


ПУТЬ ИНЖЕНЕРА- ПРОГРАММИСТА:

развитие навыков для успешной карьеры

Фернандо
Доглио

 **БОМБОРА**
ИЗДАТЕЛЬСТВО
Москва

УДК 004.4
ББК 32.973.26-018.2
Д59

Fernando Doglio

SKILLS OF A SUCCESSFUL SOFTWARE ENGINEER 1ST EDITION

© Limited company «Publishing house «Eksmo» 2024. Authorized translation of the English edition © 2022 Manning Publications. This translation is published and sold by permission of Manning Publications,
the owner of all rights to publish and sell the same.

Доглио, Фернандо.

Д59 Путь инженера-программиста : развитие навыков для успешной карьеры / Фернандо Доглио ; [перевод с английского К. А. Вантуха]. — Москва : Эксмо, 2025. — 240 с. — (Manning: профессиональные книги для ИТ-специалистов).

ISBN 978-5-04-201473-4

«Путь инженера-программиста» — это полезное руководство для всех, кто хочет построить успешную карьеру в сфере разработки программного обеспечения. Книга предлагает детальный план действий, начиная с получения базовых знаний и заканчивая достижением уровня профессионала, который способен формировать команды разработчиков и управлять проектами.

Здесь содержатся ключевые аспекты разработки: эффективное написание и оптимизация кода, использование лучших практик программирования, таких как рефакторинг и модульное тестирование, а также освоение навыков для успешного прохождения собеседований и продвижения по карьерной лестнице. Особое внимание уделяется коммуникации и командной работе.

УДК 004.4
ББК 32.973.26-018.2

ISBN 978-5-04-201473-4

© Вантух К.А., перевод на русский язык, 2025
© Оформление. ООО «Издательство «Эксмо», 2025

Оглавление

Приветствие	11
1. Как стать успешным инженером по разработке ПО	13
1.1. Ненужное	15
1.1.1. Диплом бакалавра по информатике или в аналогичной области	16
1.1.2. Знание жизненного цикла разработки программного обеспечения	17
1.1.3. Диплом в области математики, физики или аналогичной области знаний	18
1.1.4. Сертификаты	19
1.1.5. Желание работать в динамичной среде	19
1.1.6. Опыт	20
1.2. Полезные навыки	20
1.2.1. Терпение	21
1.2.2. Упорство	21
1.2.3. Настрой на непрерывное обучение	22
1.2.4. Умение принимать критику и учиться на ней	23
1.2.5. Умение общаться с людьми	24
1.3. Что делать после получения работы?	25
1.4. Выводы	25
2. Наилучшие практики в написании кода	27
2.1. Ваш код должен работать	28
2.1.1. Лучшее — враг хорошего	29
2.1.2. Сначала работающий код, потом оптимизация	30
2.1.3. От плохо написанного кода бывает польза	31

2.2. Пишите код для людей, а не для машины	33
2.2.1. Самодокументируемый код — это иллюзия	34
2.2.2. Читаемый код >> код в одну строку	43
2.3. Оверинжиниринг кода — первый из смертных грехов	46
2.3.1. Увидеть оверинжиниринг	47
2.4. Загадка плавающей ошибки	49
2.4.1. Как проводить поиск первопричины	51
2.5. Нужно стремиться быть разработчиком	53
2.6. SOLID, DRY и другие странные словечки	56
2.6.1. SOLID: добиваемся надежности кода	56
2.6.2. Поцелуйте ваш код, или принцип KISS	64
2.6.3. Держите свой код сухим (принцип DRY)	65
2.6.4. Еще одно странное словечко: YAGNI	66
2.7. Заключение	67
3. Продвинутые навыки кодирования: модульное тестирование	69
3.1. Зачем проводить модульное тестирование?	70
3.1.1. Как насчет модульных тестов «сегодня»?	71
3.2. Что тестировать?	76
3.2.1. Тестируйте не больше одной вещи за один раз	77
3.2.2. Тестируйте один модуль кода	77
3.2.3. Тестируйте только собственный код	79
3.2.4. Не тестируйте внешние вызовы	80
3.2.5. Тестируйте только то, что видно другим	82
3.3. Как писать тесты?	84
3.3.1. Ваш новый лучший друг — внедрение зависимостей	84
3.3.2. Укрощение «большой четверки»	87
3.3.2.1. Затычки	88
3.3.2.2. Макеты, дальние родичи затычек	89
3.3.2.3. Шпионим за собственным кодом	91
3.3.2.4. Ты болван!	92
3.3.2.5. Когда что использовать?	93
3.3.3. Ручное выполнение модульных тестов не предполагается	94
3.4. Когда писать свои тесты?	95
3.5. Заключение	96

