БОМБОРА
издательство

ДЖОН СОНМЕЗ

КАРЬЕРА В IT

КАК НАЙТИ РАБОТУ, ПРОКАЧАТЬ НАВЫКИ И СТАТЬ КРУТЫМ РАЗРАБОТЧИКОМ

Джон Сонмез

Карьера в IT. Как найти работу, прокачать навыки и стать крутым разработчиком

Серия «Искусство делать бизнес. Как привлекать клиентов в цифровую эпоху»

Текст предоставлен издательством
http://www.litres.ru/pages/biblio_book/?art=69394321
Карьера в IT. Как найти работу, прокачать навыки и стать крутым разработчиком: Эксмо; Москва; 2023
ISBN 978-5-04-188682-0

Аннотация

Эта книга – самый полный карьерный гид разработчика ПО, написанный профессионально, но легко и с юмором. Если вы только готовитесь начать свой путь в сфере ПО, ищете работу или хотите улучшить свои навыки, чтобы продвинуться по карьерной лестнице, книга вам идеально подойдет. Джон Сонмез поделится полезными советами о том, как составить резюме, пройти собеседование, наладить общение с коллегами и начальством, получить повышение или даже создать собственный

бизнес. Под обложкой найдется все, что вам нужно знать о разработке ПО!

В формате a4.pdf сохранен издательский макет.

Содержание

Джон Сонмез
Карьера в IT. Как найти работу, прокачать навыки и стать крутым разработчиком

Вступление

Честно говоря, я и не думал, что так скоро примусь за новую книгу, ведь предыдущий мой труд, «Путь программиста. Человек эпохи IT»[1] (к слову, бестселлер), был опубликован совсем недавно. Хотя, пожалуй, прошло все же не так уж мало времени. Книга «Путь программиста. Человек эпохи IT» увидела свет в декабре 2014 года, а эту, что вы сейчас держите в руках, я начал писать летом 2016 года. Надо сказать, когда пишешь книгу, полтора года не кажутся очень большим перерывом. Создание книги – очень большой труд. Конечно, тот факт, что вон там на полке стоит написанная тобой книга, невероятно греет душу. Однако сам по себе процесс создания текста далеко не всегда столь же приятен. Здесь, наверное, мой читатель вправе спросить, почему же я в таком случае решился написать еще одну книгу? И почему – по крайней мере, по моим меркам – так скоро после выхода предыдущей? Дело здесь определенно не в деньгах, потому что есть масса куда более прибыльных способов потратить время, чем писать книги. Не могу я и сказать, что обожаю писательский труд. И хотя этот процесс доставляет мне определенное удовольствие, порой этот опыт можно назвать скорее болезненным, нежели приятным.

---

[1] Издавалась на русском языке. Питер, 2016 г. – *Прим. ред.*

В чем же тогда смысл участия в процессе, который не превращает тебя в богача, отнимает кучу времени, да еще и причиняет ощутимый дискомфорт? Все просто – это мой моральный долг. Когда я листаю книги, написанные для разработчиков ПО, я понимаю, что на книжном рынке так до сих пор и не появилось ни одного издания, которое было бы посвящено не только секретам успешного начала карьеры, но и последующим ее этапам, а также тому, как добиться максимально возможного успеха. На моем YouTube-канале я читаю тысячи комментариев, написанных разработчиками – молодыми и не очень, опытными и «зелеными», мужчинами и женщинами – в общем, самыми разными людьми из самых разных уголков мира, которые, помимо тем, имеющих прямое отношение к разработке ПО, интересуются и другими вопросами, связанными с этой сферой, не менее важными, чем создание кода.

- Как начать свой путь в разработке ПО?
- Как развить технические навыки?
- Как лучше договариваться о зарплате?
- Работать в штате или быть фрилансером?
- Как общаться с начальством, коллегами, женщинами в сфере IT, а также бороться с предрассудками, будучи женщиной?
- Что действительно нужно знать и как это узнать?
- Что лучше: окончить университет, тематические курсы или учиться всему самостоятельно?

• Как найти работу? А если нет опыта?
• Как пройти собеседование?
• Как одеваться?
• Как продвинуться по карьерной лестнице и выйти на новый уровень?

Этот перечень можно продолжать и продолжать. Вынужден огорчить – найти ответы на эти вопросы будет лишь немногим сложнее поиска иголки в стоге сена. Однако есть и хорошие новости – эта иголка сейчас находится в ваших руках.

Итак, несмотря на мои сомнения по поводу необходимости создания новой книги – особенно после столь незначительного, на мой взгляд, перерыва, – я все же решил это сделать. Данное решение не связано с нестерпимым желанием подарить миру еще одно литературное произведение, хотя, признаться, некоторый творческий зуд я все же испытывал. Я написал эту книгу, потому как считаю, что если человеку что-то нужно, то он должен либо найти это, либо своими силами создать недостающее. Я не нашел материал, который бы полностью меня устраивал, поэтому решил написать его сам.

Ну что ж, друзья, надеюсь, вы составите мне компанию на этом пути!

Нужна ли вам эта книга

Дайте угадаю: вы сидели в Интернете или бродили по книжному магазину, и вдруг краем глаза заметили неверо-

ятно клевую обложку этой книги и не смогли отказать себе в удовольствии взять в руки том (или щелкнуть по ссылке). И теперь вы перелистываете страницы, задаваясь вопросом: «Подойдет ли мне эта книга?» Оставьте все сомнения, эта книга именно то, что вам нужно, хоть я и не имею ни малейшего понятия о том, кто вы, как вас зовут и чем вы занимаетесь. «Но откуда вам знать? – спросите вы. – А вдруг я даже читать не умею?»

На самом деле, я знаю, что вы умеете читать. Будь это не так, вы вряд ли сейчас задали бы этот вопрос. Более того, я могу рассказать вам о вас еще кое-что: у нас примерно одинаковое чувство юмора – ну, по крайней мере, от моих шуток вас не тошнит. Иначе моя книга уже летела бы в мусорное ведро.

Я не хочу рисковать потерять ваш интерес к моему труду, поэтому попробую перевести наш разговор в более серьезное русло. Я убежден, что моя книга будет полезна любому человеку, вне зависимости от его компетенции в области разработки ПО.

Для тех, кто любит все категоризировать и раскладывать по полочкам, позвольте мне поделить всех нас на следующие три категории, после чего вы сможете самостоятельно решить, к какой из них относитесь.

Начинающий или желающий научиться разработке

Если вы только начали осваивать программирование или уже кое-что знаете, но должность программиста пока еще остается вашей заветной мечтой, то наиболее полезной для вас информацией будет та, что содержится в первых двух частях этой книги. В этих главах я рассказываю о том, как стать разработчиком и как устроиться на работу в этом качестве.

Остальные части книги помогут вам восполнить пробелы в знаниях, необходимых для достижения успеха на нелегком поприще разработчика программного обеспечения, более продуктивной работы и продвижения по карьерной лестнице.

В этой книге обсуждаются по-настоящему актуальные темы, речь о которых не заходит ни в одной другой из виденных мной книг и знакомство с которыми избавит вас от той путаницы, что сопутствует началу карьеры, выбору первого языка программирования и способа обучения: в университете, на курсах или самостоятельно.

Разработчик среднего уровня

Если вы относите себя к этой группе, то наиболее интересной для вас может оказаться третья часть книги. Она посвящена восполнению всевозможных пробелов в знаниях и поможет вам эффективно управлять своей карьерой и добиваться успеха.

При этом я совершенно не имею в виду, что первая часть книги будет для вас бесполезной. Думаю, если вы уже умеете программировать, вам будет интересно почитать о том, как должно выглядеть дальнейшее продвижение, как прокачивать технические навыки, искать лучшую работу, писать резюме и вести переговоры о заработной плате.

И если вы заинтересованы в продвижении по карьерной лестнице (а иначе какой смысл работать?), то наверняка найдете полезной последнюю часть этой книги – «Развитие карьеры».

Профи

Я не сомневаюсь, что вы знаете все. И вы точно не хотите, чтобы какой-то непонятный зануда твердил вам что-то о начале профессионального пути, равно как и о том, что такое «система управления версиями», и помогал вам в метаниях между университетами, курсами и самообразованием.

Я прекрасно понимаю – и это правда – вашу позицию. Однако поверьте, эта книга написана в том числе и для таких мастеров своего дела, как вы, и вот почему: почти половина этого издания посвящена непосредственно работе программиста и его продвижению по службе.

Несмотря на то, что количество собранных вами «грабель» наверняка давно перестало поддаваться подсчету и вы являетесь весьма успешным специалистом, вы, вероятно, все же не прочь улучшить навыки взаимодействия с коллегами и начальством, успешнее «продавать» свои идеи, качественнее управлять подчиненными и, черт возьми, добиться получения прибавки к зарплате или более высокой должности.

И даже если все это вам пока неинтересно, рано или поздно обязательно настанет момент, когда вы почувствуете, что уперлись в «стеклянный потолок» своей карьеры и добиться чего-либо еще вам уже вряд ли удастся. Плавали – знаем.

К счастью, я уже успел набить шишек об этот потолок, и теперь могу поделиться с вами проверенными методами, с

какой стороны следует подходить к этой проблеме, как создавать личный бренд, выступать на конференциях, запускать побочные проекты и многое, многое другое.

В конце концов, даже если первые части книги покажутся вам слишком очевидными, вам все равно пригодится информация о технологиях обучения, устройстве на высокооплачиваемую работу, обсуждении заработной платы и выборе между сдельной и штатной работой.

К слову сказать, вы ведь наверняка еще и занимаетесь обучением «салаг»? И, вероятно, вы не всегда понимаете, как и чему их учить? Кроме того, было бы неплохо иметь в запасе несколько советов, которые вы могли бы им дать в начале их карьеры? Как я уже говорил, эта книга пригодится вам вне зависимости от того, кем вы являетесь в мире разработки ПО.

Более того, я осмелюсь заявить, что даже если у вас нет ни малейшего интереса к созданию программ, вам все равно пригодится бóльшая часть этой книги. Хоть она и написана специально для программистов, немалая ее часть посвящена управлению своей карьерой и достижению максимального успеха.

Что ж, если вы добрались этого места, значит, эта книга точно вам подойдет, ведь я понравился вам, а вы – правда-правда! – мне.

Глава 1. Как пользоваться этой книгой

Вы не могли не заметить, что книга у меня получилась довольно большая. Ради любопытства я подсчитал количество содержащихся в ней слов, и знаете что? Тут немногим меньше двухсот тысяч! Это много, и даже очень.

Что ж, будет логично начать повествование с рассказа о том, **почему же книга вышла столь толстой** и **как вы можете извлечь из этого талмуда максимальную пользу**.

В зависимости от того, на какой карьерной ступеньке вы находитесь: джуниор, спец с многолетним опытом за плечами или закоренелый профи, вам будут ближе разные части и главы книги. Кроме того, я уверен, что со временем вы наверняка захотите перечитать некоторые из них.

Зачем я написал эту книгу

Возможно, вам интересны мотивы, которыми я руководствовался, создавая **именно такую книгу**. Во вступлении я уже посвятил этой теме несколько слов, но здесь я хотел бы остановиться на ней чуть подробнее.

Подписчики моего блога[2] и канала на YouTube[3] очень часто спрашивают, как стать разработчиком ПО и как расти в карьере. **Я потратил немало времени, но так и не нашел исчерпывающее руководство, которое было бы полезно как неопытным разработчикам, так и профессионалам, касающееся достижения успеха в карьере,** и помогло бы избежать наиболее распространенных проблем на этом пути.

Некоторые из этих тем поверхностно описаны в разделе «Карьера» книги «Путь программиста. Человек эпохи IT», и со временем я почувствовал, что этому вопросу следует уделить гораздо больше внимания. В то время как книга «Путь программиста. Человек эпохи IT» в целом больше сосредоточена на образе жизни разработчика ПО, включая карьеру, **эта книга посвящена исключительно последней**.

Создавая эту книгу, я старался сделать ее отдельным, ни с чем не связанным руководством. Вам не нужно читать «Путь

[2] https://simpleprogrammer.com.

[3] http://youtube.com/jsonmez.

программиста» или любой другой «учебник», чтобы выжать из этой книги максимум. Фактически вам даже не нужно обладать каким-либо опытом в разработке ПО.

Какие цели преследует эта книга?

Прежде всего я хочу помочь начинающим разработчикам **узнать все важные вещи, необходимые для того, чтобы начать карьеру в столь запутанной и непростой области, как IT**. Говоря откровенно, мне хотелось бы рассказать о наиболее важных аспектах данной сферы, о необходимых для начала работы знаниях, и о том, как получить свою первую работу. Я убежден, что все вышеперечисленное является **камнем преткновения** для большинства начинающих разработчиков.

Следующая задача, которую я ставил перед собой, когда писал эту книгу, – помочь в заполнении пробелов в знаниях тем, кто уже работает программистом, в той области, что касается непосредственно карьеры, а также предоставить рекомендации, как выжить в этом не всегда приветливом мире. Также в этой книге я расскажу, как найти тот самый печально известный баланс между жизнью и работой, как работать в команде, как «продавать» свои идеи, как добиваться повышения заработной платы и как продвигаться по карьерной лестнице, как лучше руководить и как бороться с предрассудками. Наконец, я хочу помочь разработчикам ПО на любом этапе их карьеры **перейти на новый уровень**.

Я расскажу о том, как заработать репутацию в индустрии

разработки ПО, о различных путях по развитию карьеры, а также о книгах, которые следует прочитать. Также мы поговорим о побочных проектах, конференциях и всех остальных вещах, которые помогут вам **перейти на следующий уровень** и **стать первоклассным разработчиком**. Все перечисленное в этой книге можно отнести к категории «гибких навыков», поскольку я буду больше говорить о теории, нежели о практике.

Я считаю, что наше сообщество и отрасль в целом остро нуждаются в подобного рода информации, и я твердо убежден, что в долгосрочной перспективе она окажется более ценной, чем изучение конкретного языка программирования или фреймворка.

Эту книгу я разделил на пять частей, каждая из которых представляет собой набор коротеньких глав. Собственно говоря, как и в книге «Путь программиста».

- Старт карьеры
- Получение работы
- Все, что нужно знать о разработке ПО
- Работа программистом
- Развитие карьеры

Основная цель этой книги заключается в том, чтобы независимо от должности и компетенции вы могли извлечь из нее максимум информации для достижения следующего уровня в карьере разработчика.

Как читать эту книгу?

В целом подход к чтению любой книги очевиден: просто открываете и читаете. (Хотя, если у вас в руках печатная книга весьма увесистого вида, вы можете использовать ее как подставку под монитор.) **Эту книгу можно читать как от корки до корки** – я предполагаю, что большинство людей именно так и сделает, – **так и по частям или главам**.

Предположим, что вы заинтересовались разработкой ПО совсем недавно и пока что только постигаете азы программирования. В этом случае вам стоит начать чтение этой книги с части «Старт карьеры», как наиболее актуальной для вас. Кроме того, это первая часть этой книги.

Или, может быть, у вас за плечами уже есть пара-другая лет опыта работы в должности разработчика ПО. В этом случае, вероятно, вы захотите сразу перейти к части «Работа программистом» или «Все, что нужно знать о разработке ПО», чтобы заполнить пробелы в знаниях.

А может быть, ваш план состоит в том, чтобы как можно выше подняться по карьерной лестнице? В таком случае я рекомендую вам сразу перейти к части «Развитие карьеры». Это будет наилучшим решением.

