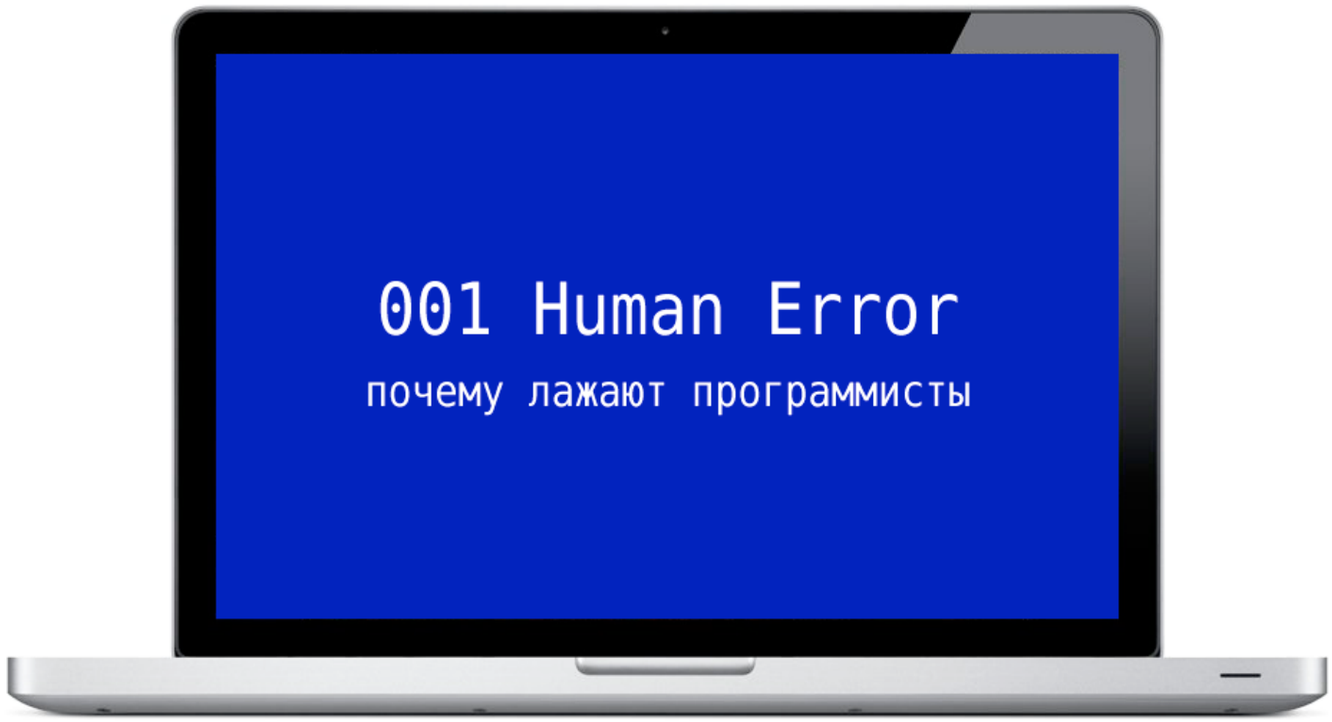


Алексей Атемасов

Введение

Я написал бóльшую часть этой книги ещё в 2012 году. После этого почти полностью её переписал и понял, что на эту тему можно написать полноценную книгу. Но к написанию полноценной книги я пока не готов. Поэтому здесь будет лишь часть того, что попадает под заголовок.
Назовем это «Версия 0.1»

Для кого эта книга? В первую очередь, для самих программистов (как это ни странно). Абсолютно разного уровня, — не важно, джунов или сеньоров. Ещё для менеджеров. Ещё для QA. В общем, почти для всех. Для тех, о ком написано, и для тех, кто с этими людьми работает.

Здесь описаны достаточно очевидные вещи. С другой стороны, любая книга может быть полезна одним и не нести никакого нового смысла для других. Мне кажется, есть целая категория людей, кому эта книга может помочь в будущем. Собственно, она и написана в помощь программистам, а не как рассказ о том, какие они плохие. Пускай местами даже достаточно жёстко.

Я человек, который очень много проработал в условиях продуктовых компаний или хотя бы в аутстаффе под продукт. Поэтому многие вещи будут рассматриваться сквозь призму продукта как исходной точки. Я свято верю, что эволюция нашего IT должна прийти к продуктовой разработке.
С другой стороны, я прекрасно понимаю, что сейчас 90% IT — это аутсорс, и не учитывать это было бы глупо. Я попытался рассмотреть описанные проблемы с разных сторон.

В процессе обработки черновика книги я постарался по максимуму заменить всю нецензурную лексику на более мягкие аналоги. Но иногда «из песни слов не выкинешь» — не всегда можно передать глубину мысли (в надежде, что она есть), заменив крепкое словцо «культурным», мать его, аналогом. В общем, 18+, с вашего позволения.

Программист vs Исполнитель

Программисты нам тут нахер не нужны.
Исполнителем, Вава, исполнителем.
(почти Пелевин)

Я не люблю это слово, но ввиду отсутствия в русском языке слова «выполнитель», приходиться использовать его — Исполнитель. Грубо говоря, человек, который приносит результат, это если вкратце.

Уровень исполнителя (в отличие от программиста) — это не количество фреймворков, языков программирования и «чем отличается абстрактный класс от интерфейса», а уровень задач, которые человек способен сделать самостоятельно. Весь технический бэкграунд — это уже подробности. Идеальный сферический исполнитель в вакууме — это человек, которому можно поставить задачу «сделать зашибись» безо всяких дополнительных подробностей, и он таки сделает «зашибись». Технические навыки, безусловно, нужны (мы всё-таки говорим о людях, которые пишут код), но не являются единственным критерием уровня исполнителя.

Когда вы садитесь в самолет, вас не интересует, на каких тренажёрах обучался пилот, на каких самолетах он летал перед этим и какой у него налётный стаж. Вам нужно взлететь, пролететь и сесть. Первое и последнее самое критичное. Тут определяется уровень пилота. Человек взял на себя ответственность и выполнил свою задачу.

Так вот, исполнитель.

Это человек, которого можно поставить на «передовую». То есть, он может обсудить, предложить решения (разные) и исполнить любое из них, — то, которое больше всего подходит бизнесу. Этот человек должен понимать бизнес (хотя бы отчасти). И это не обязательно Senior с технической точки зрения. Совсем не обязательно.

Так вот. Почему же программисты часто не являются и не становятся Исполнителями, почему очень часто это просто бесполезные Senior’ы и тем более бесполезные Middle’ы? Список описанных ниже проблем стоит не в порядке приоритета и важности, а скорее в том порядке, в котором они всплывали у меня в голове.

Часть 1

Цветочки

Программисты не любят писать и читать письма

Это просто бич. Мне кажется, каждого программиста нужно на пару неделек посадить на место менеджера и дать в подчинение пару-тройку удалённых фрилансеров. Тогда отношение к написанию писем резко изменится.

Когда человек находится вне твоей видимости, отсутствие постоянного фидбека создает впечатление, что он ничего не делает, — или делает, но слишком медленно. В общем, скверное впечатление. Поэтому, когда человек «на другом конце провода» не получает ответа на свои письма в течение дня, а то и двух-трех, он начинает нервничать. Возможно, он задал простой вопрос, который требует 10 минут внимания, но у него из-за этого вопроса блочится работа. А у программиста задача «прочитать письмо и ответить» попала в стек, и пока весь предыдущий стек не очистится, — ответа не жди.

Я знаю, что программисты, в среднем по больнице, товарищи не сильно коммуникабельные. Далеко за примером ходить не надо — я и сам таким был (да и местами есть до сих пор). Вот и эту книжку писал целых шесть лет. Можно часто услышать оправдания типа «я не знаю, что ответить», «я сильно занят [кодингом]» и, естественно, «ой, я забыл».

А теперь переворачиваем ситуацию: тебе нужны ответы на твои критичные вопросы по требованиям и ты написал письмо. Ты не можешь начать работать без этой информации. И через три дня тебе приходит: «Ой, прости, я забыл тебе написать, был слишком занят. Даже не знаю, что ответить, сделай как считаешь нужным». Получите и распишитесь.

Более того, подобный «мискоммуникейшн» грозит тем, что программист просто-напросто сделает не то, что от него требует бизнес. Подробнее об этом будет в разделе про «идеальный код».

Найдется много пожеланий и даже инструкций, которым стоит следовать при переписке. Но есть одно незыблемое: отвечай на письмо в течение дня. Даже если ответ звучит как «я прочитал, мне нужно время подумать». Это значит, что ты не забил на письмо, не потерял его, оно не попало в спам и ты в курсе темы. Это почти банальный знак приличия — как поздороваться с человеком при встрече. И да, письма нужно читать каждый день, это я так, на всякий случай.

Есть ещё один прием, которым стоит пользоваться как можно чаще: писать письма простым (простейшим) языком. Как будто на том конце «провода» сидит чувак-дуб-дерево с уровнем 7-го класса школы. Практически любое техническое письмо можно

упростить. И нет абсолютно никакого смысла заваливать грудой терминов очень далекого от программирования человека.

Вот тебе пример: ты затеял ремонт в квартире. Или даже в частном доме (красавчик, чё!). Ты не будешь делать всю эту шпаклёвку, разводку, стяжку и прочую хрень сам — у тебя есть прораб. Просто потому что его дело — делать ремонт, а твоё — заработать денег на этот ремонт.

Но не всё так просто — тут начинаются «прорабские» вопросы: *«Трёхжильный или пятижильный провод? А надо выводы под унитаз на 4 см поднимать? А кирпич-то нужен из тугоплавкой шамотной глины. Чё, не понятно, что надо было с запасом плитку закупать? Ну... это, того... вот эта херня, ну шо тут непонятно? Та посмотри, тут термоголовка не влезет — ежу ясно!»*

И естественно, моё самое любимое:
«— Надо денег на материалы. — Сколько? — Нууу... <дальше идёт неразборчивый список всего на свете> — Хорошо, так сколько? — Нууу...»
Очень похоже на:
«— Мы не успеваем сделать это вовремя. — А сколько ещё надо времени? — Нууу...»

Это далеко не последнее сравнение программистов со строителями в этой книге.

