ГЛАВА 1

Как вы думаете, что вы здесь делаете?

Идея карьеры стафф-разработчика (или «инженерный путь») является новой для множества компаний. Организации расходятся во мнениях о том, какими качествами должны обладать самые старшие программисты и какую работу те должны выполнять. Сильвия Ботрос (Silvia Botros) пишет (https://oreil.ly/xwgRn), что тем не менее большинство единодушны в следующем: вершина инженерной карьеры — это не просто «еще более сеньорный сеньор», хотя ясного понимания, что это такое, нет. Поэтому первая глава моей книги начинается с экзистенциального вопроса: зачем компаниям нужны «самые старшие» разработчики? Потом, вооружившись ответом, мы раскроем суть роли стафф-разработчика: разберем требования к техническим навыкам и лидерские качества, которые необходимы на этой должности, а также выясним, что означает «работать самостоятельно».

Обязанности стафф-разработчика могут быть самыми разными, и существует множество способов выполнять их качественно. Однако каждая из них нацелена на решение конкретной задачи, и не любой компании нужен специалист со всем спектром возможных функций. В этой книге я расскажу, что подразумевает роль стафф-разработчика: сфера ответственности, степень влияния на рабочие процессы, место в служебной иерархии, основная цель и другие составляющие. Вы сможете использовать мой опыт, чтобы понять, как вам работать, в каком направлении профессионально развиваться или кого нанять. Наконец, поскольку разные компании по-разному понимают функционал стафф-разработчика, мы постараемся согласовать эти представления с мнением ключевых фигур компании.

Давайте начнем с того, в чем заключается работа стафф-разработчика.

КТО ВООБЩЕ ТАКОЙ СТАФФ-РАЗРАБОТЧИК

Если бы существовало только одно направление карьерного роста (как показано на рис. 1.1 слева), то многие программисты столкнулись бы с трудным и даже жестоким выбором: остаться на должности разработчика и продолжать совершенствовать свое мастерство в написании кода или перейти в менеджмент и развиваться как управленец.

Поэтому многие компании теперь предлагают «инженерную» карьеру, или путь индивидуального контрибьютора (individual contributor), позволяя программистам расти параллельно тому, как растут выбравшие «менеджерскую» карьеру. Пример такого карьерного пути показан на рис. 1.1 справа.

Рис. 1.1. Две карьерные возможности: с одним путем развития и с двумя

Карьерные возможности в разных компаниях сильно отличаются друг от друга, и для их сравнения даже создали специальный веб-сайт *levels.fyi*.[1] Количество этапов карьерного пути и их названия также могут быть различными. Вам даже могут встретиться одни и те же названия в разном порядке.[2] Но слово *сеньор* используется очень часто. Марко Роджерс (Marco Rogers), директор инженерного направления, который разработал должностные иерархии для двух компаний, назвал уровень *сеньора* «переходным» (https://oreil.ly/MpwsJ). Роджерс говорит: «На позициях ниже сеньора люди развивают способность работать самостоятельно, на должностях выше сеньора возрастает влияние на процессы в компании и ответственность за результат».

Иногда уровень сеньора рассматривается как «вершина»: вам больше не нужно расти в должности.[3] Но если вы все же продолжаете свой служебный рост, то становитесь «техническим лидером». Следующая ступень после сеньора обычно называется «стафф-разработчик», это название я и буду использовать далее.

На рис. 1.1 показаны два варианта карьерного роста, и программист уровня сеньор может выбирать, развиваться ли как менеджер или как стафф-разработчик. После перехода на следующую ступень изменение должности со стафф-разработчика на менеджера или наоборот нужно рассматривать как горизонтальное перемещение, а не карьерный рост. Сеньор-стафф-разработчик находится на том же уровне, что и старший менеджер, принципал-разработчик находится на уровне директора и т. д.; в служебной иерархии отдельных компаний могут быть и более высокие уровни. (Я обозначу должности, расположенные выше сеньора, словом «*стафф+*», которое придумал и описал Уилл Ларсон (Will Larson) в своей книге «Staff Engineer».)

