуют в общей практике или только в документах?
4. Общий результат важнее локальной правоты отдельных специалистов?

Эти ответы полезнее, чем попытка навесить на людей ярлык «слабая» или «сильная команда».

Роли, которых нет в штатном расписании

Должность описывает только часть того, что человек делает в команде. Разработчик может быть тем, кто постоянно придумывает новые подходы. Тестировщик — человеком, который удерживает всех от поспешных решений. Аналитик — неформальным координатором, к которому идут за советом.

Со временем я стал разделять несколько типов ролей.

Функциональная роль отвечает на вопрос «какую профессиональную работу человек выполняет»: разработчик, аналитик, тестировщик, дизайнер, DevOps-инженер.

Процессная роль показывает, за какой участок движения работы он отвечает: владелец продукта, менеджер проекта, Scrum-мастер, участник проектной команды.

Командная роль описывает привычный способ помогать группе: генерировать идеи, координировать, исследовать, доводить до конца, поддерживать отношения, проверять качество.

И, наконец, есть социальная роль: коллега, носитель опыта, наставник, человек, который помогает новичкам чувствовать себя частью команды.

Эта оптика оказалась полезной, потому что позволяла видеть пробелы. Если в группе пять генераторов идей и ни одного человека, который любит завершать работу, идей будет много, а релизов — меньше. Если все сильны в исполнении, но никто не берёт на себя координацию, руководитель быстро становится единственным диспетчером.

Особенно внимательно нужно относиться к конфликтующим ролям. Когда человек сам разрабатывает изменение и остаётся единственным, кто подтверждает его качество, независимой проверки фактически нет. В маленькой команде совмещения неизбежны, но риск должен быть видимым, а не спрятанным за фразой «у нас все универсалы».

Инструмент: матрица ролей

Для небольшой команды достаточно простой таблицы. Она не оценивает людей, а показывает, какие виды ответственности уже закрыты и где остаются пробелы.

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.