И ещё. Пиши коротко. Если собеседнику будут нужны детали — он сам спросит.

Типа лирическое отступление. А нафиг оно всё вообще надо? Вот я (то есть, ты), такой классный из себя программист, сижу, педалю свой классный код с классным выражением своего самодовольного лица, а тут какие-то письма.

Ну письма и письма, хоть пять, хоть десять. И сквозь это всё идёт одна шальная мысль: «Ну блин, это не моя работа — я код писать должен». Чувак! **Это — твоя работа!** И только от тебя зависит, когда ты её сделаешь — когда будет удобно тебе или когда тебя придёт нагнуть твой менеджер, начальник или вообще заказчик (естественно, в самое неподходящее для тебя время).
Тебе всё равно придется это делать. Без вариантов.

Кроме этого, есть огромное количество способов организовать свою почту, чтоб жить было легче и при взгляде на почтовый ящик не бросало в дрожь, но я не буду конкурировать с гуглом: он лучше меня покажет миллион советов. Моя главная задача — объяснить, почему это важно.

Запоминай:

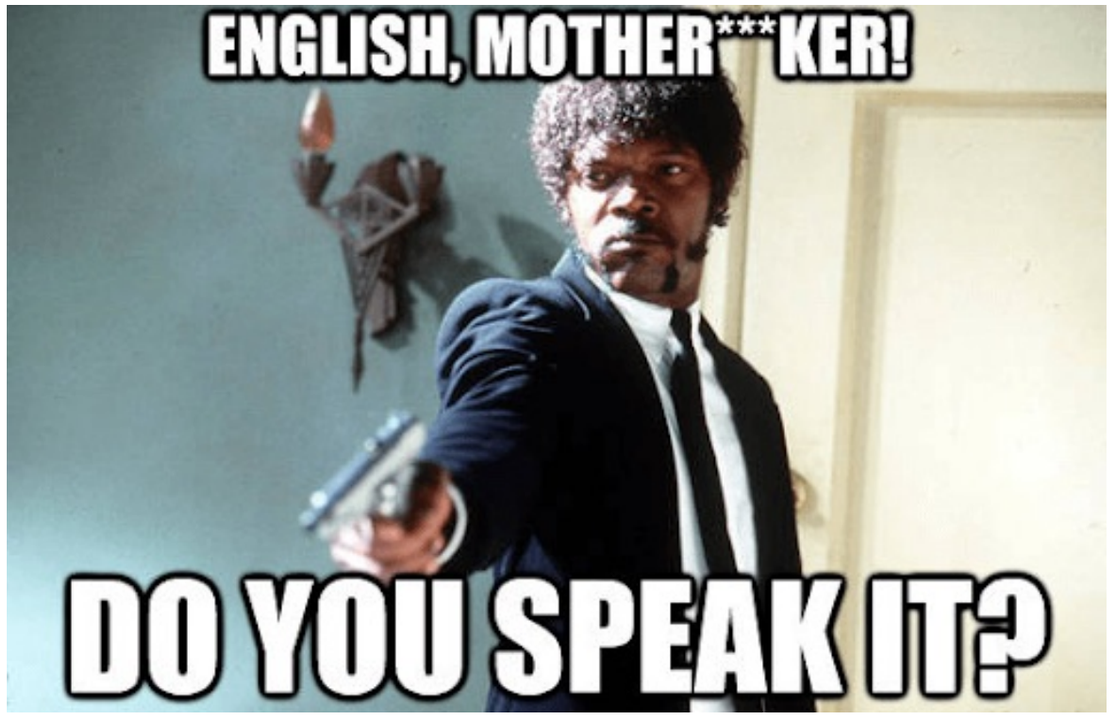
- Отвечай на письма в течение одного дня.
- Не используй технические термины в общении с людьми, которые их не понимают.
- Сокращай.
- Task tracking систем это всё тоже касается.

Незнание английского

Точнее, незнание и нежелание знать. В свое время для меня это стало полным откровением. Знание языка на уровне «лет ми спик фром май харт» считается вполне достаточным. Коверкание слов до неузнаваемости *(что за «функшен»? Это тебе оно понятно, а для меня оно «фанкшен»)*, тотальное игнорирование грамматики *(что значит “I do this”? Я не понял — ты сделал, делаешь или будешь делать?)* и, конечно, перевод в лоб, слово в слово — “How do you do? All right!” в лучшем виде. И не надо мне рассказывать что, мол, «всё равно поймут». Не поймут. В лучшем случае попросят повторить, в худшем — забьют хер.

Простой пример: ты мегакрутой программер, но ни бельмеса не понимаешь на инглиш мове. Тебе нужно общаться с заморским заказчиком (что нередко бывает). Естественно, ты не в состоянии этого делать, поэтому тебе в «помощники» берут девочку с инъяза или филфака, обзывают её “Project Manager” и она теперь общается с заказчиком.
(Ярые противники сексизма могут сюда подставить сыночка CEO или бездаря-кума босса, которого надо пристроить на работу. Но я оставлю «девочку» — таких жизненных примеров банально больше. Так вот...)

Английский она знает, но в технических вещах ни черта не соображает, за что ты её дико презираешь. А если б не её милое личико и третий размер, так и вовсе послал бы.



Но есть один момент. Эта девочка была бы не нужна, если б твоя ленивая жопа оторвалась от стула и походила на английский (можно даже вместе с мозгом). И ей бы (девочке, а не жопе) не надо было платить зарплату. Угадай, с каких денег ей платят эту зарплату? С тех, которые могли бы быть твоими.

Можно себя утешить лишь тем, что ты своей ленью создал новое рабочее место.

Это, конечно, круто, что ты можешь прочитать документацию по какому-нибудь API (кстати, «эй-пи-ай», а не «апи») и понять комментарии на stackoverflow, но там словарный запас, как у попугая. Даже мало-мальски сложные технические статьи тебе уже не под силу. А самые толковые из них, вот сюрприз, как раз на английском. Я могу ещё понять старперцов, которые в детстве учили английский по дубовым правилам, но это всё равно слабенькое оправдание — другие-то старперцы осилили и шпарят дай боже. Люди, начиная с 30 лет и моложе, имели и имеют все возможности нормально выучить язык чуть дальше «зе кэпитал оф грейт бритн».

К слову, когда в Лос-Анджелесе заказываешь себе «ипа», то на тебя смотрят как на идиота, а когда заказываешь «ай-пи-эй» — тебе действительно несут IPA.

И здравый смысл тоже никто не отменял. Незнание английского в XXI веке — это всё равно, что алфавит не до конца выучить.

Запоминай:

- Не жалей времени (и денег) на изучение английского.
- Относись к изучению языка так же серьезно, как к работе — потому что это и есть часть твоей работы.

Стеснение и лень переспросить

Слава богу, мы не индусы, которые растягивают лыбу во все 32 при видео-колле, не перечат, не возражают, не спрашивают и как манну небесную повторяют «Да, насяльника». Но у них это не от лени или стеснения, просто у них так принято — насяльника сказал — надо молча делать. Они себе карму на следующую жизнь зарабатывают. У нас вроде бы нет врожденной стеснительности, но на практике создается именно такое впечатление.

Возможно, это следствие предыдущего раздела про незнание языка, проще говоря, программисты просто устают болтать с заказчиком, соглашаются на всё, а про себя думают «Буду кодить — разберусь». Лишние 15-20 минут, потраченные на колл для выяснения деталей или на дискуссию о целесообразности задачи, а ещё лучше — в конце краткий рассказ о том, как ты собираешься делать эту фичу, — всё это экономит дни, недели, а то и месяцы времени (чем чёрт не шутит, может, даже годы).

Но ты же помнишь, «буду кодить — разберусь». А когда дело доходит до «разберусь» — начинаются вопросы. Много вопросов. И начинается переписка. Долгая, нудная тягомотина, потому что… программисты не любят писать письма.

Ещё хуже, если такой переписки нет. И программист просто делает так, как понял и как считает нужным. Очень часто он даже сам свято верит, что делает лучше, тщательно продумывая все детали той фичи, которую он сам себе нарисовал. И тем больше будет его разочарование и даже демотивация, когда ему скажут: «Всё хуйня — давай по-новой».

На выходе — просранное время, деньги и нервы.

Запоминай:

- Непонятно — спрашивай, сразу же.
- В конце колла проговаривай краткое summary.

Сферический идеальный код в вакууме

Программисты обожают писать идеальный код (по крайней мере, он им таким кажется), делать всё «по фен-шую», «по-правильному». Часто окапываются на своем рабочем месте, погружаются в задачу и не выходят из неё днями / неделями / месяцами. При этом иногда забывается смысл исходной задачи. Здесь есть, опять же, и проблемы коммуникации, но их мы прошли в предыдущих разделах. Теперь перейдем к техническим и околотехническим моментам.

При постановке задачи «Сделать автомобиль, чтобы ездить» — можно по-быстрому склепать запорожец за пару дней, а можно месяц пилить Бентли. В поставленной же задаче есть очень важное уточнение — чтобы ездить. И всё. Точка. Нам не нужна автоматическая коробка передач, климат-контроль и сидения с подогревом. Нам надо просто ехать. Если в условии задачи есть уточнение «Чтобы проехать 50 км из пункта А в пункт Б», можно даже уменьшить объём бензобака. А можно вообще предложить сделать велосипед. Программисты это умеют.

Если утрировать, есть три способа писать код: быстро «на коленке», суперкачественно и компромиссный вариант. Здесь проявляется весь опыт программиста.

«На коленке» подходит для прототипов, а точнее, тех случаев, когда этот код абсолютно точно никогда не будет поддерживаться или совершенствоваться. Это может быть и быстрая демка для потенциального клиента. Даже супер-мега-срочный багфикс на продакшене, ибо фикс на проде стерпит всё (если потом, конечно, его переделать качественно).

Суперкачественно (читай, идеально) — это долго, дорого и, возможно, совсем не нужно. При этом любой программист по своей профессии должен стремиться уметь делать именно так. (На всякий случай уточню: «стремиться уметь» != «делать».)

Компромисс — это сочетание двух крайностей.
Для его достижения Исполнитель (уже не программист) должен понимать бизнес-цель и иметь достаточный технический опыт, чтобы маневрировать от одной крайности написания кода к другой. Причем чётко осознавая, что именно он делает.