---

[1] Я также рекомендую сайт progression.fyi, на котором вы найдете богатую коллекцию карьерных лестниц, опубликованных различными техническими компаниями.

[2] Я слышала об одной компании, которая использовала следующий порядок старшинства: «сеньор», «стафф», «принципал», но потом ее купила другая компания, которая использовала порядок старшинства: «сеньор», «принципал», «стафф». Получился хаос. Купившая компания изменила названия должностей со «стафф» на «принципал» и с «принципал» на «стафф». Это никому не понравилось. И стаффы и принципалы расценили изменения как понижение в должности. Названия все же имеют значение!

[3] Мне нравится определение сеньора, которое дал мой друг Тиарнан де Бурка (Tiarnán de Burca): на этом этапе можно перестать подниматься по служебной лестнице и продолжить работать на том же уровне продуктивности, возможностей и результатов до конца карьеры и все же стать «существенной потерей», если вы покинете компанию.

НЕМНОГО О ДОЛЖНОСТЯХ

Иногда я слышу, как люди настаивают на том, что должности не имеют (или не должны иметь) значения. Они приводят разумные доводы в пользу того, что их компания является эгалитарной меритократией, которая сторонится опасностей иерархии, и говорят: «Мы живем в демократическом обществе и с уважением принимаем все идеи». Конечно, их позиция достойна восхищения: мнение любого человека важно, даже если он находится в самом начале своего карьерного пути.

Но все же должности имеют значение. Команда разработчиков среднего звена написала статью, в которой обозначила три причины необходимости должностей: «Они помогают осознать, что вы растете как профессионал, помогают преодолеть укоренившиеся в отрасли предрассудки и информируют окружающих об уровне вашей компетентности» (https://oreil.ly/oUkHe).

Первая причина является субъективной и, наверное, мотивирует не всех, но две другие описывают эффект, который должности оказывают на людей вокруг вас. Неважно, является компания эгалитарной или нет, в ней всегда будут те, кто по-разному относится к людям, находящимся на разных должностных уровнях, и большинство из нас хотя бы немного волнует собственный статус. Доктор Кипп Круковски (Dr. Kipp Krukowski), профессор практики по предпринимательской деятельности Государственного университета Колорадо, в своей работе 2017 года «The Effects of Employee Job Titles on Respect Granted by Customers» пишет: «Должности работают как символы, с их помощью компании информируют людей внутри и вне фирмы о компетентности своих сотрудников» (https://oreil.ly/zD3kp).

Мы постоянно оцениваем людей и неосознанно строим предположения. Если мы не приложим усилия, чтобы разобраться со своими подсознательными оценками, то скорее всего, попадем под влияние стереотипов. Например, исследование 2015 года (https://oreil.ly/snmmY) показало, что примерно половину из 557 темнокожих и латиноамериканских женщин — специалистов в области науки, технологий, инжиниринга и математики, участвовавших в исследовании, ошибочно принимали за обслуживающий или административный персонал.

Аналогичные предрассудки существуют и в отношении программистов. Бытует мнение, что белые и азиатские мужчины-разработчики более опытны и технически подкованны, а также лучше разбираются в коде, независимо от того, работают ли они уже несколько десятилетий или только вчера выпустились из университета. Женщины, особенно небелые, иногда воспринимаются как менее опытные и менее квалифицированные. Всем им приходится прилагать дополнительные усилия, чтобы доказать свою компетентность.

В вышеупомянутой статье разработчиков среднего звена сказано, что должности помогают разрушить стереотипы и дают представление об уровне компетентности программиста. Формирование правильных ожиданий сберегает специалистам время и силы, которые в противном случае пришлось

бы потратить, снова и снова подтверждая свой статус. Это экономит им несколько часов в неделю.