Что касается **глав** данной книги, то вы можете **читать** их по тому же принципу, что и части, – то есть **в любом порядке**. Изучите оглавление и решите, какие главы вам наиболее

интересны и максимально соответствуют интересующим вас вопросам. Я структурировал книгу именно таким образом, потому что знаю, что по мере развития карьеры разработчика ПО вы будете встречаться с разными проблемами и ситуациями.

Задумывая освоить что-то новое, все сталкиваются с вопросом, а с чего же, собственно, начинать? Ответ на этот вопрос вам пригодится и в том случае, если вы захотите изучить новый язык программирования или новую технологию.

Возможно, в данный момент вы не занимаетесь поиском работы и не намерены вступать в борьбу за повышение зарплаты. Также прямо сейчас вы можете не иметь проблем с коллегами или начальством. Однако нельзя исключать, что когда-нибудь главы, посвященные данным темам, могут стать для вас актуальными.

Знаете, меня всегда бесило, когда я хотел перечитать определенные страницы, но не мог их найти, потому что они были закопаны в какой-то из частей, которые я читал очень давно.

Именно поэтому **я решил структурировать эту книгу так, чтобы ее можно было читать как угодно – хоть от корки до корки, хоть частями в любом порядке**.

Повторенье – мать ученья!

Перед тем как мы перейдем непосредственно к содержимому данной книги, я хотел бы сказать еще пару слов. Во-первых, **если вы не будете применять полученные зна-**

ния на практике, чтение этой книги не принесет вам абсолютно никакой пользы. Вы можете быть полностью согласны с ее автором, но жизнь есть жизнь: нет практики – нет опыта.

Я не хочу быть занудой и постоянно повторять, что «нужно выполнять упражнения в конце каждой главы» (кстати, их там нет) или «нужно конспектировать каждую прочитанную главу», поэтому предлагаю вам гораздо более простое решение этой проблемы, которым пользуюсь сам.

Повторяйте!

Если вы действительно хотите что-то поменять в своей жизни и внедрить в нее какие-либо полезные принципы и привычки, одним из лучших способов добиться этого является **полное погружение в интересующие вас идеи и концепции**. Заставить свой мозг сосредоточиться на конкретных вещах можно, в частности, через повторение. С его помощью можно очень легко запоминать информацию. Лично я так и делаю.

Например, есть несколько книг, которые я читал уже раз десять, поскольку это очень ценный для развития моей карьеры и моей жизни материал, и я хочу по-настоящему усвоить концепции и философские принципы, изложенные в этих книгах.

Итак, **я настоятельно рекомендую вам не только прочитать, но и регулярно перечитывать наиболее актуальные для вас главы этой книги**. Вы можете даже

установить в календарь напоминание, чтобы вернуться к книге через год или любой другой значимый для вас промежуток времени.

Действуйте!

Все идеи и стратегии, которыми я здесь делюсь, не принесут ни вам, ни вашей карьере никакой пользы, если вы не будете предпринимать конкретные шаги по претворению их в жизнь. Чтобы максимально упростить эту задачу, я собрал коллекцию ресурсов под названием «Исчерпывающее руководство по карьере разработчика ПО».

Инструментарий включает в себя: пошаговое руководство по быстрому поиску работы разработчиком ПО (даже если у вас нет опыта), ускоренный курс подготовки к собеседованию, практические рекомендации на тему о том, как следует одеваться на работе, чтобы добиться большего уважения со стороны начальства и коллег, а также «шпаргалку по отладке», которая поможет вам в поисках и устранении багов…

Для посетителей моего сайта инструментарий стоит 175 долларов, но для вас, дорогие читатели, я сделал скидку. Вы сможете получить инструментарий на моем сайте всего за 99 долларов[4].

Дополнительные материалы к книге

Со всеми дополнительными материалами к книге, разбитыми по главам, можно ознакомиться по ссылке: https://simpleprogrammer.com/products/careerguide/links.

[4] https://simpleprogrammer.com/products/careerguide/toolkit.

С годами ваши карьера и жизнь будут неизбежно меняться, и я искренне надеюсь, что моя книга еще долго будет оставаться для вас полезной.

Поехали!

Часть 1. Старт карьеры

«Если у тебя есть мечта, ты можешь потратить всю жизнь на изучение, планирование и подготовку к ее реализации. На самом деле все, что тебе нужно сделать, – начать».
Дрю Хьюстон

Самым частым вопросом, касающимся разработки ПО, является вопрос о том, с чего следует начинать карьеру разработчика. Подчас кажется, что самое большое препятствие на пути к тому, чтобы что-то сделать и стать тем, кем хочется, – это начать двигаться в желаемом направлении. Будь то новый режим тренировок, подготовка к марафону, открытие бизнеса, написание книги или – как в нашем случае – программирование, самая сложная часть – это начало. Легче всего тратить бесчисленные часы на обсуждение и составление планов о том, как и что следует делать. Куда проще, чем реально приступить к делу. Гораздо проще думать о первом шаге и бесконечно рассуждать о том, в каком направлении его следует сделать, чем в реальности сдвинуться с места.

Секрет заключается в том, чтобы за один раз делать только один шаг. Вам нужно собраться с духом и сказать себе: «Хватит планировать и рассуждать. Какой-никакой план у меня есть. Может быть, он и не лучший, но я буду следовать ему». Именно это действие станет первым шагом на вашем

пути. В один прекрасный день вы оглянетесь назад и увидите тысячи сделанных шагов, которые привели вас туда, где вы сейчас находитесь, – на вершине горы, а не у ее основания.

Однако, чтобы сделать первый шаг, вам необходим план. Есть немало потенциальных разработчиков, которые вообще отказываются начать движение, однако при этом не меньшее количество людей пытается действовать наскоком, без какой-либо информационной подготовки или плана и сразу же приступают к делу, не зная, куда они идут и каких целей хотят достичь.

В первой части данной книги мы рассмотрим азы начала карьеры разработчика программного обеспечения. Мы поговорим о том, как составить план развития карьеры, как стать программистом, какие технические навыки необходимы, чтобы добиться успеха в мире IT, и как улучшить эти самые навыки.

Вы также узнаете, какой язык программирования следует выбрать в качестве первого и как лучше всего его изучать: через самостоятельное обучение, посещение курсов по программированию или же путем получения высшего образования.

Дочитав эту главу до конца, вы будете знать достаточно, чтобы приступить к работе и сформулировать реальный план, как и когда это сделать.

Если вы уже являетесь разработчиком ПО, то вы все равно сможете найти в этой части кое-что полезное для себя,

например внести коррективы в план развития своей карьеры и решить, как вы будете совершенствоваться в области разработки ПО. (А еще вы можете воспользоваться рекомендациями из этой книги, чтобы помочь своим знакомым, которые хотят стать программистами.)

Я могу дать вам все возможные советы, открыть все существующие секреты мира разработки ПО и подробно описать, какой путь следует выбрать, но от всего этого обилия информации не будет никакого толку, пока вы не соберетесь и самостоятельно не сделаете первый шаг. Как я люблю говорить, нужно «довериться процессу».

Итак, давайте приступим.

Глава 2. С чего начать

Когда я только начинал работать программистом, я не имел ни малейшего понятия о том, чем я занимаюсь. Мне было **непонятно абсолютно все**, и в голове крутилась единственная мысль: «У меня ничего не получится». Зачем я все это вам рассказываю? Мне кажется, что раз вы взяли в руки эту книгу, вы вполне можете испытывать точно такое же чувство. Не переживайте, **это нормально**. И **даже естественно**.

Позвольте мне подчеркнуть еще один момент. Чтобы стать хорошим программистом, вам **не нужно быть гением** и **обладать интеллектом выше среднего**.

Если вы решили создавать программы и при этом вас не сковывает чувство страха и вы не чувствуете, что тонете в обрушившейся на вас информации, скорее всего, вы что-то делаете не так. Ну или, может быть, вы – андроид. Или то, и другое одновременно.

Во всех остальных случаях вы будете постоянно испытывать трудности и раз за разом понимать, что вы в очередной раз запутались. Но я вам обещаю, что со временем это пройдет.

Как начинал я?

Рискуя прослыть в ваших глазах старым занудой, я все же скажу, что, когда я начинал программировать, у меня не было тех ресурсов, которые доступны вам сегодня. У меня не было **вообще никаких ресурсов.**

Я скачал исходный код крайне популярной игры в жанре МПМ. (Многопользовательский мир. Это что-то типа World of Warcraft, только без графики. Только текст. Да-да, это было в те далекие времена, когда мы дозванивались до «бибиэсок» с помощью модема). Когда я открыл файлы с кодом, то ужаснулся. Я всего лишь хотел создать свою собственную МПМ-игру и добавить в нее кое-какие свои идеи, но чтобы это сделать, нужно было нырнуть в эти дебри непонятных символов. В общем, именно это я и сделал. Я менял значения всех подряд переменных. Я искал код, отвечающий за вероятность нанести критический урон. После того, как я нашел его и изменил, я перекомпилировал игру и посмотрел, что получилось.

Иногда я получал, что хотел. А **иногда мое творение даже не компилировалось**. Я просто смотрел что работает, а что – нет. Вот так и выглядела моя «учеба»! Несмотря на то что я толком не понимал, что делаю, уже через неделю таких экспериментов мне удалось слепить собственную МПМ с уникальными «фичами».

Конечно, до настоящего программиста мне было еще далеко, но начало было положено. Столь необходимое каждому из нас.

Знаете, зачем я рассказал вам эту историю? Потому что **все проделанные мной действия лучше, чем любая книжка, курс или университет. По сути, это единственный правильный способ.**

Все, что вам нужно, – это просто поковыряться тут и там, понажимать кнопочки. И даже «сломать» что-нибудь в процессе. Кстати, советую иногда отвлекаться от этого увлекательного занятия и периодически заглядывать в главу «Учимся учиться: как обучать себя» в книге «Путь программиста. Человек эпохи IT»).

Обратите внимание, что **«научиться программировать» и «узнать, как попасть в мир разработки ПО» – две абсолютно разные вещи**. Безусловно, чтобы стать разработчиком, необходимо учиться писать код, однако есть кое-что еще. Эта глава как раз об этом.

Каков он, мир разработчиков?

Я хочу рассказать вам кое-что о разработке ПО. Она одновременно сложнее и легче, чем кажется. Я посвятил этому вопросу целый раздел, но сейчас давайте просто немного оглядимся.

Разработка ПО – это не «просто программирова-

ние». Конечно, программирование – бóльшая часть разработки, но на одном коде далеко не уедешь. Особенно если в ваши планы входит построение успешной карьеры. Суть разработки ПО заключается в том, чтобы придумать способ автоматизировать ручной труд или делать что-то, что вручную дается непросто.

Рассмотрим в качестве примера текстовый редактор. Я пишу эту книгу в приложении «Документы Google». Без него мне пришлось бы достать с чердака печатную машинку или взять шариковую ручку и лист бумаги. Если бы я захотел отформатировать документ по мере ввода, мне пришлось бы вручную настраивать машинку. А для исправления ошибок мне пришлось бы открыть флакончик с корректирующей жидкостью (и, наверное, бутылочку виски).

Конечно, одних только «Документов Google» будет недостаточно, чтобы напечатать книгу. Мне нужен компьютер, который состоит из кучи микросхем и программ, принтер и все в таком духе. Но я думаю, что вы уже поняли, к чему я веду.

Чтобы стать хорошим разработчиком, нужно знать одну простую, но очень важную истину:

Прежде чем что-то автоматизировать, нужно научиться делать это «что-то» вручную.

Понимание проблемы

Нередко бывает так, что начинающие разработчики (а подчас этим грешат даже профи) пытаются создавать программу, не до конца понимая, что же она, собственно, должна делать. Иногда люди просто хотят писать код (что, в общем-то, вполне приемлемо, если вашей целью является программирование для себя, а не разработка ПО на коммерческой основе). Но если вы, дорогие читатели, решились открыть эту книгу, то, скорее всего, ваш уровень притязаний находится уже **выше простого «клепания кода»**.

Процесс разработки ПО всегда начинается с констатирования проблемы, которую будет решать ваша будущая программа. Задайте себе вопрос: «А что я, собственно говоря, автоматизирую?»

Различные методологии разработки предлагают разные подходы к решению этой задачи, однако сейчас речь не о них. Мысль, которую я хочу донести, состоит в том, что прежде чем приступать к решению проблемы, необходимо сформулировать ряд требований к будущей программе и понять, в чем конкретно состоит проблема, которую вы будете решать.

В зависимости от ситуации, может быть достаточно поговорить с потенциальным заказчиком и узнать, чего он хочет и как, по его мнению, должно работать ПО. В других случа-

ях вам понадобится формализовать процесс и написать спецификацию в виде документов.

Проектирование

После того как вы придете к пониманию сути проблемы, накидайте примерный вариант ее решения в виде кода. И да, все это нужно сделать до того, как вы приступите к набору программных инструкций. Отнеситесь к этому шагу как к созданию чертежа при строительстве здания. Опять же, различные методологии разработки предлагают разные подходы к созданию подобных вещей, но **перед тем, как погружаться в написание кода, нужно иметь хотя бы примерный план**.

Такой подход применим при решении проблем как маленького масштаба, так и большого. Некоторые адепты методологии Agile (о ней мы поговорим в следующих главах) считают, что можно обойтись и без плана, главное – начать писать код. Несмотря на то что Agile не требует слишком тщательного предварительного проектирования, **полная импровизация – все еще не лучший выбор**. Вы не сможете построить дом, если будете наобум забивать гвозди в бревна.

Собственно, само программирование

Разобравшись с ви́дением работы будущей программы, вы можете либо написать несколько тестов, которые позволяют понять, что будет делать приложение (такой подход также известен как TDD, или «разработка через тестирование»), либо приступить непосредственно к программированию. В следующих главах мы вернемся к обсуждению TDD.

Написание кода – это отдельная дисциплина, поэтому я не буду здесь вдаваться в подробности этого процесса, но порекомендую к обязательному прочтению две отличные книги, посвященные написанию хорошего кода.

Первая книга называется «Совершенный код. Практическое руководство по разработке программного обеспечения»[5], ее автором является Стив Макконнелл. Это классический труд, который должен прочитать каждый разработчик.

Вторая – «Чистый код. Создание, анализ и рефакторинг»[6] за авторством Роберта Мартина. Это тоже классика, которая научит вас писать более качественный код.

Эти книги рассказывают о том, как структурировать и писать код, который будет легко понять и поддерживать другим программистам. **Материал этих изданий оказал глубокое влияние на мои навыки программирования**, осо-

---

[5] Издавалась на русском языке. Русская редакция, 2019 г. – *Прим. ред.*

[6] Издавалась на русском языке. Питер, 2018 г. – *Прим. ред.*

бенно в том, что касается ясности кода и проектирования ПО.

Тестирование и развертывание

Итак, код готов. Значит ли это, что программа готова к выпуску?

Нет! **Сначала код нужно протестировать**. Повторюсь, различные методологии разработки предлагают разные подходы. А в общем, просто помните, что перед передачей программы заказчику ее надо проверить.

Например, использование методологии «Водопад» (Waterfall) предполагает тестирование в конце проекта. А при использовании методики Agile тестирование происходит в конце каждой итерации создания ПО, которая обычно длится пару недель.

После того как тестировщики дают «добро» на выпуск кода, наступает этап развертывания, который представляет собой отдельный процесс.