Как это ни странно, и бизнес, и программист любят всё переписывать с нуля. Только, к сожалению, на разных стадиях развития продукта. Бизнес — во время перехода от прототипа к более серьезному приложению. Программист наоборот — сильно позже. Особенно если до этого приложение писалось не им.

«У программиста — похожая проблема. По завершению работы он узнает, что написал прекрасную программу, соответствующую техническому заданию, но само ТЗ было несовершенно. Если бы он только знал цель этой программы, он, возможно, сделал бы её соответствующей этой цели даже при несовершенном ТЗ.»

Эдвардс Деминг, «Выход из кризиса»

Отсюда вытекают две «конфликтные» ситуации.

Первая: программист потратил слишком много времени (и, соответственно, денег заказчика) на простую вещь, которая задумывалась как, допустим, простая демка. Бизнес недоволен потраченным временем, программист расстроен, что его старания не оценили.

Вторая: бизнес решился на переделку большой части приложения с прицелом на будущее (другой фреймворк, технология и т.д.), а программист усиленно пилил и «импрувил» свои костыли для текущей версии. Опять же, его работа идёт по звезде, как и время.

Простой пример: есть фича. Эстимейт на то, чтобы сделать её на старых костылях, — один месяц. А переписать всё с нуля плюс добавить эту фичу — полтора месяца. Причём из этих полутора месяцев сама фича займёт лишь неделю. И все последующие сопоставимые по трудозатратам фичи (на свежепереписанном коде) займут тоже одну неделю, а не месяц, как на старом говнокоде. Вот это простое умозаключение оборачивается в красивые слова и отправляется продактам, биз-девам, CEO и прочим людям, которые принимают бизнес-решения. Дай им эту опцию и, возможно, они с радостью её примут, — к примеру, у бизнеса большие планы по дальнейшему развитию этой фичи и на ней будут зарабатывать большие деньги. А может, оно нафиг не надо — мы допилили этот кусок и не вернемся к нему ещё год, а может, и никогда не вернемся. Тогда по-быстрому допиливаем костыли и откладываем в дальний ящик.

Вот только проблемка: бизнес может не узнать, что у него есть несколько вариантов. Потому что программист сидит и жуёт сопли, бубня себе под нос о том, какой тут страшный legacy и как всё плохо и жизнь — боль. И даже если он соберёт силу воли в кулак и решится заявить об этом, то всё ограничится фразами типа «Нам нужно всё переписать, потому что это старое говно». Естественно, бизнес пошлёт его нафиг, потому что из этой фразы нифига не ясно, отчего же наступит великая радость и мир во всём мире, когда старое говно перепишется на новый, блестящий, идеальный и просто мега-здатый код.

Чтобы избежать ситуаций, описанных выше, нужно, во-первых, знать, как будет использоваться результат твоей работы (коммуникация), и, исходя из этого, как нужно писать код (технический скилл). Причем только в таком порядке.

Запоминай:

- Узнавай заранее, как будет использоваться результат твоей работы.
- Адаптируй скорость и качество под задачу — для прототипа своё, для enterprise своё.
- Дай бизнесу разные варианты и опиши их плюсы и минусы. Бизнесу, в первую очередь, важно время — описывай с этой точки зрения.

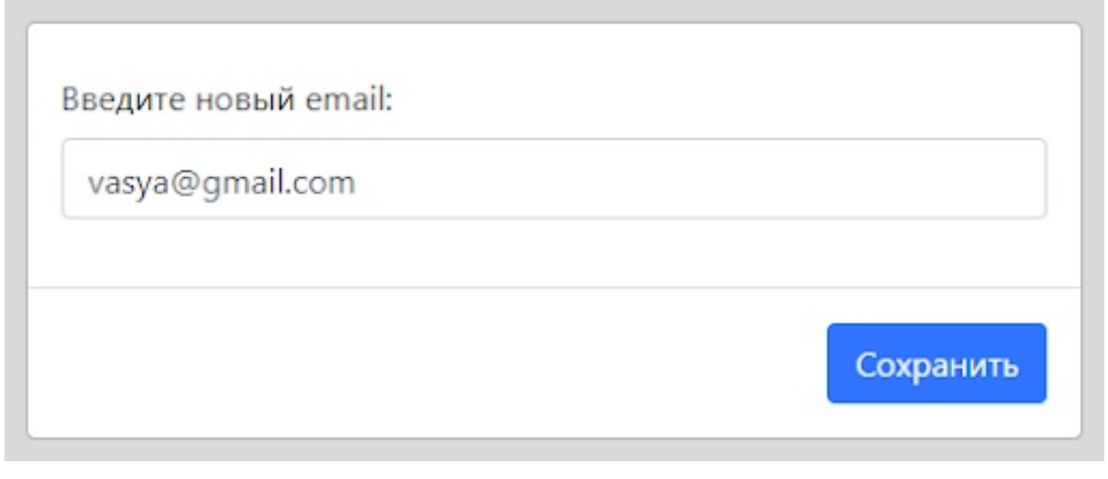
Программисты делают не то, что нужно

По сути это резюме предыдущих глав. Программисты настолько сосредоточены на чисто технических моментах и забивают на все остальные, что на выходе часто получается полная хрень.

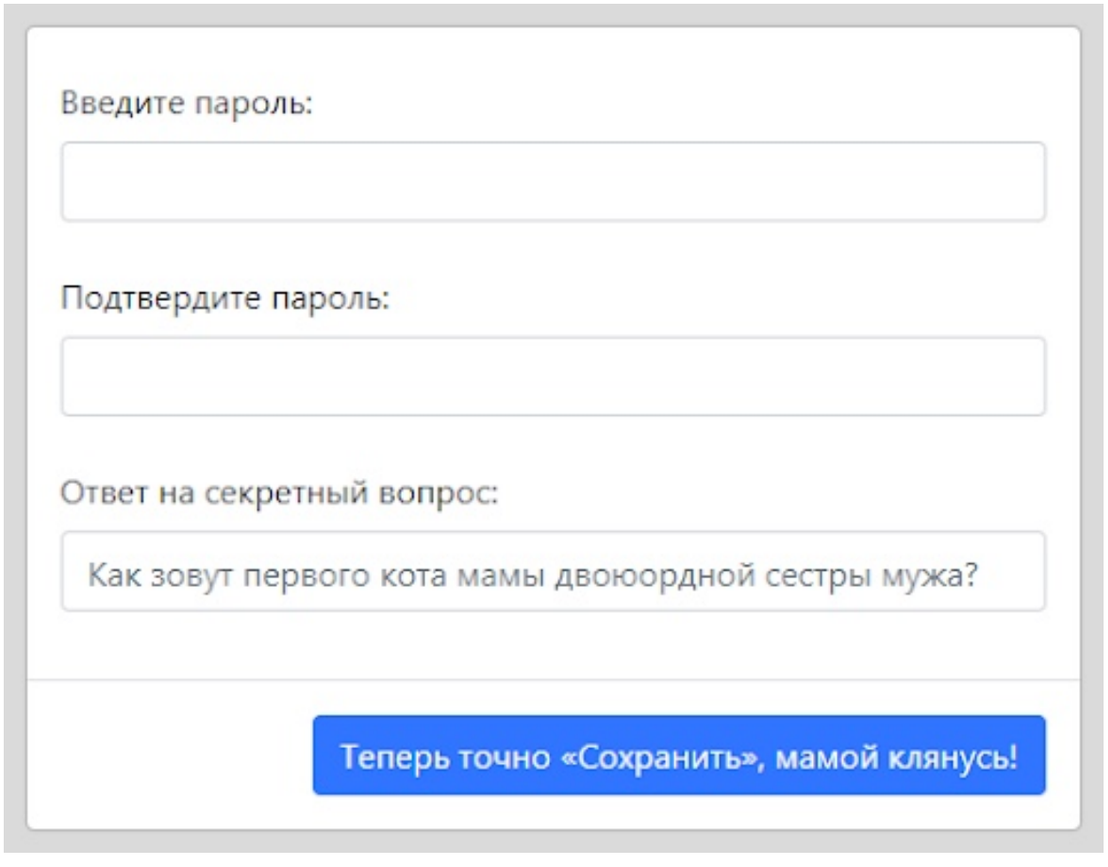
Выражу такую «глубокую» мысль:
Баг со стороны *программиста* — если система работает не так, как *запрограммирована*.
Баг со стороны *бизнеса* — если система работает не так, как *должна работать*.

То есть, со стороны бизнеса, это может быть и баг в коде, и ошибка в реализации, и недочет ещё на стадии формирования требований, и просто кто-то кого-то не понял. И бизнесу абсолютно пофиг, на какой стадии проблема.

Покажу на пальцах: если ты нажал на кнопку «Сохранить»…



...и оно не «сохранило», а предложило заполнить кучу абсолютно ненужных полей в форме — ты матернёшься: тебя об этом никто не предупредил и по тексту «сохранить» абсолютно не понятно, что дальше надо будет выполнить ещё надцать действий.



И тебе пофиг: так запрограммировали, у продукта нет толкового UX человека или просто у них все разработчики идиоты. (Ещё уродской нечитаемой капчи не хватает для полного фарша.)

Да, первый раз это будет не так раздражительно, но если надо повторить процедуру раз десять… А сколько подобных формочек вышло из под твоих рук?

Программисты зачастую не знают, что они на самом деле делают. И зачем. Плохо и то, что часто их это вполне устраивает. В случае тупого аутсорса задачи могут проходить несколько рук до того, как попадут в разработку и тогда найти, откуда ноги растут у той или иной фичи, бывает проблематично. Тут огромное преимущество у продуктовых компаний. Но даже в аутсорсе можно докопаться до сути. Просто нужно приложить больше усилий – хотя бы банально попытаться спросить. Но так как «мне за это не платят», никто не парится.

И получается полная хрень.

Запоминай:

- «Поворачивай мозг» в другую сторону: учись смотреть на результат своей работы со стороны тех, кто этим будет пользоваться.
- Заметил опечатку на второй картинке? А то, что вопрос сформулирован некорректно? (Тут ничего не запоминай — просто тестируй)

Эстимейты

А теперь представь себе картину. Ты выставил эстимейт, а потом в него не вложился. Завалил сроки. Ну ладно, один раз не гондурас. Второй. Третий... Так ты, оказывается, так себе специалист. Ты великодушно можешь выйти в оплачиваемый овертайм (по двойному тарифу, естественно) и доделать свою работу? Ахуел?!