Должность также может повлиять на работу, которую вы получите в будущем. Как и многие представители технологической отрасли, я каждый день получаю электронные письма с предложениями работы от LinkedIn. Лишь три раза я получила от рекрутеров, не читавших мою анкету, приглашение на более высокую должность, чем была у меня на тот момент. В остальных случаях мне предлагали либо позицию, которая в точности соответствовала моей, либо более низкую.

Вот *что* представляет собой должность стафф-разработчика в иерархии компаний. Но давайте посмотрим, *зачем* нужны уровни технических лидеров. Во вступлении я назвала три основополагающих навыка инженерной карьеры: панорамного мышления, реализации проектов и повышения квалификации. Почему именно *инженеры* должны обладать такими умениями? Зачем вообще нужны стафф-разработчики?

ЗАЧЕМ НУЖНЫ РАЗРАБОТЧИКИ, УМЕЮЩИЕ ВИДЕТЬ КАРТИНУ В ЦЕЛОМ?

Работа в любой инженерной компании требует постоянного принятия решений: какие технологии внедрить, что разрабатывать, продолжать инвестировать в систему или отказаться от нее. Часто круг лиц, обязанных сделать важный выбор, четко определен и последствия предсказуемы. Но некоторые решения имеют фундаментальный характер и затрагивают всю программную архитектуру, и никто не может знать, как это повлияет на другие компоненты системы.

Для принятия правильных решений нужен *контекст*. Опытные разработчики знают, что ответ на большинство технологических вопросов зависит от обстоятельств. Недостаточно понимать плюсы и минусы выбранной технологии: вы должны разбираться в деталях конкретной ситуации. Что вы пытаетесь сделать? Сколько у вас времени, денег, упорства? Каков приемлемый уровень риска? Что нужно бизнесу? Ответы на эти вопросы и есть контекст принятия решения.

Определение контекста требует времени и сил. Каждая команда склонна принимать решения исходя из своих интересов, а каждый программист, скорее всего, будет сфокусирован на достижении целей своей команды. Но часто решения, принятые одной командой, имеют последствия для всей компании. *Локальный максимум* — это лучшее решение для отдельной команды, которое может оказаться неудачным для организации в целом.

На рис. 1.2 показан пример, в котором команда выбирает между двумя программами А и Б. У обеих программ есть необходимые функции, но программу А намного проще установить: запустил — и работает. Программа Б немного сложнее: чтобы она заработала, пару спринтов придется потратить на обсуждение. Никто не хочет так долго ждать.

С точки зрения команды, программа А — абсолютный победитель. Зачем выбирать что-то другое? Но остальные предпочли бы программу Б. Оказалось, что программа А будет постоянно нагружать работой юристов и отдел безопасности, а необходимость аутентификации означает, что ИТ-отделу и команде разработчиков платформы придется работать с этой программой особым образом. Выбрав программу А — локальный максимум, — команда неосознанно выбирает решение, которое потребует намного больших затрат от всей организации. Программа Б всего лишь немного хуже для команды, но намного лучше для компании, а дополнительные два спринта окупятся за квартал. Но сделать правильный выбор может только человек, способный просчитать влияние отдельного решения на все ключевые бизнес-процессы предприятия.