4. Продвинутые навыки кодирования: рефакторинг	98
4.1. Что вообще такое «рефакторинг»?	99
4.2. Что делать до рефакторинга?	100
4.2.1. Ваш друг, контроль версий	102
4.2.2. Модульные тесты как главное средство	104
4.2.3. Базовая линия для кода	105
4.2.4. Мне нравится, когда план складывается	106
4.3. На чем сосредоточиться во время рефакторинга?	106
4.3.1. Воняет мой код или нет?	107
4.3.1.1 «Магические» значения	108
4.3.1.2 Все делают всё	109
4.3.1.3 Вы слишком примитивны!	110
4.3.1.4 Навязчивое использование операторов switch и IF	112
4.3.1.5 Дублированно-дублированный код	114
4.4. Как делать рефакторинг?	116
4.4.1. Общие техники рефакторинга	116
4.4.1.1 Обобщение свойств и методов	116
4.4.1.2 Консолидация булевых выражений	117
4.4.1.3 Замена условных выражений полиморфизмом	118
4.4.1.4 Переходим от красного к зеленому	120
4.4.1.5 Переименование	121
4.4.2. Инструменты, уменьшающие вероятность человеческой ошибки	122
4.4.2.1 Ваша IDE	122
4.4.2.2 Линтеры	123
4.4.3. Лучшие практики рефакторинга	125
4.5. Что, если вашему коду не нужен рефакторинг?	126
4.6. Заключение	129
5. Человеческий аспект программирования	130
5.1. Как вы учитесь?	131
5.1.1. Вы не обязаны знать все	131
5.1.2. Интернет — великая вещь, но формальное образование не хуже	134

5.2. Побочные проекты	136
5.2.1. О пользе побочных проектов	136
5.2.2. Как насчет работы над проектами с открытым исходным кодом?	140
5.2.2.1 Зачем вносить свой вклад в разработку с открытым исходным кодом?	141
5.2.2.2 Как выбрать проект с открытым исходным кодом?	143
5.3. Просим помощи онлайн	144
5.3.1. Делаем ошибки	145
5.3.2. Избегайте критиканов	147
5.4. Учимся общаться с другими	148
5.5. Заключение	150
6. Как искать работу и успешно пройти интервью	151
6.1. Опыт прохождения технического интервью	152
6.1.1. Чего ждать от технического интервью?	152
6.1.1.1 Беседа на технические темы	153
6.1.1.2 Очные технические тесты	155
6.1.1.3 Удаленные технические тесты	157
6.1.2. Потенциальные признаки опасности	159
6.1.2.1 Семья — это на самом деле не очень хорошо	160
6.1.2.2 Избегайте менеджеров, отстраняющихся от поражений	161
6.1.2.3 Сверхурочная работа не обязательна	163
6.1.2.4 Неограниченный отпуск может оказаться ловушкой	164
6.2. Что нельзя говорить на техническом интервью	166
6.2.1. «Чем вы здесь занимаетесь?»	166
6.2.2. «Не знаю, никогда этого не делал»	167
6.2.3. «Мне не нравилось там работать, потому что...»	168
6.2.4. «Я разработал несколько SPA с использованием SSR и MERN»	168
6.2.5. «Никто этим уже не пользуется»	169
6.2.6. «Это есть в моем резюме»	170
6.2.7. «У меня нет вопросов»	171

6.2.8. «Я разработчик на React»	171
6.2.9. «Linux? Терпеть его не могу. Я работаю под Windows»	172
6.2.10. «Я не знаю, что такое модульные тесты»	172
6.3. Чего ждать в предложении о работе после интервью?	173
6.3.1. Не все то золото, что блестит	174
6.3.2. Действительно полезные льготы	176
6.4. Заключение	177
7. Работа в коллективе	179
7.1. Стараемся оказаться на хорошем счету у менеджера	180
7.1.1. Трекер задач — не сатанинская выдумка	180
7.1.2. Встречи!	182
7.1.3. Я планирую, следовательно, кодирую	185
7.1.4. Не изобретайте велосипед	188
7.1.5. Чего никогда не говорить своему менеджеру	190
7.2. Как быть хорошим коллегой	195
7.2.1. Живите в мире с тестирующими	195
7.2.2. Оставьте самолюбие у порога	198
7.2.3. Научитесь работать удаленно	202
7.2.4. Вам необходима социальная жизнь	204
7.3. Работа над собственными навыками	206
7.3.1. Непрерывное образование	207
7.3.2. Как измерить прогресс в учебе	210
7.3.3. Разбор кода как способ учиться	212
7.4. Заключение	213
8. Общее представление о руководстве группой	215
8.1. Понять вашего руководителя	215
8.1.1. Признаки хорошего руководителя	216
8.1.2. Неприятные истины из уст руководства	218
8.1.3. Постоянные требования обновить статус	222
8.1.4. О распределении задач	224
8.2. Правило 90–10	225
8.3. Как правильно поправить руководителя	227
8.3.1. Руководитель — не лучший разработчик в своей группе	228