Вместо вовремя сданной работы за N денег, на выходе имеем наскоро слепленную хрень с костылями и велосипедами, с опозданием, да ещё и за бóльшие деньги?! Ты пообещал, ты проебал, а теперь ты ещё и не виноват.

Обычно встречается отмазка про изменения требований на лету. Мол, сначала был один объём работ, а по ходу дел всё поменялось. Тут есть несколько моментов.

Во-первых, не всегда эти изменения в требованиях так сильно влияют на финальные сроки, как это рисуют программисты.

Во-вторых, некоторые изменения вызваны самими программистами: если есть задача сделать кнопку и ты сделал её красно-серо-буро-малиновой, а клиент это не принял, — это не изменение в требованиях, это твой проёб. Никто никогда не будет прописывать требования до последней запятой — это всё равно, что взрослому человеку объяснять, что когда посрал, надо подтереть задницу.

В-третьих, если требования действительно сильно изменены клиентом, то в момент, когда новый скоуп сформулирован, — программист должен поднять красный флаг, что сроки меняются. Сразу, без промедления, на берегу. Даже если скоуп не менялся, но ты видишь, что не влазишь по эстимейту, — говорить об этом нужно сразу же, как ты это поймёшь. Дай людям, принимающим бизнес решения, запас времени, чтобы среагировать на ситуацию: поменять приоритеты, выкинуть фичу из спринта, урезать эту фичу до MVP, да пускай даже просто сказать: «Всё ок, ничего страшного», — но это решение должно быть их, а не твоим, поскольку оно не в твоей компетенции. Когда инженер делает проверку космического корабля накануне

старта и находит, условно, плохо закрученный болт, — он докладывает об этом немедленно, а не в момент зажигания.

Суть эстимейтов я для себя ощутил, когда первый раз потерял на этом собственные деньги. У меня был небольшой проект, который меня попросил сделать мой старый знакомый. Ничего военного, но времени у меня не было вообще, поэтому я оценил проект и передал его своему другому знакомому программисту, при этом объявив (свой) изначальный эстимейт, — это была ошибка номер раз: оценивать должен конечный исполнитель. Проект был совсем небольшой — изначальный эстимейт был в районе двух дней работы. Через неделю с лишним (!) проект был сдан. Он был сделан идеально — тут не поспоришь. Мой знакомый (заказчик) счастлив. Вот только эстимейт был превышен в четыре раза.

Разница между эстимейтом и реальной работой была в районе $600. Я не проконтролировал работу и просто поглядывал в общий скайп-чат с целью убедиться, что заказчик и программист понимают друг друга. Я мог слёзно попросить оплатить всё потраченное время — заказчик мой хороший знакомый, платит не из своего кармана, работа сделана суперски — почему бы и нет. Но в тот момент, когда меня спросили о финальной сумме, я решил, что лучше я сейчас заплачу из своего кармана эти $600 с копейками и запомню свой проёб на всю жизнь, чем когда-нибудь сделаю такую же хрень и попаду на $60 000. Я сказал, что не имею права просить больше, чем заявил изначально. Программисту заплатил полную сумму.

Это был не только первый, но и последний раз в таком роде. Надеюсь, действительно последний.

К черту воспоминания, возвращаемся к жизни. Очень многие пренебрегают стадией планирования. Если работа ведется по скраму или его жалкой пародии, то (обычно) есть даже выделенный день на планирование. Когда нужно не кодить, а просто сидеть, трындеть и оценивать фичи в днях / часах / убитых енотах. Этот день расценивается чуть ли не как выходной.

Хер там. Планирование — это самая тяжелая часть работы. За 15-30 минут тебе нужно в голове написать всю фичу целиком. Сделать архитектуру; предусмотреть подводные камни; учесть возможные доделки (хотя бы частично); заложить свою корявость, то бишь QA найдет баги в твоём святом коде и т.д. Ты даже можешь попробовать заложить возможные изменения в бизнес-требованиях, описанные ранее (но это высший пилотаж и «мне за это не платят»).

И после этого, выданный тобой эстимейт передается на сторону тех, кто делает деньги. Этот эстимейт переводится в доллары, ожидаемое время, когда фича будет готова, и эту фичу уже продают клиентам с формулировкой «будет готово через месяц». Да, может, это для тебя новость, но фичи часто продают заранее. Потому что, чувак, конкуренция. И если ты во время планирования ковырял в носу, есть большой риск пустить по звезде целую цепочку планов. Как результат, потеря денег. Из которых, вуаля, тебе платят зарплату. Может, из неё вычесть?!

В конце концов, ко всему вышенаписанному, в нагрузку, добавляется один очень важный факт: 1 день != 8 часов. «Чистых часов для программирования». Когда ты берёшь две задачи по 4 часа — ты не сделаешь их за один день — никогда. В твоём рабочем дне максимум 4 часа. Остальное время ты куришь, отвлекаешься на ответы, митинги, пьёшь кофе или мучаешься от похмелья. При оценке задач в часах это не учитывается — наэстимейтили, сложили, поделили на 8, получили количество дней — алаверды. Когда оценка идет в днях (с делением не крупнее, чем 0.5 дня), то, во-первых, нивелируется переоценка собственных сил и, во-вторых, тебе чисто физически проще оценить результат работы одного дня, чем результат работы отдельно взятых 3-4-5 часов и этот результат лучше подходит для сравнения трудозатрат. Ты приблизительно понимаешь, что успеваешь сделать за день — это чётко ограниченные рамки: пришёл на работу, отработал, ушёл — легче зафиксировать результат. Многие ж говорили эту фразу под вечер: «Вроде целый день что-то делал, а нифига не успел».

Под конец главы приведу ещё один хороший пример про эстимейты. В своей предыдущей компании я затеял ремонт крыши, ибо здание старенькое и крыша начала протекать. Как полагается, вызвали строителей, нужно было для начала раздолбать плитку и снять верхний слой. Строители (а это такие же бесстыжие люди, как и программисты) поклялись за день с перфоратором всё раздолбать и дальше в тишине спокойно класть крышу заново. Естественно, у них что-то там не получилось на выходных и они вышли в понедельник. Буду честен, я не думал, что это будет та-а-ак громко, поэтому перед вторым (!) днем имени перфоратора всем сотрудникам было сказано, что они могут не приезжать в офис и поработать из дому. Половина людей этим правом не воспользовалась.

На четвертый день (!!), делая кофе на кухне, я услышал весьма логичные жалобы на долбаный перфоратор (что уж там, я и сам про себя матерился как сапожник). На что я заметил: «Теперь вы отлично понимаете, что бывает, когда вы проёбываете свои эстимейты». К счастью, четырёх дней хватило и перфоратор утих.

Эстимейты строителей, как и программистов, нужно умножать минимум на два. Или учить их правильному планированию… и умножать на полтора.

Запоминай:

- Не спи на планировании — это критично важная часть работы.
- Понимаешь, что не влазишь в сроки — сообщай сразу (менеджеру, заказчику).
- Помни: 1 день != 8 часов, максимум 4.

Программисты и QA

Программисты не любят тестировать свой код. И я, как почти бывший программист, могу это понять. Даже не понять, а ощутить. Это жутко неприятно — искать говно в своем же «детище». В идеале программисты должны сами тестировать свой код «до блеска». Но даже когда ты программист на своем же продукте, даже когда ты подготовил детальный test-suite, когда ты кровно заинтересован — тебе очень тяжело заставить себя это всё непредвзято проверить. Но... Надо. Для своего же блага. Все знают, как неприятно возвращаться к багам из фичи, которую ты закончил день / неделю назад, когда ты уже полностью погрузился в следующую задачу. Прерывания для программиста — это жуткий ад. Но в прерываниях этого типа программист виноват сам.

Теперь к тестировщикам (QA). Как следствие предыдущего абзаца, QA вынуждены тратить большу́ю (а может и бо́льшую) часть времени на тестирование механики фичи, а не соответствия фичи бизнес-требованиям. Особенно при отсутствии автоматизированного тестирования. В идеальном мире (а у нас, блин, какой?), QA тестируют именно бизнес-требования, а не механику сделанной фичи.

Я, конечно, понимаю, что джунам-тестерам тоже нужна работа и проще, чем тыкать мышкой по заданному сценарию, нет ничего. Но пускай уж лучше они сразу учатся писать автотесты. Monkey-тестирование оправдано лишь на сверхмалых проектах, где время на вход в автоматизацию превышает критическую отметку. Что из себя представляет эта отметка — это тема для отдельной статьи.

Господи [или просто кто-то поумней, чем я], придумай универсальный способ, как убедить весь бизнес сразу, что автоматизация нужна (почти) всегда! И объясни мне этот когнитивный диссонанс: почему бизнес этого не вдупляет до сих пор? Я пока так и не смог разгадать этот ребус.

Я немного отошел от темы. Так вот, программист должен тестировать свой код на всё, что только придет ему в голову. А если точней, на всё, что может прийти в голову QA. Цепочка «сделал — протестил — пофиксил — протестил — сдал QA» куда короче чем «сделал — глянул, вроде всё ок — сдал QA — QA добрался до этой задачи — нашёл баг — записал в JIRA — программист добрался до JIRA — прочитал, вник — пофиксил — сдал QA» (повторить два-три раза по вкусу).

Я не делаю здесь скидку на QA, которые находят абы какие вещи и называют их багами, лишь бы что-нибудь найти. Потому что с другой стороны найдутся программисты, которые делают фичи абы сделать как-нибудь, лишь бы сдать — на них тоже не делаю скидку.

Вовремя потраченные 5-10 минут на тестирование в конечном итоге экономят куда больше времени.

Когда ты тестируешь свой код и вроде бы всё хорошо, а потом в конце вылазит какая-то мелкая гадость — вот прям совсем мелкая, как тебе кажется — лезешь разбираться и понимаешь, что для того, чтоб это пофиксить, надо или половину переписать или, не дай бог, нарушить гармонию твоего идеального кода.
И ты такой: «Да ну нахуй, не заметят». Нифига, чувак, — заметят.
Может, через неделю, может, через месяц, но таки заметят.
И всё равно придётся фиксить. И не надо потом рассказывать, что это сложно, невозможно и вообще это фича — будешь фиксить. Если тебя, конечно, раньше не уволят.