Вы, наверное, уже обратили внимание на то, что я лишь вскользь упомянул о данном этапе создания ПО, – потому что этой теме в книге посвящена целая глава. Развертывание – это процесс установки готового ПО на сервер, загрузки в магазин приложений (типа App Store или Google Play) или предоставления доступа к программе конечным пользователям любым другим способом. (Этот процесс может быть до-

вольно сложным.) Попутно код ~~можно~~ следует отправлять в **репозиторий исходного кода** , где сохраняются различные версии и изменения вашего ПО.

Если вы разрабатываете сложную программу обработки данных внушительного объема, то, скорее всего, эта программа будет **использовать какую-нибудь базу данных**. Последняя обычно хранит пользовательские данные или конфигурационную информацию приложения, и может обновляться в процессе правок исходного кода.

Многие команды разработчиков используют ту или иную форму непрерывной интеграции, чтобы код собирался автоматически, когда разработчики «присылают» свои части программы.

Разработка – это не просто набор кода

И, наконец, не забывайте про отлов багов (отладку). Бóльшая часть вашей работы в качестве разработчика ПО будет заключаться в том, чтобы понять, почему ваш (или чей-то еще) код не работает. **Как я уже говорил ранее, разработка ПО включает в себя немного больше, чем просто написание кода.**

Я считаю, что вы должны осознать этот нюанс еще до того, как устроитесь на работу. А лучше всего будет, если вы заранее обзаведетесь опытом во всех вышеперечисленных вещах.

Но не бойтесь. **Цель этой книги – информировать вас обо всех аспектах** – или, по крайней мере, указать правильное направление. Возможно, вам придется самостоятельно собирать свой багаж (знаний), а я подскажу, что стоит с собой взять.

Главное – план!

«Так, Джон, мы поняли, что разработка – это не только программирование, и нам придется проводить кучу времени за отладкой, развертыванием и бла-бла-бла, – скажете вы. – Ну а начать-то с чего?» **Спешу вас поздравить: вы уже начали!**

Поскольку вы взяли в руки книгу типа этой и начали осознавать, что разработка ПО – это нечто большее, чем просто написание кода, то на старте вы уже дадите фору **большинству других разработчиков.**

Да-да, я знаю, что все это лишь слова, но поверьте мне, это не пустые нравоучения. Когда-нибудь вы тоже станете старым вредным профи и будете вещать то же самое.

А теперь давайте поговорим более предметно. О плане. Он нужен всем. Настоящий, реальный план, без лишней воды. В частности, вам нужен план, как из бестолкового джуниора превратиться в гуру разработки. Существует множество путей, ведущих к этой цели, и о некоторых из них мы обязательно поговорим в следующих главах. Запомните главное:

неважно, какая дорога выбрана, важно, **встав на тропу, не сходить с нее.**

Как составить план?

Давайте поговорим о том, из чего должен состоять подобный план.

В первую очередь надо **объективно оценить**, каков сейчас уровень ваших навыков и чему вы хотите научиться.

• У вас есть опыт программирования?

• Вы знаете какие-нибудь языки программирования?

• Вам уже приходилось писать программы (пусть даже самые простые) или вы пока что находитесь в самом-самом начале пути?

• Обладаете ли вы какими-нибудь навыками, о которых я говорил, помимо написания кода?

• Знаете ли вы что-нибудь о базах данных, управлении версиями ПО, разработке через тестирование, отладке или методологиях создания программ?

Задайте себе еще и такой вопрос:

• В какой сфере разработки ПО вы хотели бы развиваться?

Да-да, конечно, все хотят разрабатывать игры, но стоит ли в это бросаться? Если вы планируете начать карьеру программиста именно в этой области, подумайте, готовы ли вы конкурировать с огромным количеством таких же «гейм-ди-

зайнеров»?

Очень много людей отправляются в дорогу, не продумав свой путь.

Потратьте немного времени, чтобы ответить себе на эти вопросы и составить план развития. Я, конечно, буду помогать вам на протяжении всей книги, но **это все, что я могу сделать**, уж простите.

Я могу вам очень подробно рассказать, как стать хорошим или даже отличным разработчиком, но пока вы не превратите эту информацию в **собственный уникальный план** и не начнете ему следовать, книга не принесет вам большой пользы.

Создание плана

Допустим, что вы уже обдумали все, написанное мною выше. Тогда мы можем приступить к созданию плана! На мой взгляд, лучший способ составить план – **определить цель и наметить путь к ней из вашего текущего положения** . Как я уже говорил, вместо того чтобы просто «учиться программировать» вы должны сначала четко сформулировать для себя, **специалистом в какой области разработки хотите стать.**

В части «Все, что нужно знать о разработке ПО» я расскажу о различных видах ролей и вакансий, которые вы можете рассматривать для себя. Тем не менее вы можете самостоя-

тельно проанализировать, какие сферы разработки вам подойдут лучше всего. **Вы разберетесь, что конкретно** вам следует изучить, как должны выглядеть ваши резюме и портфолио и какие курсы имеет смысл посетить. Кроме того, вы даже сможете примерно представить, какая работа и в какой компании вас устроит.

Я понимаю, что сделать выбор очень трудно, но, поверьте мне, это очень и очень важно! Скажу так: **чем точнее вы представите финальную цель, тем проще вам будет ее достигнуть.**

Как выбрать сферу деятельности?

Представьте, что вы решили стать «спортсменом». По-моему, это звучит как-то расплывчато. Как, например, будут выглядеть ваши тренировки? Может быть, вы будете «тягать железо» и бегать, или, наоборот, станете мастером спорта по плаванию. А может, ваше призвание – это ракетка и мяч?

А может, лучше всего будет осваивать все и сразу, чтобы быть готовыми к занятию любым видом спорта? Согласитесь, **звучит как бред сивой кобылы.** Примерно так же бредово звучит идея стать абстрактным «разработчиком». Как ни крути, но вам придется **выбирать вид спорта** по душе. Как только вы определитесь со специализацией, то поймете, как тренироваться и на какие соревнования ездить. Поверьте – вы существенно упростите себе жизнь.

Ваш план должен начинаться со списка навыков, которые вы хотите приобрести. Понимание того, какие навыки необходимы и как их можно приобрести, – крайне важно. После этого вы должны разобраться, что требуется для получения желаемой должности и как успешно пройти собеседование.

Наконец, нужно составить план устройства на работу. Где вы будете ее искать? Как именно? **Какие вакансии будете рассматривать?** Я бы еще добавил информацию о том, как будет выглядеть ваше обучение и развитие уже после устройства на работу.

Этот перечень способен ошеломить, но не волнуйтесь. Именно для того, чтобы **упростить вам жизнь во всех этих вопросах, я и написал эту книгу**. В следующих главах мы поговорим обо всех направлениях и способах развития, а также о том, как лучше искать работу.

Ну а теперь вы можете приступать к **набрасыванию своего плана**, чтобы попытаться понять, **каким разработчиком вы хотите стать**.

> **Вопрос Джону!**
>
> **А как понять, что я хочу разрабатывать?**
>
> Прекрасный вопрос! Возможно, вы пока не знаете, какие вообще бывают варианты. Ну, помимо разработки игр. К счастью, этот вопрос не очень сложный, однако на его исследование все же придется потратить немного времени.
>
> Далее в книге мы рассмотрим несколько сфер

разработки ПО. Большинство из них описано в части «Все, что нужно знать о разработке ПО», тем не менее самостоятельное исследование вопроса может быть полезно. Поспрашивайте знакомых программистов, чем они занимаются и какое ПО разрабатывают. Возможно, что-то особенно привлечет ваше внимание. В этом случае вы можете смело приступать к изучению технологий и языков программирования, которые связаны с заинтересовавшей вас сферой.

Существует огромный спектр технологий, на которые можно обратить внимание.

Интересны веб-приложения? Мобильная разработка? А может быть, вам было бы интересно писать код, который позволит холодильнику правильно регулировать температуру? Или отправлять космонавтов к далеким звездам?

Подумайте об этом, а затем самостоятельно исследуйте вопрос. Если задать его правильно, то найти ответы будет несложно.

Конкретный пример

Я считаю, что рассмотрение реального примера – весьма полезная методика. Поэтому давайте рассмотрим следующую вполне конкретную ситуацию, в которой человек планирует стать веб-разработчиком на Node.js.

Цель. Стать разработчиком Node.js

План

Обучение

♦ Выучить основы JavaScript

♦ Понять, как работают веб-страницы и какие существуют технологии веб-разработки (например, HTML и CSS)

♦ Выучить основы Node.js

♦ Научиться создавать приложения на Node.js

♦ Узнать о различных фреймворках и технологиях, которыми пользуются разработчики Node.js

♦ Дополнить свои знания о Node.js

♦ Узнать о технологиях баз данных, используемых вместе с Node.js

♦ Разобраться с базовыми понятиями компьютерных наук:

♦ Алгоритмы

♦ Структуры данных

♦ Узнать о лучших практиках написания хорошего кода

♦ Узнать, как разрабатывать архитектуру приложений на Node.js

♦ Подготовиться к поиску работы

♦ Прочитать описания вакансий разработчиков на Node.js и узнать требования работодателей

♦ Составить список компаний, где вы хотели бы работать

♦ Начать посещение тематических встреч в вашем городе (клуб по интересам)

♦ Начать общаться с программистами, которые разрабатывают на Node.js

♦ Нанять специалиста по созданию резюме

♦ Проанализировать вопросы, которые задают на собеседованиях

♦ Попробовать пройти собеседование (сымитировать)

♦ Создать портфолио из нескольких приложений

Получение работы

♦ Связаться с компаниями и известить их о своих навыках и поиске работы

♦ Принять участие в стажировке или подать заявки на замещение должности джуниор-разработчика

♦ Подавать заявки как минимум на две вакансии в день

♦ Подвести итоги после собеседования, чтобы понять, над какими навыками нужно еще поработать.

Поначалу ваш план будет далек от идеала, но по мере проработки он будет становиться все более полным.

Лучше плохой план, чем никакой. План всегда можно изменить, но, если его нет, вам придется тяжело. Вы будете хвататься то за одно, то за другое, в результате вас постигнет разочарование и вы наверняка захотите все бросить.

В следующей главе я помогу вам составить хороший план, и мы обсудим технические навыки, необходимые для становления разработчиком ПО.

Глава 3. Необходимые технические навыки

Я убежден, что каждый программист должен развивать не только технические, но и так называемые гибкие навыки. Я даже написал об этом целую книгу: «Путь программиста. Человек эпохи IT». Тем не менее нельзя недооценивать **важность технических навыков**.

Сказанное выше означает, что если вы не умеете писать код, то даже самые развитые гибкие навыки не помогут вам добиться успеха на поприще программиста. Стать толковым менеджером или бизнес-тренером вы сможете, а разработчиком – вряд ли.

Однако если вы уже добрались до этой главы, я вправе сделать вывод, что вы заинтересованы стать именно программистом (или даже лучшим программистом), поэтому давайте перейдем к обсуждению непосредственно технических навыков, которые вам потребуется освоить.

Навыки, за которые платят

Эта тема будоражит многих начинающих разработчиков. В начале пути кажется, что нужно **знать так много всего**, что совершенно **непонятно, за что браться в первую очередь**. Именно для того, чтобы избавить вас от этого дис-

комфорта, я и написал эту книгу. Я расскажу вам о **наиболее важных навыках**, которые принесут вам **наибольшую выгоду в карьере** разработчика.

Хочу сразу предупредить, что приведенный далее список технических навыков, которые вам понадобятся как программисту, не является исчерпывающим. Однако я постарался привести наиболее важные из них. В части «Все, что нужно знать о разработке ПО» я **подробно разберу каждый из представленных здесь навыков**.

Довольно слов! Давайте перейдем к делу и кратко ознакомимся с перечнем технических навыков, которые я считаю наиболее важными.

Всего один язык программирования

Думаю, этот вопрос точно является наиболее животрепещущим, не так ли? Нельзя быть программистом, не зная ни одного языка программирования, – понимаете, о чем я? О выборе конкретного языка мы поговорим в главе «Как выбрать язык программирования», так что пока **не заморачивайтесь этим вопросом**. Единственное, что я хочу сейчас сказать, **– выбор языка программирования не настолько важен, как вам сейчас кажется**.

Вместо этого давайте поговорим о том, почему лучше учить один конкретный язык, чем хвататься по чуть-чуть за все сразу. Многие начинающие программисты зачастую

склонны идти именно вторым путем, чтобы увеличить свои шансы быть востребованными на рынке труда. Разумеется, мои слова не означают, что нужно всю жизнь пользоваться лишь одним каким-то языком. **Совсем нет, наоборот**, но на начальных этапах большое количество языков программирования лишь запутает вас и помешает вам уделить достаточно внимания многим другим техническим навыкам, не менее важным.

Вместо этого я бы посоветовал **сосредоточиться на изучении одного конкретного языка и его тонкостей**, чтобы со временем вы могли уверенно читать и писать код на нем. Помните, я говорил о важности выбора области ПО, разработчиком которого вы хотите стать? Так вот, к выбору языка программирования применима та же самая логика.

Как структурировать код

После (а лучше в процессе) изучения языка программирования очень важно научиться правильно структурировать код. Выше я уже упоминал об одном **отличном ресурсе**, который поможет вам овладеть этим навыком: книга Стива Макконнелла «Совершенный код. Практическое руководство по разработке программного обеспечения».

Что я понимаю под структурированием кода? **Хороший и понятный код, который не требует большого количества комментариев,** – словом, **удобочитаемый**.

Увы, но многие разработчики ПО на протяжении всей своей карьеры не уделяют должного внимания развитию этого навыка. Структурирование кода – это основной признак, по которому я и многие мои коллеги оценивают навыки и компетентность конкретного разработчика. Именно хорошая структура кода свидетельствует о любви к своему делу, а не простое механическое исполнение своих обязанностей. **Структурирование кода – самая творческая часть разработки ПО**. Этот аспект важен потому, что вам и вашим коллегам предстоит работать с тем, что вы создали. На практике вы будете тратить гораздо больше времени на поддержание существующего кода, чем на создание нового.

Я не буду вдаваться в подробности, как правильно структурировать код, поскольку уже предоставил вам отличный источник информации по этой теме. Хочу лишь подчеркнуть, что **нужно с самого начала совершенствовать навык создания хорошего, чистого кода**, не откладывая его освоение на «когда-нибудь потом».

Я могу с определенной долей вероятности утверждать, что, даже если вы являетесь новичком, но при этом уже можете хорошо, чисто, кратко и понятно писать программы, то практически любой интервьюер, который увидит ваш код, **посчитает вас профессионалом**. И в какой-то степени это действительно будет правдой, потому что наличие данного навыка показывает, что вы относитесь к разработке как к профессии, а не как к «просто работе».

Объектно-ориентированное проектирование

Необходимость изучения данной методологии – штука спорная, особенно если вы учите не объектно-ориентированный язык. Однако **множество компаний и разработчиков в своей работе опирается на объектно-ориентированный подход, поэтому следует иметь общее представление о том, что же такое ООП и с чем его «едят»**.

Объектно-ориентированное проектирование – способ разработки сложных программ, который подразумевает разбиение кода на отдельные классы или объекты (экземпляры классов), которые содержат в себе (инкапсулируют) некоторый функционал и имеют определенные роли и обязанности. Разработка ПО **подразумевает управление сложностью**.