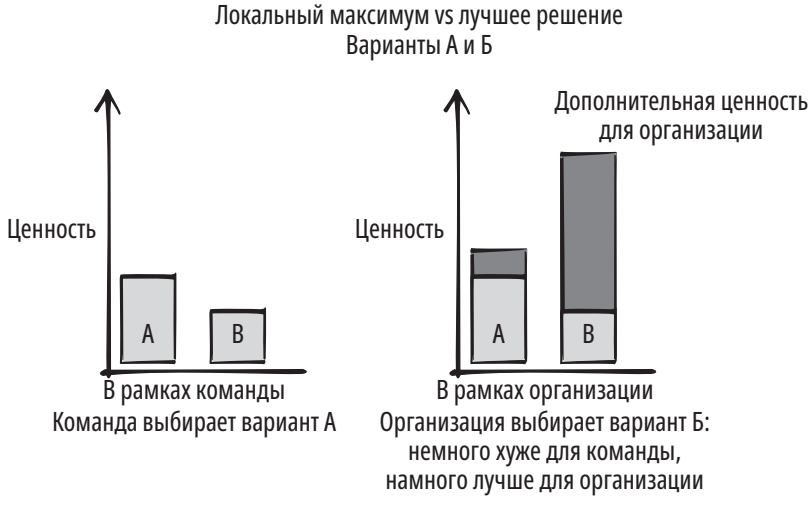

Рис. 1.2. Локальный максимум vs лучшее решение

Чтобы избежать необъективности локального максимума, команде нужен человек, ответственный за принятие решений (или хотя бы способный повлиять на этот процесс), который может взглянуть на ситуацию *со стороны*, то есть учесть интересы нескольких команд и выбрать путь, удовлетворяющий целям организации или бизнеса

в целом. В главе 2 я расскажу, какие приемы позволят получить более объективную картину и принести компании успех.

Необходимо не только проанализировать условия, актуальные в настоящий момент, но и спрогнозировать, какое влияние принятое решение окажет на бизнес в будущем. О чем вы будете жалеть через год? Что принесет вам пользу через три года? Чтобы двигаться в одном направлении, команды должны согласовать технические стратегии: на какие технологии делать ставку, какие платформы стандартизировать и т. д. Все эти серьезные шаги неоднозначны и часто вызывают споры, поэтому для принятия решения нужно выявить все значимые обстоятельства и донести до окружающих их важность. Глава 3 посвящена выбору направления развития всей команды.

Так что если вы хотите принимать основополагающие решения, нацеленные на будущее, то вам нужны люди, которые могут мыслить панорамно. Но разве менеджер этого не может? Или, например, может ли технический директор (CTO, chief technology officer) разобраться в потребностях бизнеса, перевести их на технический язык и рассказать всем, что нужно делать?

Иногда может. В маленьких командах менеджер зачастую является самым опытным инженером, принимает ключевые решения и выбирает направление технологического развития бизнеса. В маленькой компании CTO может быть глубоко вовлечен в разработку мельчайших деталей каждой задачи. Может быть, таким компаниям и не нужны стафф-разработчики. Но положение начальника может затмить технические аргументы: подчиненным часто неудобно спорить с руководством, даже если выбранное им инженерное решение не самое лучшее. К тому же управление людьми само по себе занимает полный рабочий день. Если человек тратит время на то, чтобы стать хорошим управленцем, у него остается мало времени на изучение технических новинок, а если человек старается глубоко погружаться в технические детали, то у него будет меньше возможностей заниматься подчиненными. На коротком промежутке времени это может не вызвать значительных трудностей: некоторые команды успешно функционируют, не требуя особых организаторских усилий. Но если потребности команды вступают в противоречия с техническими задачами, то менеджер должен выбирать, на какой проблеме ему сосредоточиться. И тогда либо участники команды, либо технические вопросы останутся без должного внимания со стороны руководства.

Это одна из причин, по которой организации разделяют управление технической деятельностью компании и руководство людьми. Если у вас несколько разработчиков, то проводить каждое решение через

CTO или старшего менеджера неэффективно, не говоря о том, что это демотивирует сотрудников. Если опытные инженеры получат полномочия, позволяющие им принимать решения по техническим вопросам, а также возможность погружаться в задачи и определять их контекст, то результаты и разработки только улучшатся.

Это не означает, что инженеры должны в одиночку определять техническую политику предприятия. Менеджеры, ответственные за назначение людей на технические проекты, тоже должны участвовать в принятии решений. Как поддерживать согласованность действий менеджеров и разработчиков, я расскажу далее в этой главе, а также в главе 3, в которой мы поговорим о стратегии.

ЧТО НАСЧЕТ ИТ-АРХИТЕКТОРОВ?