Запоминай:

- Тестируй свой код сразу и тщательно — иначе потом всё равно придется его фиксить.
- Не ленись тратить лишние 10 минут на тестирование.

Часть 2

Ягодки

Неумение + нежелание принимать решения

Иногда мне кажется, что это свойство нашего менталитета. Но у меня мало опыта работы внутри команды за границей, поэтому не буду утверждать категорично.

Итак, нужен дядя, который всё за тебя решит, разжуёт и расскажет, а тебе останется только написать код. Так вот, написание кода — это последний шаг в выполнении задачи. И как бы странно это ни звучало, далеко не самый важный шаг. Это процентов 10, максимум 20 от всей работы. Очень редки случаи, когда больше.

Часто задача не ясна до конца. Даже если отбросить проблемы с коммуникацией, описанные в предыдущих главах, программисту периодически (на самом деле, почти всегда) приходится принимать решение — как *именно* эта задача будет реализована. И здесь начинаются проблемы.

В этом случае, знание того, чем абстрактный класс отличается от интерфейса, абсолютно бесполезно. Люди принимают решения, основываясь на своих симпатиях (правда, это касается любой должности и даже любой области). React vs Angular, Ruby vs PHP и так далее. Программист не понимает последствий своих решений. Инженер и Исполнитель должен понимать.

Обратная сторона медали: принятие решения без необходимой компетенции. Грубо говоря, если у тебя есть срочная фича, которая должна быть через неделю в релизе, а ты в процессе задумал мега-рефакторинг и QA должен перепроверить половину приложения — это крайне хреновое решение. Для таких ситуаций есть тимлид, менеджер и здравый смысл. Но этим всем нужно ещё уметь воспользоваться.

Переходим к более высокопоставленным людям: лидам, менеджерам, начальникам отделов и т.п. Для того чтобы принимать непопулярные, жёсткие, но необходимые решения, — нужны яйца. По моим наблюдениям, таких людей от силы 2%. А скорее всего, намного меньше.

Начальник отдела может быть абсолютно аморфным созданием, а зеленый джун может ответить за свои слова и взять ответственность на себя. Это не зависит от позиции. Это больше зависит от воспитания. Аморфное создание легко распознается по ряду признаков:

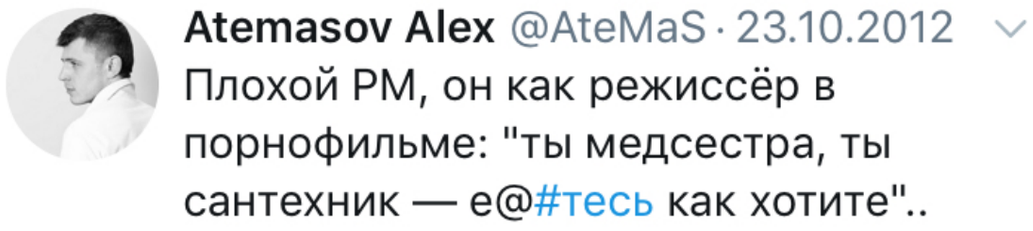
1. Делегирование или перенаправление элементарных вопросов. Грубо говоря, когда быстрее и проще ответить на явно поставленный вопрос / сделать определенное действие, чем вовлекать других людей. Да, иногда это делается в целях

выработки рефлекса. Скажем, начальник нанял заместителя, который должен забрать на себя часть мелких вопросов и освободить время самому начальнику. Но это явление временное и не может длиться месяцами.

2. Расплывчатые и нечёткие ответы на вопросы коллег и подчиненных. Или полный уход от ответов. Что-то в духе «Да что там думать — там всё элементарно». Оно-то может быть и элементарно для того, кому задают вопрос (хотя и то не факт). Но абсолютно не элементарно для того, кто задает вопрос. Хороший начальник не будет ни отфутболивать человека, ни решать за него. Он даст подсказку.

3. Игнорирование явных проблем. Сюда можно вставить картинку с собакой в горящем доме. Когда всем вокруг понятно наличие проблемы, но тот, кто должен её решать, просто засовывает голову в песок и с невозмутимым видом говорит: «А что? Всё ж хорошо!»

4. Желание всем понравиться. Нельзя всем понравиться. Поэтому и стремиться к этому бесполезно. Нужно, блядь, делать свою работу.



Иногда человек хочет стать лидом или менеджером просто ради лычки, или большой зарплаты, или и того и другого. Обычный Senior Developer — это абсолютно не Lead. От лида нужен категорически другой набор навыков. Любой, у кого в подчинении есть хотя бы ещё один человек, уже должен обладать элементарными менеджерскими навыками. Если ты отличный программист, но никакой менеджер — развивайся по технической части дальше, стань Solution Architect, выучи ещё одну технологию, язык программирования. Не лезь «лидить» и «менеджерить». Некомпетентные люди на руководящих должностях — это куда бóльшая беда, чем те же люди, но без подчинённых. Херовые руководители распространяют свою херню дальше — подчинённые, видя некомпетентность вышестоящего, снижают к себе требования и попросту начинают халтурить, а потом эта зараза перекидывается на другие команды. Вот таким нехитрым образом один незаслуженно поставленный Lead может «попортить» с дюжину программистов.

Вернёмся к главному навыку. Навык «иметь яйца» ценнее всех — люди, обладающие им, способны закрыть собой целый пласт вопросов и проблем (сейчас или в перспективе). Эти же люди в зоне риска — они могут решать проблемы в любой другой компании, а так как их мало, — они нужны везде. Везде, где умеют распознавать такой навык. Они даже могут сделать свою собственную компанию. Потому что могут всё сделать сами и/или найти людей с недостающими навыками. Но этот риск оправдан — без таких людей бизнес умрёт.

Опять же, «есть один нюанс». Принятие решения при отсутствии должной компетентности может нанести большой ущерб. Иногда нужно иметь смелость сказать, что у тебя нет экспертизы, а не бить себя в грудь и лезть на амбразуру. И для этого тоже нужно иметь яйца. Но если выбора нет, то лучше хреновое решение сегодня, чем хорошее завтра или вообще никакого.

Компетентность «нарастить» проще, чем отрастить яйца. Поэтому если лиды/менеджеры не могут / не умеют / не хотят принимать решения — их не научить — гнать их нахер ссаной тряпкой, какой бы опыт у них за плечами ни был. Хладнокровно и без зазрения совести. Можно ещё раз пересмотреть «Манибол» для уверенности.

Запоминай:

- Если у тебя нет менеджерских навыков и особого желания им учиться — не лезь на любые должности, где у тебя будут подчинённые.
- Если ты таки решил кем-то поуправлять и поначальствовать — прочитай сначала «Чёрную книгу менеджера» и подумай ещё раз.

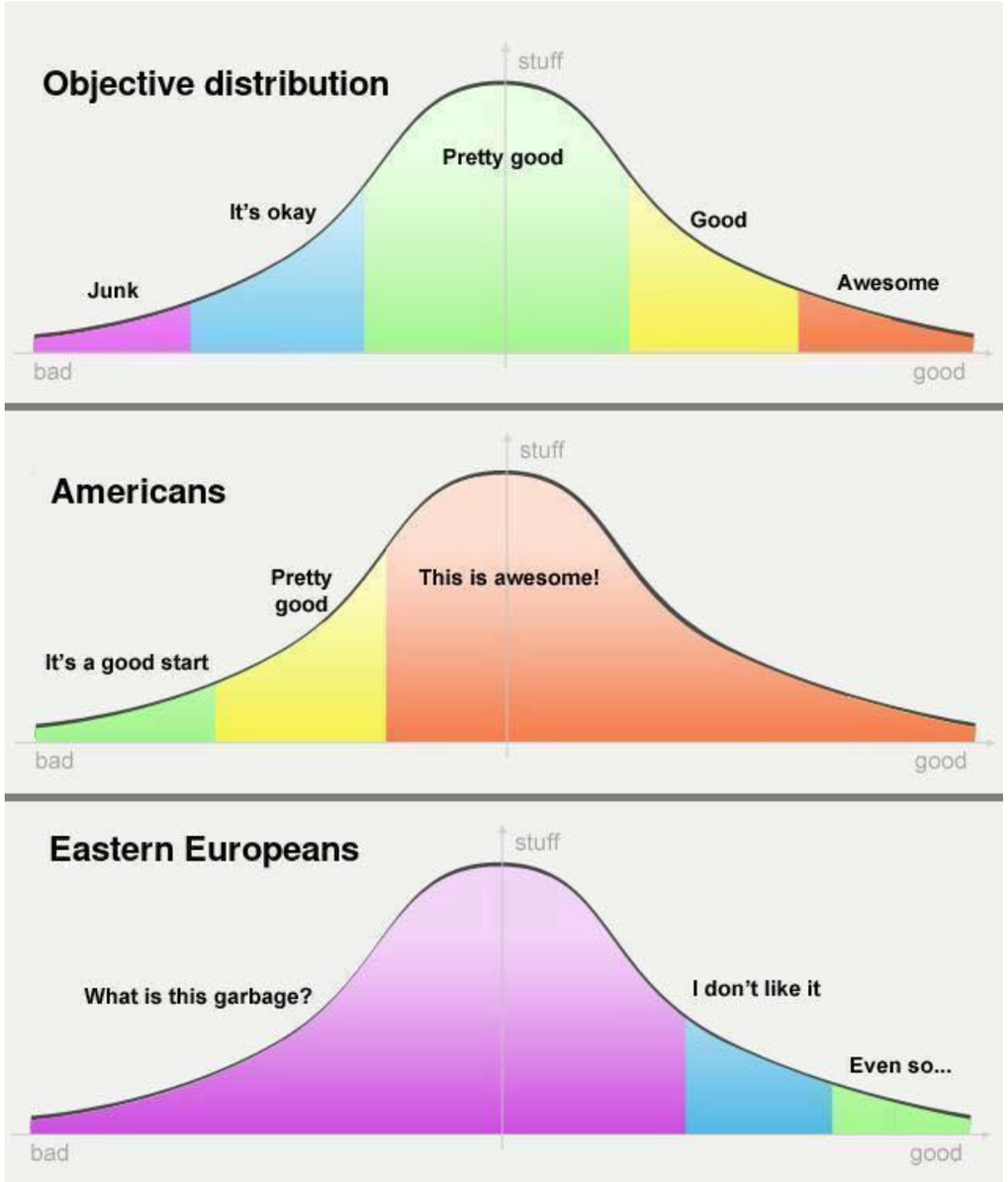
Разница в менталитете

Тут я буду говорить, в основном, про US vs UA. Просто ввиду того, что здесь у меня есть релевантный опыт.