С помощью данной методологии мы можем определить и спроектировать сложную систему, которая будет состоять из множества взаимодействующих между собой компонентов, в противовес попытке решения всех проблем одним махом.

В настоящее время стали довольно популярны языки программирования, основанные на другой концепции – функциональной, однако **наиболее востребованными языками и шаблонами проектирования ПО по-прежнему являются те, что основаны на объектно-ориентированном подходе и анализе**.

Чтобы разобраться в сути ООП, вы должны понять, что такое класс, чем отличаются различные типы наследования и в каких случаях их следует применять, а также что такое «полиморфизм» и «инкапсуляция».

Алгоритмы и структуры данных

Именно эти вещи вы будете изучать, если решите заниматься на курсах по программированию или поступите в университет, чтобы получить образование в области компьютерных наук.

Алгоритмы – это общие способы решения различных задач информатики / программирования. Например, существует несколько видов алгоритмов, которые обычно используются для программной сортировки списков элементов. Каждый из этих алгоритмов сортировки обладает собственными характеристиками, которые касаются скорости, требований к размеру оперативной памяти и поддерживаемого типа сортируемых данных. В информатике существует множество алгоритмов подобного рода, а кроме того, вы **должны уметь создавать вариации этих алгоритмов** для решения тех сложных проблем, с которыми можете столкнуться в работе над реальными задачами.

Зачастую грамотное применение алгоритмов позволяет одному разработчику решить за час проблему, на решение которой другой разработчик потратит несколько дней. Ес-

ли вы никогда не сталкивались с алгоритмами и совершенно не разбираетесь в их сути, то, работая над какой-либо задачей, **можете даже не подозревать, что на свете давным-давно существует простое и элегантное решение этой проблемы**. По-моему, все сказанное является весомым аргументом в пользу того, что навык работы с алгоритмами стоит потраченного на его освоение времени.

Структуры данных относятся к той же категории, что и алгоритмы, и их изучение тоже важно. Все разработчики ПО должны знать следующие типы структур данных:

- массивы или векторы;
- связанные списки;
- стеки;
- очереди;
- деревья;
- хеши;
- наборы.

Понимая принципы работы структур данных и алгоритмов, вы сможете легко и элегантно решить массу сложных задач, которые обязательно возникнут в вашей работе.

Когда я только начинал программировать, то плохо разбирался в структурах данных и алгоритмах, поскольку был самоучкой. Я не осознавал их ценность, пока не начал соревноваться в решении задачек на сайте TopCoder, где знание этих вещей дает серьезное конкурентное преимущество. Разо-

бравшись с принципами работы этих технологий, я вдруг понял, что те проблемы, над которыми я ломал голову, будучи начинающим разработчиком, перестали вызывать у меня какие-либо сложности. На самом деле, **я считаю, что алгоритмы и структуры данных являются одной из самых увлекательных областей разработки ПО**. Это и вправду очень полезно – в работе над сложной проблемой использовать все эти вещи, чтобы решение было чистым, элегантным и быстро работающим. Лучшим ресурсом для изучения этой темы (на момент написания книги) я считаю книгу Гейл Лакманн Макдауэлл «Карьера программиста»[7]. Данная книга содержит практически всю необходимую информацию об алгоритмах и структурах данных.

Разумеется, изучение этой темы – задачка непростая, однако она стоит потраченных на нее усилий. Это один из тех навыков, которые выгодно «оттенят» вас на фоне коллег. Возможно, вы удивитесь, но **БОЛЬШИНСТВО разработчиков ПО очень плохо подготовлены в этой области**. Если вашей мечтой является устройство на работу в такую крупную компанию, как Microsoft или Google, **без знаний алгоритмов и структур данных вас туда не примут**.

[7] Издавалась на русском языке. Питер, 2021 г. – *Прим. ред.*

Платформы разработки и связанные технологии

Чтобы стать хорошим специалистом, необходим опыт использования хотя бы одной платформы разработки и связанных с ней технологий и фреймворков. Вам, наверное, интересно, что такое «платформа разработки». **В большинстве случаев за этим термином стоит операционная система (ОС),** но он также применим и к другим абстракциям, действующим аналогично операционным системам. К примеру, вы можете быть разработчиком для macOS или Windows и специализироваться на ОС, но точно так же вы можете быть веб-разработчиком, ориентированным на конкретную веб-платформу.

Честно сказать, я не горю желанием сводить эту тему к обсуждению, что же именно представляет собой платформа разработки. У разных людей есть разные мнения на этот счет, но в контексте этой главы **я считаю платформу конкретной средой, в которой создается ПО и которая имеет свои экосистему и особенности**.

Как и в случае с выбором языков программирования, значение здесь имеет не конкретная платформа разработки, а сам **факт ее выбора**. Компании обычно нанимают разработчиков под конкретную платформу или технологию. Если у вас есть опыт в разработке ПО для iOS, то вам будет проще

найти работу именно iOS-разработчиком.

Сказанное означает, что вы должны быть знакомы как с самой платформой, так и с тем, какие инструменты разработки, шаблоны и фреймворки обычно используют разработчики при написании программ для этой платформы. **Может показаться, что язык программирования определяет платформу разработки, но в большинстве случае это не так.** Давайте посмотрим, например, на такой язык программирования, как C#. Зная C#, вы можете писать программы для Windows, macOS, iOS, Android, Linux и даже для встроенных систем каких-либо устройств. И вот вам мой совет: выбирайте не только язык программирования, но и платформу разработки.

Фреймворк или стек

В дополнение к изучению определенного языка программирования и платформы я настоятельно рекомендую изучить соответствующий фреймворк или, что еще лучше, полный стек разработки. Возможно, вы сейчас не поняли ничего из того, что я сказал. Что такое фреймворк? Что такое стек? Что за набор непонятных слов?

Фреймворк – это набор библиотек, которые используются для создания кода на некоторой платформе разработки или на нескольких платформах. Как правило, фреймворк упрощает решение задач, стоящих пе-

ред программистом. Вернемся к примеру с C#. Большинство разработчиков на C# используют для создания приложений фреймворк. NET Framework. Этот фреймворк состоит из множества библиотек и классов, которые позволяют C#-разработчику работать на более высоком уровне абстракции, благодаря чему программисту не приходится «изобретать велосипед» всякий раз, когда потребуется что-то написать.

К примеру,NET Framework содержит код для работы с изображениями. Написать с нуля такой код очень сложно, поэтому данный фреймворк значительно упрощает жизнь разработчикам C#, которые пишут код, манипулирующий изображениями.

Чем фреймворк отличается от стека? **Стек – это набор технологий, обычно включающий в себя фреймворки. Стек используется для создания приложения целиком, а не какой-то его конкретной части**. Например, есть такой стек под названием MEAN. Данная аббревиатура расшифровывается как MongoDB, Express.js, AngularJS и Node.js. MongoDB – это технология баз данных. Express.js – платформа Node.js для создания веб-приложений. AngularJS – JavaScript-фреймворк для создания пользовательских интерфейсов веб-приложений. Наконец, Node.js – среда выполнения для разработки веб-приложений на JavaScript.

Разбираться во всех этих технологиях необязательно (ес-

ли вы, конечно, не MEAN-разработчик). Главное здесь – понимать, что с помощью этих технологий и фреймворков можно создавать веб-приложения целиком – от начала и до конца. **Стеки сильно упрощают создание приложений**, поскольку образуют собой некую единую парадигму, которой может воспользоваться любой разработчик. В частности, приложениями, созданными на одном стеке, делиться будет проще, потому что вы всегда будете знать, что никакой компонент не будет конфликтовать с каким-либо другим.

В качестве еще одного аргумента в пользу стеков скажу, что знание стека означает наличие у вас полного набора навыков для создания приложения целиком. Многие компании, у которых есть приложение, созданное с использованием определенного стека, будут искать разработчиков, которые умеют работать с этим стеком, благодаря чему они смогут очень быстро включиться в работу, не тратя время на изучение используемой в этой компании технологии.

Понимание принципов работы баз данных

Несмотря на то что за последние годы ландшафт баз данных претерпел существенные изменения, мне кажется, что вряд ли эта технология исчезнет в ближайшее время. А раз так, то, наверное, стоит немного понимать, как она работает, не так ли?

На момент создания книги наиболее распространенными были **два вида технологий баз данных**: реляционные и документные. Я придерживаюсь мнения, что разработчик ПО должен быть как минимум знаком с реляционными и немного понимать принципы работы документных. При разработке ПО базы данных часто используются для хранения данных создаваемого приложения.

Некоторые команды имеют в своем составе отдельного специалиста в этой области – разработчика баз данных или администратора баз данных (DBA). Однако данный факт не означает, что вы не должны хотя бы немного разбираться в основных принципах работы баз данных.

Под термином «основные принципы работы» я подразумеваю следующее:

- как работают базы данных;
- как выполнять базовые запросы для получения данных;
- как добавлять, обновлять и удалять данные;
- как объединять наборы данных;

Кроме того, вам, наверное, будет интересно узнать, как **извлекать и добавлять данные с помощью кода** на выбранной вами платформе и/или фреймворке. Зачастую от разработчика ожидается умение писать код, взаимодействующий с БД.

Управление версиями

Управление версиями – одна из важнейших частей процесса разработки ПО. Раньше (до того, как системы управления версиями вошли в повседневный обиход программистов) разработчики попросту создавали общую сетевую папку со всеми файлами проекта или обменивались накопителями с разными версиями ПО на них.

Стыдно признаться, что я и сам занимался подобными вещами. Но я был **молод и глуп, а у вас сейчас есть шанс не наступить на те же грабли.** Сегодня ожидается, что **практически каждый профессиональный разработчик знает, как использовать систему управления версиями для загрузки и скачивания кода из репозитория, а также слияния изменений из нескольких источников**. Управление версиями на самом базовом уровне позволяет вам вести историю изменений различных файлов в общем проекте создаваемого ПО. Это также позволяет нескольким разработчикам одновременно работать над одним и тем же кодом, не мешая друг другу.

Я не буду вдаваться в подробности, лишь подчеркну, что **вы должны уметь пользоваться хотя бы одной системой управления версиями**. А лучше всего будет, если вы разберетесь в принципах работы большинства основных концепций управления версиями. Я очень сомневаюсь, что

в мире существуют профессиональные группы разработчиков, не использующие системы управления версиями.

Сборка и развертывание

Сегодня большинство проектов по разработке ПО используют ту или иную систему сборки[8]. Конечно, существуют команды, которые до сих пор занимаются задачами тестирования и развертывания программы вручную, но это скорее исключение. Давайте сначала разберемся, что такое «система сборки». Позвольте задать вопрос.

Представьте, что вы написали код и даже поместили его в систему управления версиями. **Как понять, что он вообще работает? А то вдруг у вас получилось что-то абсолютно неработающее, способное парализовать работу не только команды, но и всего отдела.**

Именно здесь в игру вступает система сборки. Как минимум она скомпилирует весь ваш код и убедится, что он не содержит ошибок. Как максимум она еще может **запустить модульные или пользовательские тесты, проверить качество кода и предоставить отчет** о текущем состоянии кодовой базы.

Система развертывания отвечает за развертывание кода

[8] Включает такие действия, как компиляция исходного кода в объектный модуль, сборка бинарного кода в исполняемый файл, выполнение тестов, развертывание программы в целевой среде, написание сопроводительной документации или описание изменений новой версии. – *Прим. пер.*

либо в производственной среде, либо в какой-либо тестовой.

Вы не обязаны быть экспертом в этих технологиях, но стоит **понимать хотя бы основы их функционирования**. Часто фактические обязанности по созданию и поддержке систем сборки и развертывания ложатся на плечи сотрудников из быстрорастущей области под названием DevOps («девопсы»). Однако наличие девопсов не освобождает вас от необходимости понимания основных принципов этой области знания.

Тестирование

Раньше разработчикам не нужно было заморачиваться тестированием разрабатываемого ими ПО. **Программисты просто писали код, а затем отправляли его тестировщикам**. Тестировщики искали баги, а программисты потом устраняли найденные ошибки. Именно так раньше выглядел процесс тестирования. Сейчас дела обстоят иначе.

Поскольку во многих проектах по разработке ПО используется так называемая методология Agile (мы обсудим ее чуть позже), сегодня разработчикам ПО и тестировщикам приходится работать в гораздо более плотном взаимодействии. **Качество создаваемого кода стало обязанностью всей команды**. Хотя лично я придерживаюсь мнения, что так было всегда. Отсюда следует, что вам все-таки придется хоть немного, но разбираться в тестировании. Напри-

мер, вас не должны приводить в замешательство следующие слова:

- тестирование методом белого ящика;
- тестирование методом черного ящика;
- модульное тестирование;
- проверка граничных значений;
- автоматизированное тестирование;
- приемочное тестирование.

Хороший разработчик (а я надеюсь, что ваша цель – стать именно таким) всегда тестирует код перед тем, как показать его кому-то еще. Если вы хотите, чтобы к вам относились как к профессионалу, а не как к халтурщику, двух мнений по этому вопросу **быть не может**.

Отладка

Ах, у скольких начинающих разработчиков ПО разбились о скалы отладки их мечты стать крутыми программистами. Все хотят писать код, но никто не хочет заниматься его отладкой. Узнали, согласны?

А теперь серьезно. **90 % вашего рабочего времени в качестве разработчика будет тратиться на поиски ответов на вопрос: «Какого черта этот долбаный код не работает?»** Я знаю, что это совсем не круто. Знаю, что вы хотите каждый день писать новый код. Однако реальный мир жесток, увы.

Если вы будете применять в своей работе такую методологию, как «разработка через тестирование», то, вполне возможно, сумеете сэкономить немало времени на отладке. И все же полностью избежать **отладки своего или чужого кода** вам не удастся.

Поэтому, вместо того чтобы следовать бессистемному подходу в том, что вам так или иначе придется делать, я предлагаю **стиснуть зубы и потратить время на обучение приемам эффективной отладки**. В главе, посвященной данному занятию, мы поговорим об этом более подробно, но я уже сейчас предлагаю вам слегка промочить ножки на берегу моря этой информации.

Методологии

Вы все еще не бросили читать эту книгу, несмотря на всю ту картину, что я перед вами развернул? Чудесно. Обещаю, этот раздел – последний. Честно-честно.

В то время как некоторые команды разработчиков программного обеспечения просто пишут код как Бог на душу положит, **другие используют какую-нибудь методологию. Ну или, по крайней мере, делают вид, что используют**.

(К слову: не ожидайте, что какая-либо команда действительно будет следовать методологии разработки ПО, которая у них используется де-юре. Я не хочу показаться циником

или тыкать в кого-то пальцем. Я просто реалист и знаю, что есть масса людей, которые говорят, что используют методологии разработки ПО, например Scrum, исходя из того, что у них ежедневно проводятся собрания.)

Поэтому очень важно, чтобы вы были знакомы хотя бы с некоторыми из основных идей, лежащих в основе наиболее распространенных методологий разработки ПО. На сегодня таковыми являются **«Водопад»** и **Agile**.

Большинство команд утверждает, что использует Agile (читается как «Аджайл»). Надо сказать, что саму по себе концепцию Agile можно трактовать довольно широко, однако существуют некоторые выработанные на практике приемы использования этой методологии и даже, осмелюсь сказать, ритуалы, о которых следует знать, если вы хотите, так сказать, вписаться в команду «гибких» разработчиков.