8.4. Как правильно разговаривать с клиентом	228
8.4.1. Как правильно поправить клиента	229
8.4.2. Раздраженные клиенты	231
8.5. Отзывы о вашей работе обязательны	232
8.5.1. Почему отзывы важны?	233
8.5.2. Отзывы разного типа	234
8.6. Спасибо	237
8.7. Заключение	237

Приветствие

Перед вами книга «Путь инженера программиста: развитие навыков для успешной карьеры», которая была написана разработчиком и для разработчиков.

По крайней мере, в одной из своих прошлых жизней, больше 20 лет тому назад, я был им. За эти годы я увидел, как наша отрасль превратилась в могучую силу, приводящую в движение весь мир. Программирование стало настолько необходимой профессией, что мы пока не готовы отказаться от нее. Потребность в программистах остается очень высокой.

И тем не менее по непонятной причине получить должность программиста становится с каждым днем сложнее и сложнее. Профессиональное сообщество активно затрудняет доступ для новичков (что безумно само по себе), а в требованиях компаний-работодателей фигурируют «единороги*» наподобие младших разработчиков с 10-летним стажем — и в силу этого начать карьеру в отрасли становится все сложнее.

Поэтому я решил написать книгу о том, как карьера в разработке программного обеспечения (ПО) выглядит на самом деле, о неписаных стандартах нашей профессии и кодексе поведения, которому вам, как вступающему в это всемирное сообществу новичку, предстоит следовать.

В то же время прошу заметить, что эта книга не научит вас писать код: предполагается, что это вы уже умеете — по крайней мере, на базовом уровне. Некоторые технические аспекты профессии в ней затрагиваются, главным образом в силу того, что они лежат в основе любого проекта создания ПО — а это, в свою очередь, повышает вашу квалификацию как разработчика. В основном же книга посвящена соображениям нетехнического характера, которые вам предстоит учитывать на пути от младшего разработчика до техлида. Я не стану компилировать информацию из интернета, а вместо этого буду пользоваться моим собственным опытом и, надеюсь, здравым суждением.

Опираясь на мои ошибки, мои удачи и ваши мысли по поводу прочитанного, вместе мы создадим ресурс, способный помочь другим в подобном

* «Единорогом» принято называть явление или предмет с несовместимыми свойствами. — *Прим. переводчика.*

положении. Поэтому, пожалуйста, заходите на форум liveBook (<https://livebook.manning.com/book/skills-of-a-successful-software-engineer/discussion>) и оставляйте комментарии. Пытаетесь ли вы получить свою первую должность в отрасли, или у вас за плечами уже есть несколько лет опыта, в любом случае финальная версия «Кодекса разработчика» прольет немного света на тот темный путь, который называется нашей карьерой.

Еще раз спасибо за интерес, проявленный к моей книге. Надеюсь, ее чтение доставит вам хотя бы половину того удовольствия, которое я получил от работы над ней.

Фернандо Доглио

1

Как стать успешным инженером по разработке ПО

В этой главе:

- Как избежать ошибочных представлений относительно начального набора навыков
- Как сосредоточиться на навыках, которые повышают вашу квалификацию разработчика

Со стороны разработка программного обеспечения выглядит весьма привлекательной индустрией: безработица отсутствует, зарплаты приличные, всегда есть куда расти, многие задачи предполагают командировки, имеется возможность работать со своего дивана на стартап в Кремниевой долине. Но раз так, почему же в разработке не трудятся все поголовно?

Правда заключается в том, что хотя отрасль может извне выглядеть интересной, попасть внутрь не так просто.

Я буквально знал, что хочу быть разработчиком, еще до того, как у меня появился первый компьютер. Свой выбор я сделал исходя из той стоимости, которую генерировали компьютеры во время моего детства. Но когда мне пришло время начать карьеру и окунуться в реальность, войти в профессию было не просто трудно: профессиональная среда была пугающей и негостеприимной.

У меня не было проводника, способного указать путь в лабиринте собеседований или хотя бы объявлений о приеме на работу. Каждые выходные я проводил по несколько часов за чтением раздела объявлений местной газеты в поисках текстов о найме младших разработчиков без опыта.