Я уже давно научился читать письма от своих американских коллег пропуская слова “great job” и “amazing”. Самое главное в письме находится после слов “but” или “however”. Представь себе упрощённое письмо: “We’ve done a great job. But we are fucked”. (Для тех, кто так и не доучил английский до passive voice — это не мы, а нас.) Почти любое письмо о факапе от человека из US будет выглядеть подобным образом, только без слова fuck. Причем большая часть текста будет посвящена “great job” и только в самом конце немного о том, что на самом деле “we are fucked”. Что, думаете, на самом деле правда?

Наш человек (тем более программист) входит в когнитивный диссонанс при виде такого письма. Как можно сделать “amazing” и при этом налажать?! Как следствие, автоматически выбирается путь наименьшего сопротивления — “We’ve done a great job” — и не нужно делать никаких телодвижений.

Здесь закладывается самый главный корень «мискоммуникации». Одни и те же слова и письма интерпретируются по-разному, и начинается расхождение в ожиданиях. А следом идут обиды и тыкание пальцем друг в друга при факапах.



Можно, конечно, попробовать научить своих американских коллег выражаться более реалистично, но проще научиться понимать, что они на самом деле имеют в виду. Я думаю, многие видели картинку на предыдущей странице, но почему-то мало кто воспринимает её на свой счёт. Неприятно, конечно, пропускать похвалы в свой адрес и выискивать косяки в письмах — так уж работает наш мозг — но это во имя своего же блага. Оптимизм — это хорошо, а трезвый расчет — лучше.

Кстати, американцам абсолютный неуд за такую манеру говорить. Понятны их подспудные мотивы и общая культура, но так они делают хуже сами себе. Будут возможности — буду им тыкать в это. К сожалению, меня на них всех не хватит. Они, правда, меня за такую — противоположную — манеру общения считают немного «ку-ку», но я как-нибудь переживу.

Запоминай:

- Учись правильно читать письма от западных коллег — у них почти всегда больше комплиментов, чем надо, и меньше негатива, чем стоило бы.

Лояльность

Лояльность к компании, в которой ты работаешь. Хотя бы временная. Только настоящая, а не на словах. Толкание речей на корпоративах и вылизывание жопы начальству — это, конечно, некоторым нравится, но все начальники знают цену подобному лизоблюдству. Даже если сами таким грешат (я бы сказал, тем более, если таким грешат).

Ты зачем-то пришел работать в эту компанию. За деньгами, скиллами, приятным коллективом или тебя просто больше никуда не брали. Это твои подробности и, кроме тебя, они никого не волнуют. У тебя есть работодатель, который платит тебе зарплату и хотя бы этим он уже тебе ничего не должен. На самом деле, тебе вообще никто ничего не должен. По умолчанию. Перестань делать вид, что ты одним присутствием здесь делаешь всем большое одолжение. Знаешь картинку (справа)? Так это не шутка.

Сделай одну простую вещь. Замени свой стандартный вопрос, который сидит в голове у большинства программистов, «а что компания делает, чтобы мне было хорошо?» на вопрос «а что мне сделать, чтобы компании было хорошо?». Поставь себя на место тимлида, менеджера, босса компании и подумай, чего хотят эти люди. Любой человек на любой позиции может найти минимум два-три ответа на такой вопрос. Всегда. Это и есть лояльность.

Выставиться тимлиду пивом с фразой «ты меня классно научил делать вот это и вот это» — тоже правильный ответ на такой вопрос. Мелочь, а приятно. Только сделай это искренне, а если иначе, то только испортишь впечатление.

(картинка © Ильи Горбова)

Все мы знаем, что ты пришел сюда работать не до конца своей жизни. Год-два, может, три. Реже дольше. Об этом знают все и везде, даже в компаниях, где на каждом углу раздают корпоративные футболки и кричат о единстве и сплочённости коллектива (а слышится «аллилуйя»). Именно поэтому и раздают — чтобы купить кусочек твоей лояльности, и кричат — чтобы ты увидел, что тут все такие типа пиздец-как-лояльные. Потому что в среднем по больнице это неправда. Поэтому ты сейчас читаешь эту главу.

Кстати, банальное выполнение своей работы качественно и вовремя, без лишнего нытья и разведения вони по любому поводу, когда ты не устраиваешь начальству лишнего геморроя на ровном месте — это наибольшее проявление лояльности. (Сейчас мои бывшие начальники читают это и ехидно хихикают.) Да, за это по головке не гладят, но уважают точно.

Как резюме главы, я вверну очень субъективную и абсолютно недоказуемую мысль. Но коль уж ты дочитал до этого места, — прочитай внимательно и вдумчиво.

Есть такая штука — карма. Да-да, именно она. Я не считаю карму чем-то мифическим и потусторонним: я считаю, что это очень сложное социальное явление, для описания которого ещё не придумали правильного инструмента. Кто знает, с какими людьми ты пересечешься через 3-5-10 лет, что эти люди будут о тебе знать и кого из твоих бывших коллег будут о тебе спрашивать.

Фишка в том, что в положительной карме убедиться можно, а в отрицательной — почти никогда — вряд ли ты узнаешь, какая перспектива прошла мимо тебя.

Короче, не будь говном.

Запоминай:

- Тебе никто ничего не должен. Что заслужил — то и имеешь. Это вообще единственная непоколебимая истина, которую я знаю.
- Задавай себе правильный вопрос: «что *мне* сделать, чтобы было лучше?»
- Не ной!

Деньги

Откуда берутся деньги? Вариантов несколько, но всегда они берутся из чьего-то кармана. Либо это карман создателей продукта, либо инвесторов, либо кастомеров (клиентов). Почти все всегда стремятся к последнему (кроме тех, кто просто пилит деньги инвесторов). Нужно постоянно помнить о том, что когда ты раз в месяц получаешь свои N тысяч долларов — они всегда вытащены из чьего-то кармана. Это очевидная вещь. Ну так вся книжка про очевидные вещи. На которые и о которых, тем не менее, почему-то заб*и*вают и заб*ы*вают. Программист часто настолько далёк от монетизации и бухгалтерии продукта, что редко задумывается, а с каких шишей ему платят зарплату.

За что же на самом деле платят программисту?
За реализованные фичи, которые должны принести доход заказчику (просто, да?). Не за идеальный код, не за ультрамодные технологии, не за умничанье на митингах. Чем больше эти фичи приносят владельцам / инвесторам продукта, тем больше они могут заплатить программисту.

Ах да.. Мы же в аутсорсе. В основной своей «массе». В случае аутсорса логика всё равно примерно та же. Потому что продают скиллы людей, хоть и делают упор в основном на чисто технические скиллы (об этом будет в следующих главах).

Представь, что тебе нужно пойти и продать то, что ты только что накодил. Попробуй напродавать хотя бы на свою месячную зарплату.

Что я всей этой лирикой пытаюсь объяснить? Деньги не берутся из воздуха. Твоя зарплата или почасовой рейт существуют не в вакууме, а привязаны к реальным показателям успешности и прибыльности продукта / компании. Твой код, написанный под NDA, на работе, за деньги ничем кардинально не отличается от кода, написанного бесплатно для open source проекта. Ничем, кроме денег, которые этот код приносит.

А теперь перейдем лично к тебе, друг мой.

Если тебя интересуют деньги и только деньги, то ты уже в очереди на увольнение. Ты уже никому не нравишься, и когда ты налажаешь, тебя уволят. Если ты пришел за повышением зарплаты с формулировкой «мне тут за углом предложили», — ты продвигаешься в этой очереди на первые места. Ты, конечно, можешь быть супер-мега важен на этом проекте / продукте, но в голове у своего начальника ты уже с вещами возле двери с табличкой «выход».

За просиживание жопы на стуле денег не платят, повышение за выслугу лет — архаизм. Performance review придумали не для того, чтобы раз в год повышать тебе зарплату, а чтобы держать её на актуальном уровне и у тебя не было морального права попросить больше, чем ты сто́ишь. Попросить ты, конечно, можешь. Но аргументов, кроме описанного выше «мне тут за углом предложили», у тебя не будет.

Если у тебя в компании нет performance review и через год работы тебе не повысили зарплату — значит ты этого не заслуживаешь. Съешь это и успокойся. Сядь, проанализируй свою работу непредвзято, а лучше привлеки в помощь кого-то со стороны. Расценивай свою работу как две строчки в финансовой отчетности: сколько тебе платят и сколько на тебе зарабатывают. Да, у тебя нет полной информации для второго пункта, но ты можешь пойти и узнать. Поднять жопу и оценить, чем ты важен и важен ли вообще. Не можешь откопать такую информацию — пойди и спроси напрямую.

Если ты в аутсорсе и твою голову просто перепродают — спроси, как её можно продать дороже. «А я ещё и на машинке умею» — давайте и это продадим.

Пойми, друг мой дорогой, тебе дали больше «за углом налево» только потому, что там клиент жирный заходит (или расширяется) и они могут тебя продать ему ещё дороже. Или заткнуть тобой дырку, чтоб проект пошёл, и зарабатывать на остальных. Аутсорс компании считают по марже «купил-продал» (барыги, по-другому не скажешь). Это никак не связано с твоим реальным уровнем.

Это настолько важно, что я напишу ещё раз так:

Это никак, блядь, не связано с твоим реальным уровнем!!!

Или ты действительно считаешь, что при умении 2+2 на JS, и то только на Angular, твоя честная зарплата реально $3K? (И это мы ещё верстать не умеем.) Зарплата middle javascript developer — самого убогого языка программирования на данный момент (это пишет человек, у которого JavaScript это main skill).

Да, я знаю, рынок и всё такое. Раз дают, значит я столько стою. И я даже соглашусь. Людей, умеющих в javascript, — куда меньше, чем людей, умеющих в PHP или Java, просто ввиду относительно молодой технологии, плюс та нагрузка, которую на эту технологию возложили, увеличивает эффект. Поэтому спрос превышает предложение и дальше по тексту. Ну так скоро все научатся делать JS (что мы и наблюдаем). А ещё китайцы.