Чуть позже мы еще вернемся к этому вопросу. А пока…

Ошеломлены? Спокойствие, только спокойствие

Понимаю, что вы увидели весьма внушительный список того, что нужно знать, чтобы стать разработчиком. **А ведь мы только начали**. Я более чем уверен, что из всего вышенаписанного вы поняли далеко не все, да и вообще пока слабо представляете, с чего следует начинать изучение всех этих штук.

Не переживайте, все так и должно быть. Вам не нужно знать все это прямо сейчас (если, конечно, вы уже не работаете программистом). Ну а если вы и правда являетесь действующим разработчиком и при этом понятия не имеете обо всем вышесказанном, то фу такими быть! Я, конечно, шучу, но **не знать такое и правда стыдно**.

В любом случае, я собираюсь охватить большинство из этих тем более подробно в части «Что нужно знать о разработке ПО». Поэтому можете пока расслабиться.

Далее мы поговорим о том, как получать технические навыки в целом. Поэтому, когда вы дойдете до главы, посвященной этим самым навыкам, этот материал не застанет вас врасплох.

Вопрос Джону!

Слушай, да в этой книге просто гора ссылок! А еще ты рекламируешь кучу своих и чужих продуктов. Что за фигня, чувак?

О, я даже рад, что вы об этом спросили. Давайте сначала поговорим о ссылках. Да, в этой книге их полным-полно, но вы не обязаны переходить по каждой из них. Открывайте лишь те, что интересны лично вам. Эти ссылки ведут на дополнительную информацию по той или иной теме. Я указываю их только потому, чтобы вы могли сильнее погрузиться в заинтересовавшую вас тему.

Честно говоря, большинство ссылок ведут на мои посты в блоге или видео на сайте YouTube, где я

обычно раскрываю какую-либо тему гораздо шире. Ну или просто потому, что я посчитал их чертовски интересными.

(Кстати, если вы откроете ссылку **https://simpleprogrammer.com/products/careerguide/links/**, то попадете на страницу со списком ВСЕХ ссылок в книге, сгруппированных по главам.)

А что касается ссылок на другие мои проекты… Да, вы правы, это действительно реклама. Вы имеете полное право назвать меня меркантильным, но я предпочитаю называть это иначе – «быть умным».

Книги стоят не так уж и дорого. Так что написание собственной книги вряд ли можно считать рецептом обогащения. Скажу больше, если вы пишете книгу, у вас обязательно должна быть еще какая-нибудь мотивация для этого занятия, кроме заработка денег. Одной из причин, по которой я написал эту книгу, является продвижение других моих продуктов, которые, по моему мнению, могут принести вам пользу.

Я не считаю, что ссылок слишком много. И уж тем более я не принуждаю вас покупать хоть что-то по этим ссылкам. В этой книге больше 600 страниц и, по-моему, она ценна уже сама по себе.

Глава 4. Как развить технические навыки

Теперь, когда вы ознакомились со всем этим длинным списком необходимых технических навыков, вам, наверное, интересно, как вообще следует их развивать и сколько времени это займет. Но пусть вас не беспокоит вопрос времени. На всем протяжении карьеры разработчика вы будете постоянно заниматься совершенствованием своих навыков. **Воспринимайте этот перечень как увлекательное путешествие, а не как конечную точку.**

Вам всегда будет куда расти, если захотите. Надо сказать, что лично я потратил немало времени в попытках развить свои технические навыки, идя не в том направлении.

Однако, создав в течение трех лет более пятидесяти узкоспециализированных учебных курсов для программистов, я научился молниеносно развивать технические навыки, попутно обучая им других людей.

Раньше я думал, что лучший способ освоить то или иное техническое умение – это взять большой справочник и прочитать его от корки до корки. О, я прочел просто невообразимое количество книг, в каждой из которых было минимум 800 страниц. Надо сказать, это занятие не принесло мне хоть сколько-нибудь значимой пользы. Разве что мои руки, возможно, подкачались от их веса. Я не хо-

чу, чтобы вы допускали те же ошибки, которые в свое время сделал я. Наоборот, **я хочу показать вам гораздо более эффективный способ освоения новых умений.**

Как научиться быстро учиться?

Перед тем как мы перейдем к изучению технических навыков, давайте немного отвлечемся и поговорим о том, какие бывают способы быстрого обучения чему-либо и о самообучении в целом. В следующей главе этого раздела мы рассмотрим тему самообучения более подробно, а пока я хочу остановиться на основах и поговорить о методологии, которую использую для быстрого изучения чего-либо.

Я уже говорил, что потратил немало часов своей жизни на изучение и преподавание различных технологий. Бывало даже так, что я за несколько недель полностью изучал какой-то язык программирования, после чего делал поворот на 180 градусов и шел его преподавать. В процессе я разработал довольно надежную систему освоения любого необходимого навыка, причем это было не столько сознательное усилие, сколько естественное следствие моих действий. Я пытался учиться так быстро, что мне приходилось придумывать эффективные способы добиваться этого и, естественно, развивать модели обучения, которые помогали мне все быстрее и быстрее осваивать новые умения.

В этой части книги я сделаю лишь краткий обзор основ

быстрого обучения, а если вас интересуют подробности, то в главе «Мой десятишаговый процесс» (а также в некоторых других главах) моей книги «Путь программиста. Человек эпохи IT» вас ожидает целый курс, посвященный этой теме.

Основы

Суть моей идеи довольно проста. Прежде всего вы должны получить представление о том, что вы изучаете, а также оценить масштаб темы. Соберите достаточно информации об интересующем вас объекте, чтобы понять общую картину. Далее вам нужно сузить вопрос до такой области, которую вы легко сможете освоить за адекватный период.

Далее вам необходимо определиться с целью. Здесь я имею в виду, что вы должны понимать, зачем вы учите что-либо, а затем определиться с метриками, по которым будет понятно, выучили вы тему или нет. Многие люди хотят чему-то научиться, но совершенно не понимают, как проверить, что они действительно научились этому.

Найдя отправную точку, приступайте к сбору ресурсов для обучения. Я рекомендую не просто открыть какую-то одну книгу и методично ее штудировать, а собрать несколько источников: другие издания, блоги, подкасты, журналы, видеокурсы и учебные пособия, мнения экспертов и так далее.

После того как у вас наберется несколько подобных ре-

сурсов, создайте с их помощью личный учебный план. Я вижу его как некую систематическую последовательность шагов, которые должны привести вас к определенной заданной цели. Например, вы можете использовать оглавление одной из выбранных вами для чтения книг, чтобы понять, в каком порядке лучше всего изучать вопрос и какие вещи являются наиболее важными. Как только вы определите порядок, в соответствии с которым будете учиться, можете смело считать, что половина работы уже сделана.

Далее приступайте к погружению в тему. Сверяясь с планом, изучите в каждом модуле достаточное количество информации, чтобы запустить процесс обучения, а потом попробуйте «сыграть» самостоятельно. После чего отдельно поработайте с вопросами, которые были непонятны в процессе «игры». Так вы сможете обучаться на практике.

Главное на данном этапе – не поглощать сразу слишком много информации. Играя соло, будьте естественно любопытны, чтобы стимулировать обучение. Затем вернитесь к тексту и прочитайте его или воспользуйтесь материалом по интересующей вас теме, держа в голове вопросы и опираясь на полученный вами опыт, который поможет вам понять, какие вещи действительно важны.

Одной из самых больших проблем, с которой люди сталкиваются в процессе изучения объемной темы, является непонимание того, что важно, а что – нет. «Покрутив» материал и сформировав собственные вопросы, вы поймете, как

решить эту проблему. После этого материал будет усваиваться куда легче, поскольку вам уже будет понятно, на что нужно обращать внимание прежде всего.

На следующем этапе попробуйте научить кого-то тому, чему вы научились сами.

Пока не особо важно, кем будет ваш ученик и в каком формате будет проходить обучение. Вы можете «обучать» хоть свою собаку или белочек в парке, суть сейчас не в этом. Здесь важно то, что в процессе рассказа вам приходится упорядочивать свои мысли таким образом, чтобы их можно было внятно выразить.

Именно на этом этапе изучение переходит в понимание.

В общем, все вышесказанное и есть та самая формула быстрого обучения. Если вы хотите увидеть более обстоятельный пример в компании с учебным пособием и видеороликами, его можно найти на моем сайте. А теперь давайте перейдем к более подробному рассмотрению процесса изучения и развития технических навыков.

Обучение на практике

Я придерживаюсь мнения, что **лучший способ обучения** – это обучение на практике. Когда речь заходит о технических навыках, это утверждение становится не просто моим мнением, а фактически аксиомой. Большинство технических навыков невозможно освоить, лишь прочитав книгу

или посмотрев видео.

Да, книга или видеоролик позволят вам получить представление о том, что возможно сделать с использованием конкретной технологии, языка программирования или инструмента. **Однако пока вы не пощупаете эти технологии руками или не попробуете решить какую-либо проблему, используя новые знания, ваше понимание изучаемого предмета будет оставаться поверхностным**. Практически все навыки, о которых я говорил в предыдущей главе, требуют большего, чем просто книжные знания. Только так вы сможете стать по-настоящему компетентными в выбранной вами области.

Приведу довольно очевидный пример: как вы считаете, можно научиться программированию, читая про синтаксис языка?

Если вы никогда ошибочно не сохраняли файл в неправильную ветку, не работали не с той версией исходного кода и не использовали историю версий, чтобы выяснить, на каком этапе появилась ошибка, то вы не будете знать, как пользоваться системой контроля версий – вам просто будет казаться, что вы разбираетесь в вопросе.

(Если упомянутые термины вам незнакомы, не переживайте, сейчас речь не о них.)

Напомните-ка мне, я случайно не обещал вам рассказать «где-то дальше в книге» про технические навыки? Все же вы читаете эту книгу для того, чтобы чему-то научиться, верно?

Правильный ответ на оба эти вопроса – «да». Однако обучение на этом не заканчивается.

Прочитав все вышесказанное, вы получите общее представление об обсуждаемой нами теме, но рано или поздно вам придется отложить книгу и сделать что-то самостоятельно, чтобы освоить то, о чем вы прочитали (я имею в виду упомянутые мной технические навыки).

Как обучаться на практике?

Сильно рискуя получить почетное звание Капитана Очевидности, я все же решил поговорить с вами о том, как правильно обучаться на практике. Отнеситесь ко всему, что будет далее сказано, как к полезной напоминалке.

Всякий раз, приступая к освоению какого-либо технического навыка, определите для себя, для чего он вам нужен. Если умение не требуется вам «здесь и сейчас», подумайте, а нужен ли вам вообще этот навык? Мы тратим огромное количество времени на изучение технических умений, которые нам никогда не пригодятся. Поверьте, лично я делал это столько раз, что это даже не смешно.

Научиться чему-то гораздо проще, если вы будете испытывать в этом навыке срочную необходимость. Гарантирую, что если вам понадобится освоить прыжки с парашютом, то быстрее всего вы научитесь это делать в самолете, из которого нужно как можно быстрее эвакуироваться.

Но как быть, если прямо сейчас у вас нет острой необходимости в выбранном вами навыке? Что делать, если вы изучаете что-то просто потому, что хотите получить работу, где вам, скорее всего, придется использовать это умение? Ответ прост – найдите причину использовать выбранный вами навык здесь и сейчас. Создайте себе цель.

Пример обучения на практике

Давайте разберем вопрос на примерах. Представьте, что вы очень сильно хотите разобраться в реляционных базах данных. Для этого вы можете прочесть книгу, посвященную данной теме, поупражняться в искусстве конструирования SQL-запросов и в результате чему-нибудь научиться.

А теперь представьте, что ваша цель заключается в создании базы данных всех фильмов, имеющихся в вашей коллекции. А также представьте, что вы хотите уметь делать запросы к этой базе, добавлять в нее новые фильмы, удалять старые, обновлять названия и т. д. А если вы захотите создать приложение, чтобы с его помощью упростить работу с базой данных?

Представили? Поздравляю, у вас появилась цель! Так, и что дальше? Как подойти к изучению темы реляционных баз данных? А вот теперь вы можете смело открывать книгу по теме или просмотреть соответствующие видеоуроки, чтобы найти информацию, необходимую для реализации вашей

идеи. И вот теперь вы будете заняты созданием реального продукта, а не просто выполнением какого-то абстрактного тестового задания.

Только представьте, как много вы всего сможете узнать, руководствуясь подобным подходом. Ведь это куда веселее, чем выполнение всяких непонятных лабораторок?

Как я обучаю техническим навыкам

Как я уже говорил, я помог с освоением технических навыков довольно большому количеству людей. Возможно, если вы узнаете, как я это делаю, то сможете и сами научиться этому. Как думаете, я прав?

Когда я обучаю людей чему-то, я хочу, чтобы затраченные ими средства окупились в полной мере. Я не хочу загружать их вещами, которые им вряд ли пригодятся, или задачами, которые они вполне могут решить сами, если столкнутся с ними.

Я стараюсь фокусироваться на том, чтобы дать людям информацию, которая будет полезна здесь и сейчас, а не «через сто лет» и «на всякий случай». Моя программа обучения техническим навыкам основывается на трех столпах.

- Как ученики могут использовать эту технологию?
- С чего начать?
- 20 % информации, знания которой достаточно, чтобы добиться максимальной эффективности.

Разберем каждый из этих пунктов подробнее.

Как можно использовать технологию?

Я считаю, что Google может дать ответ на любой вопрос. Но получить ответ не удастся, если вы не знаете, что спрашивать. Поэтому обучение я начинаю с рассказа о конкретной технологии и описания ее возможностей.

Разумеется, это всего лишь обзор. Я не пытаюсь показать сразу все способы использования новых знаний, а просто рассказываю о наиболее интересных моментах изучаемой темы.

Когда заходит речь о языках программирования, я всегда начинаю с истории языка и описания типичных областей его применения. Затем я перехожу к описанию всех конструкций языка (делая акцент на уникальных). Следующий этап – знакомство с библиотеками как стандартной части языка и описание способов их использования.

Целью такого подхода является поверхностное описание темы, без углубления в детали. Если вас заинтересует что-то конкретное, вы можете узнать об этом самостоятельно. На этом этапе я пытаюсь сделать неизвестное известным касаемо общих понятий. Я хочу, чтобы ученики поняли, что именно они не понимают и могли закрыть этот пробел самостоятельно, если в этом будет нужда.

Я стараюсь изложить тему так, чтобы после обучения вы

говорили не «Я не знаю, может ли технология X решить эту проблему», а «Я знаю, что технология X может решить эту проблему, но пока слабо представляю как. Если мне понадобится узнать ответ на этот вопрос, я смогу его найти».

Представьте, что вы пытаетесь изучить деревообработку, не зная, что существует такая вещь, как дремель или фрезерный станок. Вы не обязаны знать, как пользоваться этими штуками, но если вы даже не подозреваете об их существовании, то вы ощутимо ограничиваете себя в возможностях.

Как начать?

На следующем этапе я пытаюсь научить своих студентов умению приступать к изучению чего-либо. Зачастую эта часть освоения технологии является наиболее сложной – она предшествует «действию», поэтому я стараюсь сделать данный процесс как можно более безболезненным. Моя задача состоит в том, чтобы показать студентам, как загрузить и установить все необходимое для работы, а затем создать проект и скомпилировать свой первый код.

Как только человек преодолевает это препятствие, он может приступать к экспериментам, пытаясь создать что-то конкретное в процессе работы с технологией. Если входной порог оказывается слишком высоким, некоторые люди так и остаются на уровне чтения книг о технологии и просмотра обучающих видео, не решаясь «запачкать руки».