Найти первую работу в качестве инженера-разработчика ПО может оказаться, мягко говоря, непростым делом. Большинство компаний, набирающих инженеров начального уровня, ожидают от них навыков работы

с какими-нибудь из самых свежих фреймворков и технологий или понимания достаточно сложных концепций, таких как шаблоны проектирования, наилучшие практики в области разработки и управление версиями. Покончив с этим списком, они переходят к расплывчатым требованиям типа хороших «межличностных навыков» или знания смежных областей ИТ.

Приведенный ниже список требований к кандидату составлен из множества объявлений в Интернете, где приглашаются на работу младшие разработчики:

- Диплом бакалавра в профильной области и минимум год опыта на аналогичной позиции.
- Знание методик безопасной разработки программного обеспечения.
- Знания среднего уровня в области проектирования, разработки, модификации и внедрения ПО, включая объектно-ориентированное программирование.
- Знание других областей ИТ.
- Демонстрация навыков работы с репозиториями ПО.
- Демонстрация навыков эффективной коммуникации и межличностного общения.
- Способность к автономной мотивации и эффективной самостоятельной работе.
- Демонстрация навыков решения нестандартных задач.
- Владение на среднем уровне C#, ASP.NET MVC, SQL Server, TypeScript и React.js.
- Опыт использования Git и GitHub.

Глядя на этот список, как может любой человек, желающий войти в отрасль, не почувствовать себя растерянным и не испугаться предложить свою кандидатуру? Всякий, читающий этот перечень, решит, что ему нужно минимум еще два года опыта, прежде чем его начнут воспринимать всерьез.

Я сам проходил через этот опыт 18 лет назад, и все еще помню, какие вопросы задавал себе тогда:

- Стоит ли мне вообще тратить силы на подачу резюме? Из 10 запрашиваемых навыков у меня в наличии всего 3.
- Стоит ли мне перестать изучать X и переключиться на Y, потому что на этой неделе всем потребовались разработчики на Y?
- Где взять опыт, если я ищу свою первую работу?

По-видимому, каждый разработчик в начале карьеры задает себе те же самые вопросы. Но принципиальный момент заключается вот в чем: они нормальные. Вы не приходите к неизбежному выводу, что не рождены быть программистом, а просто проходите через опыт младшего разработчика.

Именно поэтому я и пишу книгу: чтобы помочь вам построить путь к успешной карьере. Я проходил через все круги ада, выпадающие на долю

любого начинающего разработчика; я получил свою первую позицию, где моя зарплата была мизерной из-за отсутствия опыта. Мне случилось встретить хороших людей, научивших меня многому, касающемуся командной работы, и плохих, показавших мне на своем примере немало вещей, повторять которые не следует.

На протяжении книги я буду рассказывать о том или ином опыте, приобретенном в дороге. Я расскажу о наборе наилучших практик, которые вам предстоит применить к собственному коду. Я покажу вам некоторые основополагающие технические понятия, например, модульное тестирование и рефакторинг, чтобы вы понимали значение этих терминов и важность обозначаемых ими процессов в программном проекте. Расскажу, как находить баланс частной и профессиональной жизни так, чтобы не выгореть в процессе поиска, и дам несколько намеков, полезных на первом техническом интервью. В завершение поговорю о том, что значит быть разработчиком в большом коллективе и даже о том, что значит руководить коллективом, — чтобы вы лучше понимали, чего добивается от вас менеджер. Я надеюсь показать, что вы просто находитесь в начале пути, и что ваши сложности и сомнения совершенно нормальны и представляют собой часть опыта разработчика, — а вы согласились на его приобретение во всей полноте в тот момент, когда решили зарабатывать на жизнь умением писать код.

А что касается этой главы, то в ней я хотел поговорить об основополагающей вещи: что именно нужно, чтобы работать в качестве программиста? Дело в том, что в Интернете много информационного шума и, задав этот вопрос Google, можно получить много непохожих друг на друга статей. У каждого свое мнение, и многие имеют тенденцию концентрироваться на функциональных навыках, то есть на вещах, которым необходимо научиться, прежде чем вообще начинать задумываться о рассылке резюме.

Мой опыт говорит, что эти навыки не самая важная вещь — если понадобится, приобретете их в процессе работы. Самые важные качества разработчика (даже если он ни одного дня не работал по специальности) не относятся к разряду технических.

Давайте пройдемся по обоим спискам: общепринятых ошибочных представлений о том, что необходимо для работы программистом, и действительно полезных и нужных навыков.

