1 Туда ли я попал?

Чтобы узнать, нужна ли вам эта книга, потратьте немного времени и просмотрите вопросы, перечисленные далее. Отвечать нужно честно и без долгих раздумий. Если вы вообще не понимаете, о чем идет речь в каком-то варианте ответа, не думайте и переходите к другому вопросу.

Я пишу программу:

а) в Блокноте;

б) в браузере;

в) в другом источнике.

Если происходит сбой:

а) я публикую описание сбоя под названием «Помогите!!!» и привожу точный список всего используемого «железа» на подходящем форуме;

б) я ввожу много строк `print`, которые выводят мне содержимое переменных;

в) я занимаюсь отладкой на GDB.

Для проверки версии я использую:

а) ничего не использую. Если вдруг я что-то случайно сотру, мне придется заново все писать. Поэтому я тщательно этого избегаю;

б) SVN;

в) Git или mercurial.

Я комментирую код:

а) никогда не комментирую, потому что мне не хочется долго сидеть за компьютером;

б) никогда не комментирую, потому что думаю, что мой код понятен и без комментариев;

в) никогда не комментирую, потому что мой код не требует разъяснений.

Если мне нужен XML-парсер:

а) я трачу выходные на его написание, как бы сложно это ни было;

б) мне он не нужен;

в) я читаю статьи о SAX- и DOM-парсерах в «Википедии», просматриваю разные библиотеки и их узлы для используемого мной языка программирования, взвешиваю их плюсы и минусы и ищу активные сообщества, связанные с этой темой.

Для валидации почтового ящика:

а) я быстро ввожу две строки, которые проверяю на собственном адресе электронной почты;

б) я проверяю, включен ли в адрес знак @;

в) я ищу в поисковике регулярное выражение (Regular Expression), которое уже было опробовано надежными проектами.

Мой код и его распространение:

а) я держу код в секрете тщательнее, чем государственную тайну;

б) если мой код увидит кто-то, кому уже исполнилось полгода, мне будет немного стыдно;

в) мой код входит в ядро Linux.

Я тестирую код:

а) вообще не занимаюсь этим. Если что-то не работает, рано или поздно я замечаю это;

б) после каждого изменения кода;

в) вообще не занимаюсь этим. После каждого изменения код тестируется на сбои автоматизированными юнит-тестами.

Как выглядит международный стандарт формата даты и времени:

а) Д.М.ГГ, а как еще;

б) количество секунд, прошедших с полуночи 1 января 1970 года, — Всемирное координируемое время, при котором секунды не считаются, а сохраняются в 64-битном операторе;

в) ISO 8601.

Оптимизация:

а) меня не волнует;

б) применяется мной до тех пор, пока программа не начнет выполняться за 43,3485 тактового цикла на моем компьютере 2,4 ГГц Core 2 Duo с 256 Мбайт кэша второго уровня;

в) меня не волнует, если на взаимодействие с пользователем (User Experience) время ожидания никак не влияет.

Другие программисты:

а) лучше меня;

б) хуже меня;

в) бывает по-разному.

Я совершенствую свой код:

а) никогда не совершенствую, но всегда рассчитываю переписать его, чтобы он был более качественным;

б) шаг за шагом, когда у меня есть свободное время;

в) шаг за шагом, даже когда у меня на это нет времени.

Теперь подсчитайте, сколько каких у вас ответов. Если больше ответов *«а»* и *«б»*, то эта книга определенно для вас. Если же у вас больше половины ответов *«в»*, не смейтесь, пожалуйста, над нами и продолжайте дальше программировать драйвер в ядре Linux. Или чем вы там еще занимаетесь.

2 От гордыни к смирению

> Мне регулярно приходится гуглить синтаксис языка, который я использую каждый день на протяжении 10 лет. #coderconfessions.
>
> *@HackerNewsOnion, Twitter, 10 июля 2013 года*

В последние годы в околокомпьютерных кругах часто вспоминают об эффекте Даннинга — Крюгера[1], согласно которому именно непрофессионалам крайне свойственна завышенная оценка своих способностей. Результаты исследований указывают на то, что в реальности все по-другому и гораздо проще: люди в целом плохо оценивают свою компетентность, в основном лишь до некоторой степени правильно.

Программисты с небольшим опытом работы колеблются между завышенной самооценкой и мыслью о том, что они совсем глупы для профессии. Когда в ходе планирования нового проекта к ним приходит вдохновение, они нередко высоко оценивают собственные способности и в то же время у них не возникает даже представления о том, насколько длителен процесс написания исправного кода. Осознание горькой правды приводит либо к отчаянию (в проектах с дедлайном), либо к нежеланию продолжать что-то делать (в проектах для души).

У завышенной самооценки есть предпосылки. Начинающие разработчики постоянно чему-то учатся, и это льстит их эго. Если изобразить процесс обучения в виде графика, то в первые 12 месяцев обучения чему-то новому прямая будет так круто подниматься вверх, что от успехов может закружиться голова. Однако очень сложно распознать то состояние, когда, несмотря на долгое и упорное самообразование, программист все равно может только кое-как барахтаться в лягушатнике. Мир программирования для новичка полон «неведомой безызвестности», говоря языком Дональда Рамсфелда, то есть полон пробелов в знаниях, которые можно вообще не осознавать.