Что бы вы ни изучали, держите в уме эту информацию. Сосредоточьтесь на способах того, как начать работу с определенным техническим навыком на ранних этапах процесса обучения. Поищите соответствующие учебные пособия или руководства, которые рассказывают о том, как начать работу и сдвинуться с мертвой точки.

20 % информации для максимальной эффективности

Напоследок я пытаюсь научить студентов тем 20 % информации, которые они будут использовать 80 % времени. Практически все в нашей жизни подчиняется так называемому принципу Парето. Он звучит следующим образом:

«20 % усилий дают 80 % результата, а остальные 80 % усилий – лишь 20 % результата».

Освоение технического навыка напрямую зависит от понимания, что именно входит в эти 20 %, которые вы будете использовать 80 % времени.

На этом этапе практика играет более значимую роль, чем теория. Большинство книг и учебных пособий имеют формат справочников, которые не делают акцент на 20 % наиболее важных технологий. Когда вы активно работаете с технологией, то быстро понимаете, какая ее часть используется чаще всего, поскольку нехватка знаний в этой области ощутимо скажется на вашей работе.

Предлагаю вернуться к примеру с реляционными базами данных. Написание запросов для выбора данных из таблиц совершенно точно входит в 20 % наиболее востребованных в данной технологии умений. Если вы прочитаете книгу, посвященную языку SQL, то увидите, что темам получения данных, их добавления, обновления, удаления, а также индексации уделяется одинаковое количество внимания. Но если вы попытаетесь создать собственную базу данных, то наверняка заметите огромное количество запросов, связанных с выбором данных. Помимо прочего, вы очень быстро встретитесь с необходимостью объединения таблиц.

Вместо того чтобы тратить время на изучение всех тонкостей теории реляционных баз данных, гораздо эффективнее сосредоточиться на том, чтобы научиться писать операторы выбора, объединять таблицы и выполнять другие операции общего характера, составляющие заветные 20 % наиболее востребованных умений.

Вот почему практика столь важна.

Бывает полезно понаблюдать за работой эксперта в интересующей вас области, или даже предложить ему посотрудничать с вами в качестве ученика. Видя, как мастер применяет те навыки, которым вы хотите научиться, вы сможете понять его 20 % и ускорить процесс своего роста в профессии. Обучение в процессе работы – чрезвычайно эффективный способ развития себя как специалиста.

Читайте, что пишут эксперты

Напоследок я хочу дать вам еще один совет по развитию технических навыков. **Уделяйте побольше внимания чтению экспертов, которые уже обладают нужными вам техническими навыками.** Когда я изучал свое ремесло, я тратил около 30 минут в день на чтение разных блогов, связанных с интересующими меня темами. Когда мне потребовалось вникнуть в суть тонкостей языка C++, я глотал книги Скотта Майерса по эффективному программированию на C++. Зачастую мнение эксперта помогает разобраться в каком-либо вопросе, до ответа на который додуматься самостоятельно было бы невозможно.

Одно дело понять синтаксис языка программирования или принципы использования фреймворка; и совсем другое – овладеть ими на первоклассном уровне. Присмотритесь к тому, как эксперты применяют на практике те навыки, которые вы только пытаетесь приобрести. Почитайте, о каких проблемах и тонкостях упоминают признанные профи в данной области. Подобная информация способствует более качественному освоению темы.

Практика, практика и еще раз практика

Надеюсь, мои советы окажутся полезными и помогут вам

получать и развивать необходимые умения. По-моему, польза обучения на практике очевидна, особенно в области технических навыков.

Я также надеюсь, что вы осознаете необходимость наличия плана обучения и его четкой цели.

Скажу напоследок лишь одно: практикуйтесь.

Оттачивание любых технических навыков требует времени. Чтобы в чем-то преуспеть, придется много тренироваться. Не расстраивайтесь, если вам кажется, что обучение займет обескураживающе большое количество времени, особенно в ситуациях, когда вы застопорились. Навыки придут, если у вас есть план и четкая цель. Продолжайте в том же духе и просто доверьтесь процессу.

Глава 5. Как выбрать язык программирования

Самый популярный вопрос, который задают мне совсем-совсем новички: «Какой язык программирования выбрать?» Для некоторых это становится камнем преткновения, который им так и не удается преодолеть.

В своей преподавательской практике я встречал множество разработчиков, которые были не уверены в себе и постоянно меняли свой выбор, прыгая от одного языка программирования к другому из-за боязни принять неправильное решение. Если вы узнали в этом описании себя, то эта глава для вас.

В ней я сначала развею часть ваших сомнений, а затем поделюсь парой идей по выбору первого языка программирования, основанных на практике.

Выбор языка практически ни на что не влияет

Да-да, вы все правильно прочитали. Как ни странно, но не имеет особого значения, какой язык программирования вы выберете. Есть несколько причин, почему я так считаю, и одна из них – многие языки программирования по своей су-

ти очень похожи. Да, синтаксис может разниться. Да, языки программирования могут выглядеть по-разному. У них даже могут быть совершенно разные наборы конструктивных особенностей. Но фактически у всех языков программирования между собой куда больше общего, чем вы могли бы подумать.

Практически в каждом из них присутствуют конструкции ветвления, циклы, возможность создания методов или процедур, а также способы организации кода на высоком уровне абстракции. Существует огромное количество языков программирования, которые настолько похожи, что если вы знаете один их них, то почти знаете и другой. Примерами такого рода являются C# и Java. JavaScript схож с каждым из них.

Самое сложное – выучить первый язык программирования. Как только вы это сделаете, освоить другой язык будет гораздо проще. После изучения двух или более языков программирования каждый последующий будет даваться вам все проще и проще. Если сейчас вы не знаете ни одного языка, мое утверждение может показаться вам легковесным, однако доверьтесь моему опыту. Как я уже говорил, многие языки программирования похожи, и поэтому вам будет не очень сложно переключаться между ними. То есть даже если вы выучите язык программирования, а затем решите, что это не то, что вам нужно, или получите работу, на которой требуется использовать другой язык программирования, это не будет иметь большого значения. Самое трудное – тяжелая

работа по изучению первого языка – уже позади.

Если вы посмотрите на вакансии таких крупных компаний, как Microsoft или Google, то увидите, что большинство из них не содержит требований к языку программирования. У меня была масса собеседований, на которых меня просили решить задачу по программированию на любом удобном мне языке. Никаких ограничений, никакого конкретного языка программирования, который я должен был бы знать.

Что следует учесть при выборе языка программирования?

Как уже было сказано выше, я не думаю, что это принципиально важно. Однако если вы все равно ощущаете неуверенность, вот вам несколько советов.

Карьерные перспективы

Скажу так – для большинства из вас наиболее важным аспектом является то, какую работу вам поможет получить тот или иной язык программирования и каковы его перспективы. На данный момент по каждому из популярных языков программирования можно найти массу вакансий. Рейтинг того или иного языка может со временем измениться в любую сторону, но, если вас беспокоит наличие вакансий, рекомендую сделать выбор в пользу наиболее популярных язы-

ков программирования. На момент написания данной книги такими технологиями являются следующие:

- C#
- Java
- Python
- Ruby
- JavaScript
- C++
- PHP

Вакансии для разработчиков на этих языках есть всегда.

Однако, выбирая язык, следует учитывать особенности рынка труда той страны, где вы живете, особенно если вы не планируете переезжать.

Например, если вы живете в небольшом городке в Аркансасе, в котором есть только одна технологическая компания, делающая все на Java, я бы посоветовал вам начать учить именно этот язык. Конечно, далеко не каждый из вас столкнется с подобной ситуацией. Но если кто-то из моих читателей все-таки в ней окажется, то выбор будет невелик.

Если же вы планируете переезжать или работать на фрилансе, можете попробовать освоить более эксцентричный и менее популярный язык. Так у вас будет больше шансов стать экспертом в области, где не так много профи. Но для начала я все же порекомендовал бы вам обратиться к чему-то более популярному.

Наряду с популярностью языка следует учитывать и его перспективы.

На момент написания этой книги язык Objective-C вряд ли можно назвать подходящим вариантом, так как сейчас большинство разработчиков под iOS переходят на язык Swift. Кроме того, сама Apple вкладывает большие средства в развитие этого языка.

Но если вы уже выучили Objective-C и теперь не знаете, как быть дальше, спешу вас успокоить. Рабочих мест, связанных с данным языком, довольно много. Существует множество приложений, написанных на нем, которые нуждаются в поддержке.

Тем не менее никто из нас не является провидцем и не может знать наверняка, какие языки будут популярны, а какие – нет.

Например, некоторое время назад я предсказывал смерть JavaScript. Однако он жив и поныне.

Недавно я посетил одну конференцию, где одним из докладчиков был соавтор языка Objective-C. Этот язык появился в начале 1980-х годов.

О чем это я? Ах да, о выступавшем. Так вот, данный человек по имени Том Лав является автором книги, в которой написано, что язык JavaScript мертв и никогда и нигде использоваться не будет. Однако на момент написания этой книги JavaScript входит в пятерку (а по некоторым данным, и в тройку) самых популярных языков программирования в

мире.

Суть в том, **что знать наверняка невозможно.** Языку программирования Ruby понадобились годы, чтобы стать популярным.

Возможно, JavaScript – это один из худших языков программирования, который можно было придумать и который поначалу использовался лишь для создания небольших всплывающих окон на веб-страницах. А сегодня на нем пишут все подряд.

Мой вам совет – не гадайте. Если, конечно, у вас нет магического шара. Но если он у вас есть, то зачем вам вообще программирование? Отправляйтесь прямиком на Уолл-стрит, там вас ожидает головокружительная карьера!

Вопрос Джону!

За что ты не любишь JavaScript? Крутой же язык!

Каюсь, иногда мое мнение о JavaScript выглядит так, как будто у меня на него большой зуб или он нанес мне глубокую психологическую травму в детстве.

Позвольте рассказать вам небольшую историю о том, как появилась эта технология. В мае 1995 года Брендан Эйк, который в то время работал в компании Netscape, создал за 10 дней некую штуку под названием JavaScript, стремясь получить простой «связующий язык», который был бы легок для освоения web-дизайнерами и программистами, работающими неполный рабочий день. (Цитата из «Википедии».) В

общем, я хочу сказать, что JavaScript, будучи созданным всего за 10 дней, изначально не мог быть хорошо продуман. Это факт. Знаю, что никто не любит гонцов с плохими вестями, но давайте обойдемся без насилия. Хочу подчеркнуть, что я **НЕ НЕНАВИЖУ** JavaScript. Просто для меня этот язык не является верхом элегантности, и поэтому я предпочитаю его не использовать. Вот и все.

Справедливости ради следует отметить, что в 2015 году появился новый стандарт этого языка (известный как ECMAScript). При создании этой версии языка ее авторы учли многие недостатки JavaScript.

Стыдно сказать, но эта версия мне даже нравится… Но лишь слегка!

Тем не менее мое мнение здесь мало что значит, ведь JavaScript сейчас – один из наиболее популярных языков программирования. Хотите верьте, хотите нет, но я смирился с этим фактом, однако оставил за собой право прохаживаться при случае по нелюбимому мною предмету.

Ну а если вы все еще сомневаетесь в моей правоте, подумайте, почему одна из самых продаваемых книг, посвященных этому языку, называется «JavaScript: сильные стороны»[9]?

[9] Речь идет о книге Дугласа Крокфорда. Издавалась на русском языке. Питер, 2013 г. – *Прим. ред.*

Технологии, которые вам интересны

Выбирая язык программирования, важно учитывать свои предпочтения в технологиях. Если вы начнете поиск, отталкиваясь от этого фактора, то сможете гораздо быстрее определиться с выбором. Я знаком с большим количеством Android-разработчиков, которые занимаются этим только из любви к Android. Для многих из них оптимальным выбором будет Java, потому что именно на этом языке разрабатывается большинство приложений для данной ОС. (Однако вы можете создавать Android-приложения и на других языках, например C#, Ruby и даже JavaScript.)

В общем, слушайте свое сердце, это упростит вам изучение вашего первого языка программирования. Чем больше вас привлекает изучаемая технология и чем больше вы ею увлечены, тем легче вам будет преодолевать виражи учебного процесса. Например, я очень хотел заниматься разработкой под iOS, потому что у меня появился мой первый iPhone, и я был в восторге от этой технологии. Эта воодушевленность значительно облегчила мне изучение Objective-C и создание моего первого приложения под iOS.

Будь я равнодушен к этой технологии, вряд ли я продвинулся бы в ней достаточно далеко.

Уровень сложности

Еще одним важным моментом, который следует учитывать при выборе языка программирования, является уровень сложности.

Некоторые языки программирования более сложны в освоении, чем другие. Я не рекомендую начинать изучение программирования с C++, потому что по сравнению с другими языками программирования он довольно сложен для понимания. В этом случае вы будете иметь дело с управлением памятью и указателями, а также с некоторыми другими неприятными конструкциями, которые могут вызвать у новичка настоящий ступор.

C++ это великий язык – один из моих любимых, но его изучение тот еще квест. Гораздо проще будет начать знакомство с программированием с таких языков, как C#, Lua, Python, Ruby или PHP. Более того, для начинающих программистов существуют даже специальные языки, такие как Scratch или Basic.

Я не хочу отговаривать вас от изучения более сложного языка, такого как C++, если вы действительно этого хотите. Однако следует заранее представлять, во что вы ввязываетесь, и принять осознанное решение, насколько легким для изучения должен быть ваш первый язык.

Доступные ресурсы

Оцените ресурсы, которые вам пригодятся при изучении выбранного языка программирования. Например, в случае некоторых малоизвестных языков вы можете столкнуться с недостатком учебных материалов по ним – книг, онлайн-видео или любых других ресурсов – все это может существенно осложнить процесс освоения языка.

Популярные языки программирования хороши тем, что вам будет доступна масса онлайн-руководств, учебных курсов, книг и других материалов, поэтому перед началом обучения обязательно изучите этот вопрос. Несмотря на то, что сейчас существует огромное количество разнообразных ресурсов для начинающих, стоит заранее учесть степень доступности обучающих руководств.

Также рекомендую оценить, какими ресурсами располагаете конкретно вы, например компьютером или программным обеспечением. Даже более сложный язык программирования будет проще изучить, если у вас под рукой есть много интерактивных онлайн-учебников.

Изучать JavaScript можно онлайн через браузер, без установки чего-либо на компьютер. В свою очередь такой язык, как C ++, потребует для своей работы определенных инструментов и программного обеспечения, получить которые может быть непросто.

Наконец, подумайте о людях, которых вы знаете. Есть ли среди них те, к кому вы могли бы обратиться за советом в случае трудностей?

Не зацикливайтесь на вопросе ресурсов, но и забывать о них тоже не стоит.

Адаптивность

Напоследок давайте поговорим о такой вещи, как адаптивность. Языки программирования по-разному адаптируются к исходным условиям. Например, на момент написания этой книги язык программирования C# можно с уверенностью назвать одним из наиболее адаптируемых благодаря таким компаниям, как Microsoft и Xamarin (теперь входит в состав Microsoft).

Изучая C#, вы вовсе не будете ограничены лишь разработкой под Windows или web-программированием. Этот язык может использоваться для создания приложений под любую из доступных платформ.

С помощью C# можно создавать приложения для Linux и Mac, и даже для Android и iOS. Многие другие языки программирования обладают сходной гибкостью.