Я привожу пример javascript'еров, поскольку он наиболее показателен и я с ним больше всего сталкивался. К примеру, люди, которые приходят на вакансию PHP, просят меньше и обладают куда более фундаментальными знаниями, чем непонятные личности, откликающиеся на вакансию UI Developer (почему-то считая, что UI Developer — это то же самое, что JavaScript Developer) и полтора часа в тему и не в тему на любой вопрос пихающие мне заготовки из Angular'а.

(После этого абзаца все желающие могут покидаться в меня говном по поводу PHP, но факт остается фактом.)

Хочется крикнуть прокуренным полустарпёрским голосом «Да я! Да за $1.5K! Да я уже и HTML/CSS, и JavaScript, и PHP, и автоматизированное тестирование делал, что unit, что acceptance, и вообще уже был начальник отдела» (не сильно громко?). Просто тут уже вылезут совсем старпёрские и заслуженные лица со словами «Да мы и за $300! Да ещё больше делали!»

Под конец этого раздела поверну мысль в такую сторону. Чем *больше* ты «человек-оркестр», тем *больше* ты стоишь. Да, тебя не запихнёшь на месячный проект с Joomla / Drupal, потому что там такие монстры не нужны. Ну и нафиг оно тебе надо?! Такие вещи хороши для стажёров, джунов и тех, кто просто… ну не смог быть полноценным программистом. Тоже не грех, в принципе.

Есть примерная цепочка от-и-до полного стека разработки продукта: DevOps — Database — BackEnd — UI (JavaScript) — UI (HTML/CSS) — UI (Design) — Management — Product — Sales и где-то по дороге надо вставить QA, только не знаю, где именно (заметил, что один лишь JavaScript не значит UI? BackEnd тоже может иметь несколько ступеней.) Чем больше ты из этого умеешь, тем больше ты сто́ишь. Всё знать досконально нельзя, но иметь определенный бекграунд и общие знания по каждому пункту — огромный плюс в карму и к потенциальному рейту. Если ты сам можешь сделать продукт целиком — ты сто́ишь космических денег.

Запоминай:

- Перед тем как идти за повышением ЗП, — напиши на бумажке три списка: «Что я *сделал* полезного для компании», «Что я *делаю* полезного», «Что я *хочу/собираюсь делать* дальше и что мне для этого нужно». Если уже с первым списком не задалось, то аргументов у тебя нет.
- Подумай дважды, прежде чем просить повышение с формулировкой «мне предложили больше в другой компании». После этой фразы жизнь уже не будет прежней (в текущей компании).
- Расширяй свой профиль — так ты будешь больше стоить.

Сначала бизнес — потом технология

Как пример: (почти) все программисты ненавидят Майкрософт. И я тоже. Наверное, все, кроме тех, кто там работает, и тех, кто не умеет работать на других технологиях (последние тоже ненавидят, но не палятся). Тем не менее, мелкомягкие технологии живут, цветут и пахнут (иногда попахивают), на них пишут и пишут много. А всё по той причине, что они дают целый ряд решений “out of the box”, потому что их индусокод проверен годами — и это те решения, что нужны бизнесу. Я сам 5 лет проработал с MSSQL и даже проникся. К чему это я?
Бизнес определяет технологии, а не наоборот.

Пару лет назад я прочитал в фейсбуке удивленный отзыв одного человека, который на афтепати какой-то конференции подслушал разговор группы фронтендщиков. Разговор был в духе «пофиг на бизнес, главное всё педалить на React».
Я разделяю удивление этого человека, правда последнее время я уже перестал удивляться.

Да, программисты любят хайповые технологии. И их можно понять. Среди этого хайпа время от времени появляются сто́ящие вещи, которые двигают всё вперед. В идеальном мире все должны стремиться к познанию чего-то нового, к развитию и всё такое. Даже в нашем мире это нужно делать. Когда-нибудь (но, к сожалению, не при моей жизни) мы вообще будем жить в мире книжек братьев Стругацких, в полном Полудне, в идеальном комму… бля, это слово сейчас нельзя произносить. Так вот, мы не в идеальном мире. В идеальном мире никто не будет писать на WordPress (и слава богу), а сейчас надо. Ну не лично тебе, друг мой, а в принципе.

<ФИЛОСОФСКОЕ ОТСТУПЛЕНИЕ>

Программиста можно сравнить с переводчиком. Переводчик не меняет суть сказанного — он берёт мысль и передаёт её на другом языке. Хороший переводчик передаёт мысль наиболее точно, с учётом особенностей языка, но не искажая её. Программист тоже, в своём роде, переводчик. С человеческого языка на машинный:

«Я хочу розовую кнопку «Бздыщь!»
=
```
HTML: <button class=”btn btn-pink”>Бздыщь!</button>
+
CSS: .btn-pink{background-color: pink;}
```

Если у тебя есть текст о котятах, а переводчик (к примеру, на английский) специализируется на текстах по квантовой физике, то на выходе будет текст о котах Шрёдингера, несмотря на то, что в оригинале о них ни слова.

Инструмент подбирается под задачу, а не наоборот.

</ФИЛОСОФСКОЕ ОТСТУПЛЕНИЕ>

Сейчас и я, и ты продаём свои мозги за деньги, о которых написано в предыдущей главе. За что платят — то и делаем. Така хуйня, малята. Поэтому не сто́ит городить огород из пяти технологий там, где нужно просто взять, прости господи, WordPress или переписывать всё на React просто потому что это круто. Не нужно искать применение любимым / новым / хайповым технологиям там, где они вообще не нужны.

Кстати, вот ещё один вброс под конец главы: иногда средненький программист лучше хорошего и даже отличного. Просто потому что он будет делать то, что нужно, а не то, что ему хочется.

Запоминай:

- Делай и используй то, что нужно и больше подходит бизнесу, а не то, что нравится лично тебе.
- … а чтобы узнать, что нужно бизнесу — читай выше про коммуникацию.

Скиллы по программированию — это далеко не всё

Поскольку эта книга пишется, в основном, для программистов, то нужно вспомнить и о таких банальных вещах тоже.

Кандидат отбирается по трём ключевым критериям:

— Знание конкретной технической области (язык программирования, фреймворки, библиотеки, опыт использования).

— Абстрактное мышление (логика, математика, возможно, некие специфические знания, если того требует конкретный проект).

— Софт-скиллы (поведение, знание английского, умение общаться с окружающими).

Прошу заметить — пункты практически равнозначны.

К сожалению, сейчас, когда программистов нанимают такие же программисты, рекрутеры заканчивают свою работу на стадии «довести до собеседования», а HR особо не имеют права голоса при найме технаря на работу, то список выше «усыхает» до одного пункта — самого первого. Всё сильно упрощается. Пройдёмся детальнее по каждому пункту.

Знание конкретной технической области
Тут вроде бы всё понятно. Программист должен уметь программировать, всё логично.

Абстрактное мышление (логика, математика)
Первый вопрос, который я часто слышу или читаю в глазах кандидатов на собеседовании: «а нафига оно надо?». А оно надо. Надо потому, что, когда ты выйдешь за рамки своих знаний и не найдёшь прямых ответов в гугле, — тебе именно эти навыки и пригодятся. В этом случае придется включать мозги на полную катушку.

Софт-скиллы
Здесь вроде бы тоже всё ясно. Человек должен уметь не только писать код, но и вести себя подобающе, не срать под ноги и не вонять на пол офиса. Вот только на это, кажется, все забили. Сейчас HR’ы, читающие эту книгу, начинают возмущаться и про себя (или даже не про себя) говорить: «Ничего подобного! Откуда ты вообще, умник, знаешь, как у нас в компании?! У нас такого нет!» Как конкретно «у вас» в компании — я не знаю. Я знаю, как в среднем по больнице. Потому что после работы в «вашей компании» они приходят на собеседование ко мне. И я не могу вчехлить, как их вообще куда-то взяли на работу.

Наличие софт-скиллов может нивелировать некоторый недостаток технических навыков. Забитый же прокачанный технарь может оказаться абсолютно бесполезен вместе со всеми своими фреймворками. Перед тем как имплементировать задачу, её нужно понять, чаще всего ещё и обсудить, и уточнить. Сначала общение, потом программирование. Без первого не будет второго. Даже если где-то посередине впихнуть project manager'а — ему всё равно придётся иметь дело с тем же самым «дуболомом-супер-технарём».

Отдельным пунктом пройдусь по внешнему виду: это пиздец! Если подробней: это полный пиздец! Я понимаю, что программисты редко выходят за пределы обитания своей популяции, но даже в офисе созерцать вид отдельных особей просто невыносимо. Друже, ты действительно считаешь, что можно носить одну и ту же вещь целую неделю, тем более если она уже с дыркой?! Я знаю, что это твоя любимая футболка — купи их таких десяток и меняй по кругу (ты же Middle JS Developer с $3K в месяц). Я думаю, что одной из главных обязанностей HR должно стать «отмывание сотрудников». Или маму / жену в офис вызывать?

Чувствую, скоро можно будет открывать курсы «Привожу ваших сотрудников в нормальный вид» (кстати, есть у меня один человек с большим опытом в этом деле).

Запоминай:

- Учи математику. Решай логические задачи. Не успел в школе — догоняй сейчас.
- Учись выражать свои мысли. Пиши посты, статьи, книжки (вот прям как я). Тренируй дикцию, в конце концов.
- Странно будет звучать в книжке для взрослых дядек и тётек, но... Мойся чаще, минимум раз в день; меняй одежду, покупай новую. И будь добр, следи за своим дыханием — тебе с людьми общаться.

Понты

Существуют уже целые сайты, посвящённые ИТ-шникам, где описаны самые гадкие и нелепые вещи, которые вытворяют «короли жизни». Это, конечно, «жёлтая пресса», но и правда там есть.

Когда я вижу запрос на «эргономическое кресло», время на саморазвитие и чтение книг (в рабочее время, естественно) и «я, конечно, могу и поверстать, но не буду» — и всё это прямо в саммари резюме, — я просто закрываю нахрен это резюме. Я не собираюсь иметь ничего общего с этим человеком.