1.1. Ненужное

Если вы пытаетесь выяснить в Интернете, как строить собственное обучение, чтобы предложить кандидатуру на первую свою должность в отрасли, или просто размышляете над планированием 5 лет после начала работы, то

позвольте мне поговорить о распространенных ошибочных представлениях, каковы требования для входа в профессию разработки ПО.

Заметьте, что наличие любого перечисленных ниже навыков не снизит ваши шансы, — но они не относятся к числу вещей абсолютно необходимых, чтобы начать работать. Воспринимайте навыки из перечня не как обязательные к исполнению задачи, а как навыки, которые со временем неვნредно было бы приобрести.

1.1.1. Диплом бакалавра по информатике или в аналогичной области

Или, по существу, от четырех лет формального образования. Я пытался его получить, приложил все усилия к тому, чтобы его закончить, но моя первая попытка закончилась неудачей. Я хотел быть инженером-разработчиком ПО, что в моей стране требует 5 лет карьеры, но на втором курсе у меня уже была первая работа, и формальное образование на том закончилось.

Сожалею ли я? Раньше сожалел, и именно поэтому впоследствии получил диплом в технической области (2 года учебы + дипломный проект), чтобы формализовать таким образом полученный к тому времени значительный опыт.

Нужны ли мне были первые 2 года университета? Да, но только потому, что они формализовали многие программные концепции, до которых я так или иначе дошел своим умом, обучаясь программированию для собственного удовольствия. Но в те времена не было экспресс-курсов, доступ в интернет ограничивался модемом со скоростью 2400 бод, а учебные ресурсы сводились к электронным журналам для хакеров (представлявших собой текстовые файлы, разбавленные некоторым количеством псевдографики).

Сейчас ситуация совсем иная, и к услугам всякого желающего научиться кодировать — почти неограниченное количество ресурсов. Это не преувеличение; существуют бесплатные онлайн-источники, например, YouTube или freeCodeCamp.org, где есть все, чтобы пройти путь с нуля ко вполне пригодному для реальной работы состоянию. Разумеется, справедлива истина, что не всякий может научиться, просто впитывая информацию, но существуют и другие варианты, платные курсы в местах типа Udemy или Skillshare, и там, если вам по силам вложить сумму, меньшую, чем плата за университетский диплом, вы получите доступ к онлайн-аудиториям, сессиям вопросов и ответов и возможность контактировать с другими студентами, которые решают те же задачи, что и вы.

Там можно приобрести практические знания, необходимые для того, чтобы приступить к написанию кода, и хотя для многих имеет ценность

упоминание диплома колледжа в резюме, но многие обращают все меньше внимания на этот пункт. Само собой, что это утверждение более верно для одних стран, чем для других, но так же верно, что наша профессия в высшей степени интернациональна: мы можем работать на компанию, находящуюся в любой точке глобуса, и как только мы начинаем это делать, университетский диплом начинает значить все меньше и меньше с течением времени.

1.1.2. Знание жизненного цикла разработки программного обеспечения

Типичный проект разработки ПО должен пройти через следующие стадии:

1. Сначала нужно провести анализ требований, чтобы в точности понимать, что нужно получить на выходе.
2. Затем перейти к планированию проекта, чтобы понимать, когда делать все эти вещи, и сколько времени они потребуют.
3. Третьим шагом будет разработка архитектуры. Как только известны ответы на вопросы «что» и «когда», пора начинать думать об ответе на «как», и в основе своей он будет содержаться в архитектуре.
4. И вот только тогда можно приступать к написанию кода. Большинство разработчиков концентрируется на этой стадии, но, как видите, работа начинается не с нее.
5. Вслед за этим происходит тестирование кода и продукта целиком. Убедиться, что на предыдущей стадии мы получили правильные результаты, абсолютно необходимо перед переходом на следующий шаг.
6. Внедрение продукта в эксплуатацию. Иными словами, передача его пользователям, после чего они уже могут тестировать продукт сами и сообщать вам результаты.

Поскольку процесс цикличен, обычно мы, получив результаты тестирования продукта пользователями, возвращаемся к началу, но я думаю, что идею вы поняли.

Само собой, этот цикл будет отражаться в ваших рабочих задачах, и применять знание его вам придется ежедневно. В то же время выставлять его знание в качестве требования к младшему разработчику — это все равно что требовать от молодого хирурга в его первый рабочий день быть главным на операции, минуя стадию ассистента.

Со временем он будет способен это делать, он, вероятно, читал об этом, — но вряд ли вы захотите поручить ему оперировать самостоятельно в первый же день.

У разработчика все то же самое: не следует рассчитывать, что вы с первого дня будете понимать содержание всех стадий цикла разработки: все равно вы