Например, Ruby был перенесен на множество различных платформ и используется во многих технологических областях. JavaScript также легко адаптируется. Вы даже можете использовать JavaScript для управления платами Arduino и

робототехники. (Мой хороший друг Дерик Бейли написал неплохую статью на этот счет.)

Некоторые другие языки программирования менее гибки. Например, если вы изучаете R или Go, вы будете чуть более ограничены технологиями и платформами, под которые можно разрабатывать программы на этих языках. Все больше языков программирования, особенно популярных, переносятся на новые платформы и используются во множестве различных технологий. При этом часть языков по-прежнему остаются не столь универсальными.

Поэтому, если вы хотите иметь возможность быть сегодня веб-разработчиком, а завтра писать программы для Android, или попросту работать с более чем одной платформой или технологией, выясните, насколько адаптивен выбранный вами для изучения язык.

Пара мыслей напоследок

Несмотря на все высказанные мной соображения, которые не следует упускать из виду при выборе языка, я хочу подчеркнуть, что итоговый выбор технологии не так уж и важен.

Гораздо важнее выбрать что-то одно и придерживаться своего решения столько времени, сколько требуется для прохождения кривой обучения, необходимой для достижения мастерства. Многие начинающие программисты бросают дело на полпути, потому что им кажется, что они ничего не по-

нимают. Мы еще вернемся к этой теме, в главе «Ваш первый язык программирования».

Главное – не сдаваться! Уверяю вас, что все получится. В какой-то момент вы можете ощутить скуку или решить, что надо было выбрать другой язык и затем начать перебирать разные технологии, но поверьте мне – это будет плохая идея. Когда я начинал изучать программирование, одним из самых важных навыков для разработчика было глубокое понимание языка. Я просматривал массу книг по C++, пытаясь узнать обо всех тонкостях языка. Сегодня в этом занятии нет необходимости. В наши дни программисты работают на более высоком уровне, что предполагает более широкое использование библиотек и фреймворков, нежели функций самого языка. Разумеется, вы по-прежнему должны уметь программировать, но тончайшее владение каким-либо языком сейчас излишне.

Именно по этой причине я рекомендую вам не тратить слишком много времени на выбор языка. Просто начните учить какой-то один и не меняйте своего решения.

Глава 6. Ваш первый язык программирования

Итак, выбор сделан. Что дальше? Конечно же, открыть книгу и начать читать! Прекрасный выбор, который с большой долей вероятности заставит вас отказаться от программирования раз и навсегда.

Помните, я говорил об обучении на практике? Именно этим мы и займемся в этой главе.

В ней вы найдете план, следуя которому сможете проще и качественнее изучить первый язык. И не просто изучить, а почувствовать себя в нем комфортно, и даже стать в нем профи. Возможно, что освоение первого языка окажется наиболее сложным этапом освоения программирования в целом, но это не обязательно так.

Большинство программистов (как и я сам в начале пути) учились так:

1) начинали штудировать учебник;

2) пытались что-то из него повторить;

3) это что-то не желало работать;

4) вновь и вновь перечитывали материал, пока не удавалось разобраться в проблеме;

5) PROFIT!

То, чем я собираюсь поделиться с вами, получено в результате обучения многих разработчиков программного

обеспечения не только в процессе изучения их первого языка программирования, но и улучшения их навыков разработки на этом языке. При создании этой главы я также руководствовался собственным опытом освоения таких языков, как C ++, C # и Java.

Фактически в этой главе будет описана последовательность действий, которой я придерживался бы при изучении своего первого языка программирования, если бы знал то, что знаю сейчас.

Посмотрите, как функционирует уже работающее приложение

Как я уже говорил, большинство начинающих программистов, желая освоить какую-то технологию, просто открывают соответствующий учебник и начинают его читать. В мире существует несколько прекрасных книг, которые ставят своей целью научить вас программировать через практику. Однако я придерживаюсь мнения, что лучше всего начинать с изучения исходного кода уже написанного и точно работающего приложения, пытаясь понять, как работает программа.

Да, это сложно. Да, сначала все будет непонятно и некомфортно. Но советую привыкнуть к этому ощущению.

Я предлагаю вам взять любое популярное приложение с открытым кодом и изучить его. Для поиска объекта исследования воспользуйтесь сайтом GitHub. Отдельно рекомендую

вам научиться скачивать код, собирать и запускать его самостоятельно. Будет очень круто, если в это время с вами сможет побыть опытный программист и помочь вам в наиболее трудных моментах! Но если таких знакомых у вас нет – ничего страшного.

Одинаково важно изучить листинг и для ознакомления с синтаксисом языка, и чтобы увидеть, сможете ли вы понять какие-нибудь программные инструкции или осмыслить происходящее в коде. В процессе изучения кода запускайте приложение на выполнение, чтобы ощутить связь поведения программы с ее листингом.

Вы совершенно точно будете ощущать дискомфорт. Скорее всего, вы даже почувствуете, что не понимаете ровным счетом ничего.

Как я уже говорил – это нормально. Если вы сможете понять хотя бы пару строчек, это будет очень круто. А если вам удастся изменить пару параметров так, что код запустится, а в приложении что-то изменится, то вы – огромный молодец!

Изучая программирование подобным способом, вы сможете понять, как выглядит структура хороших приложений. Представьте себя археологом, изучающим письменность древних цивилизаций. Благодаря этому методу вы окажетесь на несколько шагов впереди других начинающих программистов. Потому что в отличие от них, вы будете знать, как выглядит тот язык, на котором вы учитесь писать.

Перед путешествием всегда полезно изучить местность. В

программировании дела обстоят точно так же.

Просмотрите пару качественных материалов по теме

Продолжая тему изучения местности, в качестве следующего шага я рекомендую вам пробежаться по тексту пары книг глазами, – именно пробежаться, а не читать их от корки до корки.

Суть данного этапа заключается в том, чтобы понять, какие в изучаемой области имеются основные темы и понятия. Затраченные вами усилия обязательно окупятся, когда вы почувствуете, что поняли, какие вещи вы будете изучать и как они соотносятся друг с другом.

Программка Hello World

На этом шаге вы тоже не будете читать книги или осваивать какую-то учебную программу. Вскоре вы этим займетесь – но только если захотите. (На самом деле выучить язык программирования можно и без книг, если следовать действиям, описанным в этой главе. Например, я буквально за пару недель сумел изучить языки Go и Dart, пользуясь лишь онлайн-документацией и следуя процессу, аналогичному тому, что мы здесь обсуждаем.)

Итак, сейчас ваша задача – создать самую простую про-

грамму, какую только можно написать на изучаемом вами языке. Помните, в главе «Как развить технические навыки?» я говорил об уровне, достаточном для старта обучения? Это и есть ваша текущая цель.

Чтобы почувствовать уверенность в своих силах и получить практические знания, вы должны начать программировать.

Поэтому мы начнем с программы Hello World. Большинство книг по программированию начинается именно с нее. Все, что она делает, – выводит на экран надпись Hello World. На данном этапе от вас практически не требуется разбираться в языке. Вам просто нужно ознакомиться с базовой цепочкой инструментов, необходимых для создания и запуска программы на выбранном языке программирования.

В учебнике, посвященном вашему языку программирования, наверняка есть текст этой программы. Если это не так, сделайте в Google запрос «Hello World + твой язык программирования». В Google есть все. В процессе создания программы Hello World вы поймете, как выглядит базовая структура программ на этом языке программирования.

Изучите базовые конструкции и попрактикуйтесь на реальных задачах

Ура! Вы добрались до этапа, когда можете открыть книжку или приступить к прохождению учебного курса! Сейчас

вам будет определенно легче воспринимать материал, чем если бы вы сразу погрузились в него.

Ознакомьтесь с основными конструкциями языка, а затем попробуйте создать с их помощью несколько программ. Подумайте, какие реальные задачи мог бы решать ваш код, чтобы вы могли как можно лучше закрепить навык. Далее приведен список основных базовых конструкций любого языка программирования.

- Возможность вывода на экран.
- Базовые математические действия.
- Хранение информации в виде переменной.
- Организация кода с помощью функций, методов или модулей.
- Вызов функции или метода.
- Булева алгебра.
- Операторы if/else.
- Циклы.

Хорошая новость: если все перечисленное вам уже знакомо, считайте, что вы уже умеете базово программировать на любом языке. Да, в каждом случае синтаксис будет отличаться, но суть останется прежней. Данный этап займет больше времени, чем предыдущий. Изучите каждую из упомянутых конструкций, а затем поработайте с ней на практике.

В случае, если вы учите язык самостоятельно, постарайтесь понять, какие конструкции имеются в языке и в каком порядке их следует изучать. Если вы работаете по книге (а

лучше нескольким), в ней должен быть прочерчен путь обучения, а также должны приводиться примеры и задачки различной сложности. Старайтесь всегда понимать, для чего вы учите ту или иную тему и где можно использовать полученные знания.

Завершив этот этап, вернитесь к программе, исходный код которой вы изучали в самом начале, и оцените, насколько понятнее она теперь для вас выглядит.

Усвойте разницу между функциями языка и библиотеками

Есть одна вещь, которая часто сбивает с толку начинающих программистов (особенно учитывая уровень развития нынешних языков программирования) – знание того, что является частью языка, а что – частью стандартных библиотек, которые поставляются вместе с языком. Во многих случаях эта разница будет практически незаметна, поскольку вы очень часто станете использовать библиотечные функции.

Однако, как я уже сказал, знать, где находится грань между библиотекой и самим языком, необходимо.

Вы можете считать меня формалистом, однако именно эта придирка поможет вам расставить по полочкам всю ту мешанину, которая наверняка осталась у вас в голове после всех предыдущих этапов. Разобравшись в этой теме, вы будете знать, что на самом деле большинство языков программи-

рования куда проще и меньше, чем библиотеки. Подчеркну, что изучение библиотек более сложное дело, чем изучение языка программирования.

Дело в том, что программирование сегодня – это скорее умение использовать библиотеки и фреймворки, нежели доскональное знание языка. Именно поэтому я так много внимания уделяю важности понимания границы между языком и его библиотеками.

Это понимание позволит вам стать более продвинутым разработчиком. Зная, как найти библиотеку, которая поможет в решении той или иной задачи, вы будете создавать более качественный код, нежели любой «эксперт в языке».

Изучите имеющийся код и разберитесь в каждой строке

Если вы добрались до этого этапа, то вы уже должны быть знакомы со всеми основными концепциями изучаемого языка программирования, и должны иметь опыт применения большинства из них при решении реальных задач. Также вы должны понимать разницу между языком и его библиотеками.

На этом шаге вы все еще можете ощущать неуверенность в своих силах и не чувствовать, что разбираетесь в языке. Вероятно, вы уже понимаете, как работает каждая конструкция, но не имеете ни малейшего представления, как из это-

го всего слепить рабочую программу. Этот процесс можно сравнить с изучением иностранного языка с нуля.

На этом шаге многих начинающих программистов одолевает отчаяние, и они начинают думать, что им никогда не стать настоящими разработчиками. Одним из лучших способов продвинуться вперед и понять, где в ваших знаниях есть пробелы, это заняться изучением уже существующих программ, строка за строкой, добиваясь полного понимания их работы. (Да, вам не всегда будет очевидно, почему это работает именно так, но понимать, как это работает, важнее.)

Вопрос Джону!

Ты точно уверен, что мне не обязательно знать, почему код работает именно так? В чем прикол понимать «как», не понимая «почему»?

Посмотрите на это так: если вы не понимаете, *как* что-то работает, то вряд ли поймете, *почему* оно работает. Нельзя понять смысл предложения, не понимая смысла отдельных его слов. А если вы не понимаете смысл предложений, то не поймете и смысл текста. Именно поэтому мы начинаем с нижнего уровня.

Я хочу быть уверен, что вы понимаете каждую строку, каждое выражение в любом коде. В противном случае вы не сможете понять, как все это складывается в одну большую, правильно функционирующую программу.

А теперь давайте сфокусируемся на изучении языка,

и все остальное приложится, обещаю.

В качестве следующего шага вы можете обратиться к коду программы из предыдущих пунктов и просто начать перебирать файлы проекта. Откройте любой из них и начните изучать построчно. Если вы можете с уверенностью сказать, что делает каждая строчка, идем дальше! Если же вы понимаете не все – а такого, скорее всего, будет очень много, – берите в руки учебник и попытайтесь уменьшить количество белых пятен.

Согласен, это довольно утомительно. И да, весьма скучно. Но, поверьте мне, это окупит себя с лихвой.

Как только вы сможете уверенно сказать, как работает каждая строчка кода, – пусть и без понимания, почему именно так, а не иначе, – вы готовы двигаться дальше.

Создайте ЧТО-ТО. А потом еще раз, и еще

Настало время начать по-настоящему использовать язык программирования! К этому моменту вы уже должны были написать несколько небольших программ и опробовать большинство функций языка. Но только создание реальных приложений позволит вам овладеть технологией более качественно.

Не следует браться за что-то амбициозное или с наворо-

ченным пользовательским интерфейсом. Рекомендую ограничиться приложениями, которые умеют принимать только ввод данных с клавиатуры и выводить информацию на экран. Главное сейчас – научиться создавать простые программы, ориентированные на использование изучаемого языка программирования и стандартных библиотек, а не на использование дополнительных фреймворков – к ним мы перейдем позже.

Предлагаю несколько идей для программы.

• Программа, которая решает математическую задачу на основе данных пользователя.

• Программа типа «Выбери свое приключение», в которой ход программы определяют действия пользователя.

• Простенькая текстовая приключенческая игра, в которой пользователь может отдавать команды, чтобы поднимать предметы, перемещаться по комнатам и т. д.

• Программа, которая будет читать ввод из текстового файла и записывать вывод в другой текстовый файл.

• Чат-бот, который разговаривает с пользователем и притворяется человеком или дает юмористические ответы.

Используйте конкретную технологию или платформу

Раньше вы должны были в основном изучать и использовать выбранный вами язык программирования изолирован-

но. Так и задумано, потому что сначала вам нужно научиться понимать сам язык программирования и его стандартные библиотеки и почувствовать себя комфортно. И только потом добавлять какую-то среду и другие фреймворки для создания реального приложения.

Чтобы создать что-то полезное с помощью языка программирования, вам нужно будет применить его к определенной технологии или платформе.

Пришло время взяться за несколько небольших проектов, для которых потребуется использование языка программирования на определенной платформе.

Давайте предположим, что вы изучаете Java.

До сих пор вы писали код на Java, который сработает на любой платформе, где запускается Java, поскольку вы в основном использовали стандартные библиотеки и просто работали с вводом и выводом на экран или в файл. Теперь вы, например, решаете использовать Java для создания приложения для Android. Вам нужно будет узнать, как создавать приложения для Android, и о фреймворке Android. Однако вы уже знакомы с Java, поэтому вам не придется учить огромное количество концепций сразу.

Вы, конечно, можете изучать Android и Java вместе – на самом деле я создал курс, где именно этому и учил, – но если вам хочется по-настоящему овладеть языком и избежать путаницы, то разучить язык отдельно от платформы или технологии, а затем объединить их, вероятно, будет намного про-

ще.

Теперь вам пора развивать конкретные, специализированные навыки работы с изучаемым вами языком программирования, которые будут полезны для получения работы.

Выберите любую платформу или технологию, с которой, как вам кажется, вы хотите работать в будущем, и начните создавать на ней небольшие приложения. Пока я рекомендую вам сосредоточиться только на одной технологии или платформе. Вы всегда сможете добавить еще одну позже.