В некоторых компаниях должность ИТ-архитектора считается одной из инженерных в должностной структуре. В других компаниях архитекторы — это проектировщики каких-то систем со своим особым карьерным путем, отличным от карьеры программиста, применяющего эти системы. В этой книге я буду рассматривать разработку программного обеспечения и его архитектуры как часть обязанностей стафф+ разработчика, но имейте в виду, что в нашей отрасли это не всегда так.

ЗАЧЕМ НУЖНЫ РАЗРАБОТЧИКИ, КОТОРЫЕ ВЕДУТ КРОСС-КОМАНДНЫЕ ПРОЕКТЫ

В идеале команды должны дополнять друг друга, как фрагменты пазла, закрывая все потребности проекта. А еще в идеале имеется новый прекрасный проект, в котором нет ни априорных ограничений, ни унаследованных систем, а каждая команда целиком и полностью посвящает себя работе. Сферы ответственности команд четко определены, и в них никто не сомневается. В начале проекта каждой команде поручают работу только над одним из компонентов продукта. Технические консультанты компании Thoughtworks назвали такое деление на команды обратным законом Конвея (https://oreil.ly/HdKyK). Конечно, на этом идеальном проекте возникают только трудности, связанные с фундаментальными научными исследованиями и поиском новаторских решений, а люди жаждут разгадывать технические головоломки, чтобы покрыть себя неувядаемой профессиональной славой.

Я бы хотела поработать на таком проекте. А вы? К сожалению, в жизни все совсем иначе. Команды, вовлеченные в любую кросс-командную работу, наверняка образовались до того, как этот новый проект был задуман, и работают над чем-то другим, может быть, даже над чем-то,

что, по их мнению, является более важным. На середине проекта внезапно появляются новые обстоятельства. Сферы ответственности команд начинают пересекаться и разрываются, и это негативно сказывается на программной архитектуре. Сложные и непонятные вопросы оказываются не увлекательными алгоритмическими и исследовательскими задачками, а необходимостью пробираться через лабиринты унаследованного кода, вести переговоры с перегруженными командами, не желающими ничего менять, а еще нужно угадать намерения разработчиков, уволившихся несколько лет назад.[1] На старте виден не весь объем работы, и даже понимание того, какие именно изменения нужно внести, может стать большой проблемой. Внимательно прочитав проектную документацию, вы, скорее всего, обнаружите, что ключевые решения, требующие максимальной согласованности действий команд, были отложены на потом или от них вовсе отказались.

Вот так выглядит реальный проект. Вы можете сколь угодно тщательно распределять части масштабного проекта между командами, но все равно в итоге какие-то задачи не достанутся никому, а какие-то — попадут сразу в две команды. Могут возникнуть задержки в передаче данных и появиться неточности при переводе на другие языки, и все это приведет к конфликтам интересов. Команды будут принимать решения, которые выгодны только им (принцип *локального максимума*), и заведут проект в тупик.

Если вы хотите, чтобы работа продвигалась, то вам нужен человек, который будет отвечать за весь проект, а не за отдельные его части. На начальном этапе этот человек определит объем работ и распределит задачи. Как только проект перейдет в стадию реализации, этот человек, скорее всего, станет автором или соавтором высокоуровневых требований и главным контактным лицом по этим вопросам. Он будет поддерживать должное качество технической работы, прогнозируя риски и задавая вопросы, которые другие боятся задать. Он также будет заниматься неформальным обучением и наставничеством руководителей отдельных частей проекта или просто послужит им хорошим примером. Если работа застопорится, он сможет выявить причины пробуксовки и устранить их, поскольку обладает исчерпывающей полнотой знаний о бизнес-процессах в компании (см. подробнее в главе 6). Он посвятит в тонкости проекта людей, не задействованных в нем, и разъяснит им, в чем ценность продукта и как результаты отразятся на каждом сотруднике.

---

[1] О чем они вообще думали? Зачем они это сделали? И разумеется, последующие разработчики зададут те же вопросы вам.