Запомни, все твои хотелки в конце концов вычитаются из твоей же зарплаты. В том или ином виде. Твоё кресло нужно окупить — это балласт, который не приносит никакой выгоды. С какого чёрта тебе платить за чтение книг и ковыряние в доках какой-то библиотеки, которая нам никогда не пригодится? Клиент за это платить не будет, твой начальник тоже. Так кто же заплатит за твой выпендрёж, товарищ?

Когда я слышу на собеседованиях ответы «я не буду решать эту задачу, она мне не интересна», «та уже поздно, пора идти пить пиво» или хотя бы «я не знаю, всегда так делал и работало»...

Сто́ит больших усилий не послать кандидата прямо на месте. И не загнать его в чёрный список всех знакомых рекрутеров. А ещё не треснуть по башке.

Помни — иногда тебя просто впадлу уволить. Потому что аутсорс (аутстафф) и на тебе зарабатываются деньги — на одном твоём существовании в этой компании. Или просто руки не доходят с тобой разобраться.

К этому разделу можно отнести и программистское нытьё. Оно начинается абсолютно по любому поводу просто потому что нужно поныть — априори. Это обязательная программа. Нет туалетной бумаги — надо поныть. Завезли бумагу — конфеты в офисе не такие. Купили другие конфеты — душно / холодно / жарко / дует / стол не так стоит / монитор в 24" маловат. А ещё, конечно же, как и у всего нашего постсовкового рода — жлобство и любовь к халяве, причем любой. И жалобы, что халявы нет. А если есть, то жалобы, что её мало.

«Дорогие читатели, представляем вашему вниманию экземпляр вида Homo Programmaticus Ahuel Vdoskus. Данный вид обладает уникальной живучестью, собственно, поэтому вы и можете лицезреть эту книгу. Кроме этого, имеет сильно развитые навыки наглости, порой переходящей в хамство, и самолюбования в дозах критичных для нормальной работы головного мозга. Поэтому мы и выгнали его нахуй с собеседования.»

В моей бывшей компании я старался устраивать такую атмосферу и такие плюшки, в которых самому было бы приятно работать. Не скажу, что разрешалось всё, но очень многое. (Пиво посреди дня? Да ради бога. Только работай нормально.) Тем не менее, вышеперечисленного нытья хватало. Правду говорят, что к хорошему быстро привыкаешь. С другой стороны я неоднократно слышал от программистов из других компаний про «церберские условия» (ну, как для них — для программистов). Я не думаю, что прям так много начальников компаний видят себя с кнутом погоняющими рабов на галерах и жлобятся на более качественную туалетную бумагу (хотя такие, безусловно, есть). Мне кажется, тут работает рациональный, прагматичный подход — зачем тратить больше, если программистам всё равно всё не так. С точки зрения высшего начальства, все хотелки — это, в первую очередь, строчки в расходной ведомости. Зачем тратить $500, если можно потратить $50 и результат будет такой же. Нытьё то же самое, только темы меняются.

К тому же, кто знает, может, вам не купили новое кресло потому, что компания концы с концами еле сводит и как воздуха ждёт начала месяца с новыми чеками от клиентов. Такое бывает, уж поверьте. Однажды я остался с $250 за две недели до конца месяца — это были деньги на себя и компанию, а в компании хотя бы на 20 человек это «тьфу». Хорошо хоть туалетной бумагой до этого затарились — хоть обосрись.

Так вот, к чему это я. Неадекватное поведение начальства может быть не только врождённой жадностью и желанием купить себе новый «гелик». Это может быть и ответной реакцией на неадекватность подчинённых. И даже самое адекватное начальство может со временем плюнуть — и со словами «да ну их в жопу» включить режим мудака.

«Действию всегда есть равное и противоположное противодействие».

Запоминай:

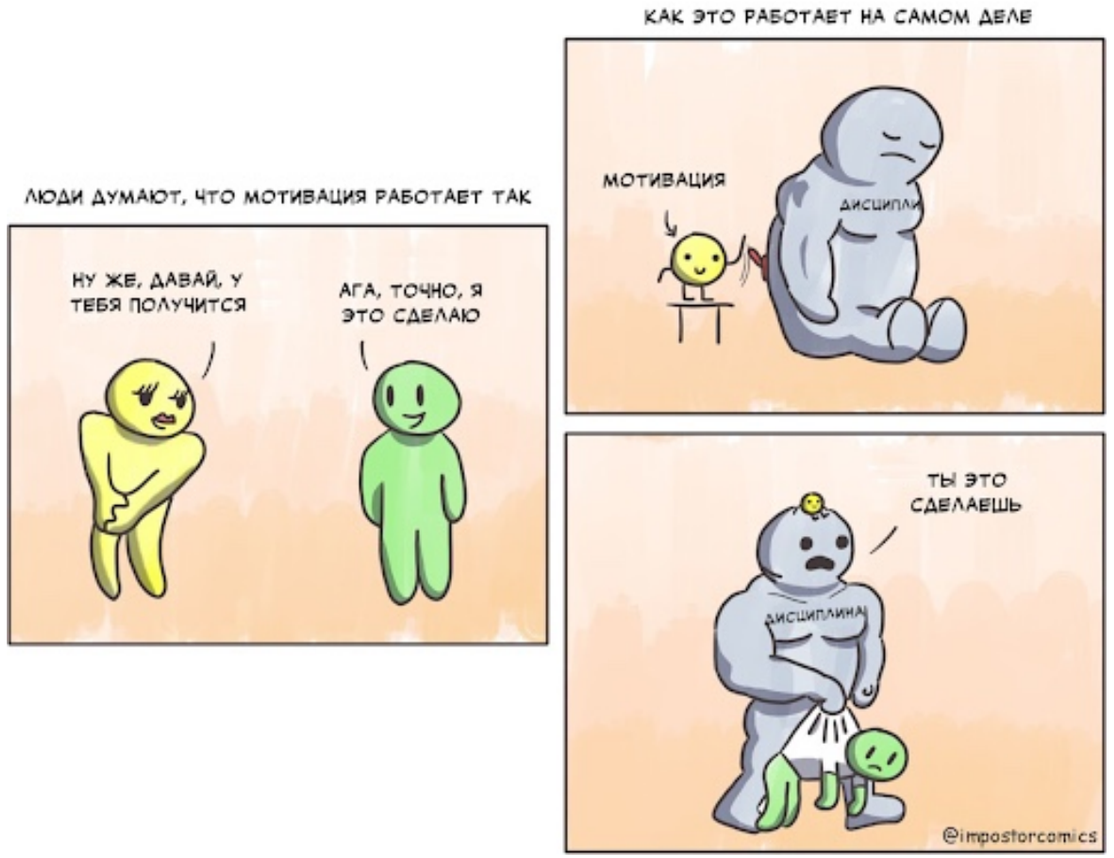
- Иногда тебя просто впадлу уволить… но всему есть край.

Дисциплина

И самодисциплина. Я всегда был сторонником мотивационного подхода (или мне так казалось), но потом я понял, что мотивация — это, конечно, хорошо, но без дисциплины никуда. И если человеку не хватает внутренней дисциплины, надо её навязывать извне. Любыми доступными способами.

Дисциплина — это привычка делать нужные вещи вовремя. Ты чистишь зубы по утрам? Что ж тебе мешает вовремя приехать на работу? Это понятия одного рода. Что мешает пользоваться ёршиком в туалете, убирать после себя на кухне и банально не хамить всем вокруг на каждом шагу? Как я (как начальник) могу всерьёз рассчитывать на человека, которого в пол первого (дня) нет на рабочем месте? (Сейчас мои бывшие начальники вновь ехидно хихикают.) Или человек, от «звезды» которого шарахаются все окружающие со словами «Да я лучше в гугл схожу, чем у него спрошу». Это реальная цитата из жизни.

Мотивация — это разовый пинок под зад или, наоборот, плюшка. И эффект тоже разовый. Любая мотивация рано или поздно сходит на нет. Дисциплина — это непрерывный внутренний пинок под зад самому себе. Вечная самомотивация. Не знаю, как ещё доходчиво объяснить...



Давай проведем сравнение с футболистами. Есть гениальные игроки, есть просто хорошие и средненькие. Так вот даже средненькие футболисты играют в топовых клубах — просто они отлично умеют «таскать рояль» (Гаттузо, Невилл, Макелеле и т.п.). А просто хорошие футболисты становятся отличными. Возьми пример с Роналду (даже если ты его не перевариваешь) — просто хороший игрок, который стал сверх-супер-пупер совсем не из-за таланта, а из-за дикой работоспособности и самодисциплины. Попробуй 15 лет не пить алкоголь, держать диету и чёткий распорядок дня. Да хотя бы пару дней так попробуй.

Мой бывший начальник (и заодно владелец компании) однажды сказал мне:
«Если я вижу, что человек после туалета не помыл руки — то, как правило, он надолго в компании не задерживается.
И не потому, что его за это сразу увольняют. Я через нужных людей намеками передаю, что так делать плохо. Просто человек, который не может следить за своей гигиеной, не может следить и за качеством своего кода.»

Этот человек меня многому научил. В частности, связывать, казалось бы, несвязанные вещи…

Запоминай:

- Заведи себе одно-два занятия, которые тебе нужно делать чётко по расписанию: тренажёрка, английский, медитация, да хоть еженедельные походы к психологу.
- … нет, отвозить ребёнка по утрам в школу/садик не считается. Так же как и чистить зубы по утрам.
- Постоянно опаздываешь — установи себе штраф: 1 минута опоздания / 10 грн (можно $1) и в конце месяца отправляй накопившуюся сумму на благотворительность, родителям или друзьям на пиво (чем не благотворительность).

И напоследок

Не уверен, что многие дочитали до этого места, но тем, кто дочитал — большой респект.

В этой книге сформулированы проблемы, но не так много решений. В идеале каждую главу я планировал написать в формате: «Проблема → Решение». Но идеала не выходит. Возможно, когда я напишу версию 1.0 этой книги, все ответы будут найдены, но что-то мне подсказывает, что нет :) Я думаю, что многие решения индивидуальны.

Главная цель этой книги — заставить задуматься.
И если хотя бы один человек, прочитавший её, сделает это, — то моя миссия выполнена.

Алексей Атемасов
atemas.com