Так вы не только ограничите то, чему вам предстоит научиться на данном этапе, но и приобретете более глубокие знания и компетентность в конкретной технологии. Это поможет вам стать гораздо увереннее в себе и значительно повысит востребованность ваших навыков.

Решайте сложные алгоритмические задачи

Вы уже должны быть достаточно знакомы с изучаемым языком программирования. Вы должны довольно хорошо знать его и использовать в разных приложениях. Вы должны были выбрать конкретную технологию, применить к ней свои навыки и с легкостью создавать с ее помощью базовые приложения.

Тем не менее вы наверняка по-прежнему не чувствуете,

что в совершенстве владеете языком программирования.

Не волнуйтесь, это тоже нормально.

Когда я только изучал C++, я помню, что даже после того, как я понял все об этом языке, использовал его для создания нескольких приложений и даже поработал программистом, пишущим код на C++, я все еще не чувствовал, что по-настоящему овладел этим языком.

Я чувствовал себя хорошим программистом на C++, но не отличным.

Мне хотелось улучшить свои навыки работы с языком, но я не знал как.

Затем я обнаружил сайт под названием TopCoder, где проводились соревнования для программистов. Каждую неделю там появлялся новый набор задач по программированию, и можно было посоревноваться с другими программистами в решении сложных алгоритмических задач.

Сначала это было ужасно. Я не мог решить даже самую простую задачу. Я смотрел на решения других людей и понятия не имел, как они к этому пришли или даже как работает их код.

Они использовали C++ таким образом, какого я и представить себе не мог.

Но со временем, пытаясь решать все новые задачи и глядя на то, как их решают другие люди, я постепенно начал становиться лучше… намного лучше. Я начал видеть закономерности в том, как решались определенные типы задач. Я на-

чал по-настоящему понимать, как использовать функции C++, которые я раньше игнорировал. Я узнал, как эффективно использовать стандартные библиотеки, языковые возможности и структуры данных для решения сложных задач.

Я стал разбираться в C++ не просто хорошо, а превосходно. Наконец-то я почувствовал, что овладел этим языком.

Я хочу, чтобы вы поступили так же.

Вам не обязательно заходить на TopCoder, есть множество других мест, где вы можете попрактиковаться в решении задач программирования алгоритмического типа.

Я уже упоминал об одном хорошем источнике подобных задач, но вот еще несколько:

- «Карьера программиста» Гейл Лакманн Макдауэлл;
- «Жемчужины программирования»[10] Джон Бентли;
- Проект Эйлер[11];
- Codility;
- Interview Cake;
- TopCoder.

Не думайте, что создать их будет просто, наоборот.

---

[10] Издавалась на русском языке. Питер, 2002 г. – *Прим. ред.*

[11] Проект Эйлер – некоммерческий проект в Интернете, объединяющий любителей математики и программирования. Проект был запущен в 2001 году Колином Хьюзом и сейчас поддерживается небольшой группой энтузиастов. Участники проекта могут выбрать любую из существующих в текущий момент задач и решить ее с помощью любого известного им языка программирования. После ввода правильного числового ответа участник получает доступ к форуму по данной задаче, где участники обсуждают и сравнивают между собой найденные ими алгоритмы. – *Прим. ред.*

И это нормально. Со временем вы поймете, что все задачи делятся на несколько типов, и научитесь с ними работать. Перенимайте опыт других людей, смотрите, **как они справились с теми проблемами, с которыми столкнулись вы**. Старайтесь понять, почему те, кто успешно решил задачу, выбрали именно такое решение. Это один из лучших способов совершенствования себя как программиста.

Лично я для обучения использую ресурс TopCoder. Решив предлагаемые на этом ресурсе задачи с помощью изучаемого вами языка программирования, вы повысите степень своего мастерства и **сможете с легкостью проходить собеседования на должность разработчика**, выгодно отличаясь от других кандидатов.

Глава 7. Высшее образование

В следующих трех главах **я расскажу вам о трех путях (или стратегиях) становления вас как разработчика**. Мы поговорим о **высшем образовании**, о **курсах по программированию**, а также о **самообразовании**. Все эти варианты имеют право на существование, но я хотел бы обратить ваше внимание на плюсы и минусы каждого из них и поговорить о наилучших способах использования каждого из этих подходов.

Давайте начнем с наиболее традиционного пути – высшего (или среднего специального) образования. Думаю, что в рассказах о том, что такое «университет» или «колледж» вы не нуждаетесь. Гораздо лучше будет поговорить о том, к чему следует быть готовыми, отправляясь по этому пути.

Выбор данного пути означает, что ваше обучение будет проходить в аккредитованном учебном заведении по такой специальности, как, например, «информационные технологии», «вычислительная математика» или что-либо подобное. Так начинает большинство программистов. Хороший это способ или лучший? Попробуем разобраться.

Преимущества

Давайте сначала взглянем на преимущества этого варианта. Ваши родители наверняка думают, что плюсов тут огромное количество. Может быть, они даже считают, что получение высшего образования является единственным приемлемым вариантом. Я же постараюсь быть более объективным.

Честно говоря, я не фанат высшего образования, но определенная выгода в получении корочек (красных или синих – неважно) все же есть.

Множество компаний все еще нанимают только тех, кто имеет высшее образование

Несмотря на то, что мы разменяли третье десятилетие XXI века, многие компании относятся к вопросу найма сотрудников очень консервативно. Особенно это касается разработчиков.

Объявляя вакансию, многие фирмы желают видеть только тех кандидатов, у кого в кармане имеется диплом **аккредитованного высшего учебного заведения или колледжа**. Разумеется, это не означает, что в такие компании невозможно попасть без бумаги об окончании высшей школы. Но сделать это будет нелегко. Поделюсь собственным опытом.

Незадолго до окончания колледжа меня наняли на работу в Hewlett-Packard. К тому времени я уже несколько лет ра-

ботал программистом. Фактически я работал в HP по договору подряда. **Обычно Hewlett-Packard не нанимает сотрудников без диплома о высшем образовании**, однако я стал исключением. Так произошло, потому что я доказал свою квалификацию, когда работал сдельно.

Мне пришлось преодолеть огромное количество трудностей, чтобы получить предложение о работе. **А когда я наконец его получил, то был… разочарован**.

Вместо того чтобы принять во внимание мой опыт и способности, меня отнесли к категории сотрудников «без высшего образования». Это означало, что из всего диапазона зарплат мне назначили самую низкую, сказав при этом, что я должен радоваться самому факту приглашения на работу.

Я рассказываю вам все это лишь для того, чтобы показать, как сильно в некоторых компаниях влияет на вашу карьеру наличие документа об окончании вуза. Получив диплом университета или колледжа, вы откроете для себя куда больше возможностей, чем те, кто окончил курсы или является самоучкой.

Тем не менее следует знать, что существуют компании и с другим подходом, которые возьмут вас на работу и без диплома, просто в таком случае выбор будет более ограничен. К сожалению или к счастью, но сделать с этим ничего нельзя. Остается лишь принять ситуацию как есть.

Подытожу: **диплом об окончании университета предоставляет больше возможностей**.

Хорошее понимание основных концепций информационных технологий

Зачастую бывает так, что программисты-самоучки, являясь очень хорошими специалистами, не обладают некоторыми базовыми знаниями, которые можно получить в колледже или университете. Сегодня эти навыки менее важны, чем практические аспекты разработки программного обеспечения, но **я считаю, что каждый разработчик должен знать об операционных системах, структурах данных, алгоритмах, логике предикатов, компьютерной архитектуре и многих других темах, встречающихся в большинстве учебных программ высших заведений**. Эти темы довольно сложно изучить самостоятельно, особенно если вы даже не подозреваете об их существовании.

В следующей главе я расскажу вам о том, как **в некоторых наиболее крутых компаниях** проводятся собеседования разработчиков, в процессе которых интервьюер пытается выяснить, понимает ли претендент все те основные концепции, которые изучаются в вузах.

Я человек весьма прагматичный и обычно выступаю против традиционных систем образования, но также считаю, что **большинству программистов необходимо иметь** некоторые теоретические знания, не относящиеся непосредственно к написанию кода, но лежащие в его основе.

Несмотря на то что университет с меньшей долей вероятности даст вам практические знания, необходимые для рабо-

ты в качестве разработчика ПО, большинство учебных программ позволяет овладеть глубокими знаниями о концепциях информационных технологий. Эти концепции могут быть чрезвычайно полезны при входе в более сложные области программирования, такие как работа с системами, работающими в реальном времени, разработка новых алгоритмов и повышение их эффективности. Такие новые сферы, как машинное обучение, также нуждаются в людях с хорошим пониманием этих концепций информатики.

Структурированные знания

Высшее образование ценно тем, что позволяет получить структурированные знания. Есть немалое количество людей, которые не могут работать без конкретного плана действий. Многие начинающие программисты бросают обучение, потому что их попросту пугает тот объем информации, который нужно изучить, а также отсутствие плана. Кроме того, бывает и так, что человеку сложно чему-то научиться из-за недостатка **силы воли и самодисциплины**.

Если вы узнали себя в одной из перечисленных выше групп, то университет для вас – это лучший способ стать разработчиком. Самостоятельное обучение означает самостоятельную разработку плана обучения, а также самостоятельное определение того, сколько времени ежедневно вы будете уделять программированию. В колледже или вузе есть лишь небольшая часть факультативных дисциплин «по выбору», а

все остальное будет строго регламентировано в соответствии с программой обучения. Вам остается просто придерживаться учебного плана.

Стажировки и прочие возможности

Колледжи и университеты нередко предлагают своим студентам стажировки или какие-либо иные связи и ресурсы, недоступные самоучкам. **В части компаний существует практика найма сотрудников что называется «со студенческой скамьи»**, для чего эти фирмы устанавливают с учебными заведениями соответствующие контакты. В этом случае вам будет гораздо проще устроиться на работу после получения диплома.

Во многих колледжах существуют специальные программы, проводятся различные конференции и другие мероприятия, значительно облегчающие поиск нужных контактов и налаживание связей.

Преимущества такого подхода очевидны, если вы хотите начать свою карьеру в по-настоящему крупной компании (такой как Microsoft или Google).

Опытный разработчик может получить работу в одной из этих крупных компаний на основе заслуг и опыта, но для начинающего разработчика ПО стажировки – отличный способ начать карьеру. Поскольку большинство программ стажировки проводятся в колледжах и университетах, то вы должны быть студентом или недавним выпускником вуза, чтобы иметь к ним доступ.

Недостатки

Но не будем забывать и о недостатках, присущих высшему образованию. **Да, эта часть книги понравится вашим родителям куда меньше.**

Разумеется, недостатки есть у всего, в том числе и у высшего образования. Какие-то из них более очевидны, какие-то – менее.

Затраченное время

Первый и наиболее очевидный недостаток – это время, которое вы потратите на получение диплома. **Учеба в вузе занимает минимум четыре года.** На протяжении всех этих лет вы не будете работать программистом и поэтому не сможете записать этот период в трудовую книжку или резюме.

Как вы понимаете, за четыре года может произойти очень много событий. Поэтому обязательно подумайте, стоят ли все вышеперечисленные преимущества четырех или более лет вашей жизни. Кроме того, вуз сам по себе отнимает время. **Не все предметы будут развивать вас как разработчика.** В учебном плане вас будут поджидать и дисциплины общего характера, не имеющие никакого отношения к разработке ПО. Соответственно, в контексте развития вас как специалиста все эти предметы являются пустой тратой времени. Тесты, контрольные, экзамены тоже не принесут много пользы. То же самое можно сказать и про лекции. Осо-

бенно если вы знаете более оптимальные способы усвоения информация.

Традиционное образование не стремится к максимально эффективному использованию времени. Большинство учебных программ рассчитано не на то, чтобы максимально развить наиболее сильных студентов, а на то, чтобы подтянуть до приемлемого уровня наиболее слабых. Этот аргумент – один из главных доводов против высшего образования. Лично мне высшие учебные заведения кажутся не очень эффективными. Я советую вам хорошенько обдумать этот вопрос, прежде чем вы пойдете подавать документы на поступление.

Финансовые затраты

Следующий недостаток – стоимость обучения. Каждый человек стремится сохранять деньги, а не тратить. Но поступление в вуз означает, что вам придется расстаться с весьма крупной суммой. Вы и сами это знаете, но все же скажу, что **высшее образование – это дорогое удовольствие, и с каждым годом оно становится еще более дорогим товаром**. Если во время обучения вам предстоит жить вне родительского дома, то расходы будут еще выше. **Я знаком с большим количеством разработчиков ПО и людьми других профессий, которые все еще выплачивают кредит за свое обучение – спустя десятки лет после окончания вуза.**

По-моему, это не лучшая перспектива. Ведь если бы вы

вместо получения дорогостоящего образования все эти годы работали и получали за это деньги, вам бы не пришлось оплачивать учебный кредит и проценты по нему. Так что же такого особенного дает высшее образование? Разве что чуть большую зарплату, и то не факт.

Согласен, данная точка зрения выглядит весьма мрачно, но я знаю массу программистов, которые потратили немалые суммы на образование и теперь находятся в весьма **плачевной финансовой ситуации**.

Но не переживайте, у меня есть для вас пара рекомендаций, которые помогут вам смягчить финансовый ущерб, наносимый получением высшего образования. Мы поговорим о них чуть позже, в разделе, посвященном стратегиям.

Позволю себе напоследок дать вам еще один совет.

Убедитесь, что при поступлении в вуз вы учли все факторы, включая расходы на образование, уровень вашего интереса к данной сфере, расходы на проживание и питание, потерянный доход от возможной работы. В общем, все, что поддается подсчету.

Расписываться в собственном невежестве после окончания вуза и просить государство погасить ваш студенческий долг не просто безответственно, а максимально глупо.

Устаревшая и не имеющая ничего общего с реальностью информация

Для того чтобы опубликовать книгу, требуется большое количество времени. Для того чтобы создать учебную про-

грамму, добавить в нее новые предметы или вычеркнуть старые – еще больше.

Зачастую преподавательский состав в колледжах и университетах очень далек от тех приемов разработки ПО, которые используются в реальном мире. В результате образовательные программы вузов плохо отражают навыки и технологии, наиболее необходимые в работе.

Да, знать основные концепции отрасли IT полезно. Однако в работе вам куда больше пригодится умение пользоваться системой контроля версий и знание методологии Agile и наиболее распространенного фреймворка JavaScript.

Справедливости ради следует сказать, что многие учебные заведения понимают свои слабые места и предпринимают определенные шаги, чтобы сделать учебные программы более актуальными. Однако так поступают далеко не везде. И это одна из причин, почему я вообще решил написать эту книгу.

Я хочу дать вам все необходимые знания о карьере в области разработки программного обеспечения, потому что чувствую, что большинство колледжей и университетов не в состоянии это сделать.

Вы, конечно, можете и самостоятельно преодолеть это ограничение, изучив своими силами другие аспекты разработки ПО. Только вот, подозреваю, в процессе этого вам наверняка захочется спросить: «А зачем я тогда вообще плачу за высшее образование?»

Отвлекающие факторы

Если вас когда-нибудь интересовало, почему существует рейтинг самых «отвязных» вузов страны, то следующий абзац ответит вам на него. Многие люди считают студенчество лучшим периодом своей жизни. **Обучение в университете – это не только лекции, семинары, лабораторки и экзамены**

Конец ознакомительного фрагмента.