Другая причина завышенной самооценки заключается в том, что неопытные программисты склонны завершать только 80 % заданий проекта и прекращают над ним работать, как только им становятся неинтересны поставленные задачи. Согласно принципу Парето[2] в эти оставшиеся 20 % как раз заложено 80 % работы. Можно создать образцовое приложение, даже обладая небольшими знаниями. Проблемы,

---

1 https://ru.wikipedia.org/wiki/Эффект_Даннинга_—_Крюгера.

2 https://ru.wikipedia.org/wiki/Закон_Парето.

вызванные незнанием или неудачными решениями, показывают свое истинное лицо не в начале проекта, а лишь в ходе работы над последними 20 % задач.

Переоценка собственных способностей или недооценка задачи могут обеспечить и некоторые преимущества. Если бы каждый специалист в самом начале своей карьеры мог точно определить степень своей некомпетентности, человечество все еще жило бы в пещерах («Небоскребы? Мы же ничего не знаем об этом!»). Даже в отдельных проектах иногда полезно ошибаться в оценках. Когда Дональд Кнут, недовольный текстом второго тома своей главной работы «Искусство компьютерного программирования», сам решил в 1977 году написать хорошую программу, на это ему понадобилось около полугода. В результате только почти через 12 лет была выпущена программа TeX. Докторант Джордж Бернард Дантциг, обучаясь на курсе статистики в Университете Калифорнии в 1939 году, слишком поздно обнаружил два недочета в выполненном домашнем задании. Через несколько дней он извинился перед профессором за затянутое выполнение задания: мол, они были сложнее, чем казалось раньше. Однако домашнее задание здесь было ни при чем, Дантцигу совершенно случайно удалось вывести ранее неизвестную в статистике теорему. Таким образом, в некоторых случаях ошибочные решения могут пойти только на пользу, поскольку они помогут не бояться сложности и запутанности задания.

Отличная новость! У других все происходит точно так же. Даже опытные программисты забывчивы, бывают рассеянны, отлынивают от работы и считают написанный ими код самым ужасным в мире. Существует лишь несколько проблем, свойственных исключительно неопытным или торопливым программистам.

Трудноподдерживаемый код

Зачастую код пишут, не принимая в расчет его поддерживаемость (на том же языке либо на других). Это объясняется в первую очередь недостатком профессионального опыта у программиста. Только если изначально самому попытаться разобраться с непроходимыми джунглями кода собственного производства, можно в дальнейшем при написании кода думать о тех, кто его будет читать.

Выбор неэффективных инструментов

Тот, кто знаком только с одной половиной инструментов языка программирования или определенной методологии, будет использовать эти знания при выполнении всех задач, над которыми работает. Часто проблема заключается не в полном незнании, а скорее в недостаточном доверии к другим методам работы.

Переоценка своих способностей

Даже хорошие программисты часто переоценивают эффективность своей работы. С одной стороны, это объясняется тем, что они, как и остальные люди, не могут реально оценить, сколько будет длиться проект. С другой стороны, зачастую постановка конкретной задачи происходит уже в ходе работы. Плохие программисты в данном аспекте ничем не отличаются, разве что у них дела обстоят в три раза хуже.

Нехватка знаний

Для неопытного программиста ново каждое понятие на его профессиональном пути. К тому же в сфере разработки ПО есть довольно много идей, которые постоянно находят применение в новых областях. Опытные программисты могут распознать идеи такого рода и тем самым облегчают себе работу при использовании знакомых понятий в новых контекстах.

Слабые стороны могут быть сильными

Создатель языка Perl Ларри Уолл в своем основном труде «Программирование на языке Perl» характеризует лень, неусидчивость и завышенную самооценку как важные характеристики программиста. Лень нужна, потому что она мотивирует программистов беречь как можно больше сил. Так они смогут написать оптимальную для работы программу, основательно зафиксировать все, что они написали, и составить список часто задаваемых вопросов, чтобы потом не отвечать на их огромное количество. Неусидчивость позволяет программистам писать ПО, которое не только удовлетворяет их запросы, но и может их предсказывать. А завышенная самооценка позволяет программисту творить чудеса. Более того, другие недостатки разработчиков могут стать их преимуществами, если только их правильно использовать.

Глупость

«Нередко тот, кто пишет самый жуткий спагетти-код, как раз и способен удержать его в голове целиком. Ведь только поэтому он так и пишет», — полагает Питер Сейбел[1]. Не очень способный программист попытается найти наиболее простое решение и, вероятнее всего, с его помощью написать код, который не поймут, не прочитают и не усовершенствуют другие.

Некомпетентность

Уверенное владение языком программирования или абстрактными понятиями — штука хорошая. Но как только приходится изменять парадигму — так случилось при появлении объектно-ориентированного программирования, — многие из считавшихся компетентными программистов страдают как раз из-за своих имеющихся знаний. Ничего же не подозревающие индивиды в такой ситуации адаптируются намного легче, поскольку они могут идти навстречу новому без лишнего груза.

Забывчивость

Дуглас Крокфорд, одна из ключевых фигур в создании JavaScript, так ответил в одном интервью на вопрос о том, является ли программирование сферой профессиональной деятельности исключительно для молодежи: «У меня все прекрасно получается, может быть, даже лучше, чем раньше, потому что я научился не зависеть

---

[1] *Сейбел П.* Кодеры за работой. — М.: Символ-Плюс, 2011. — С. 273.

так сильно от собственной памяти. Я теперь лучше документирую свой код, так как уже не столь уверен в том, что смогу вспомнить через неделю, почему я что-то сделал»[1]. Кроме того, программист, который постоянно забывает имена и синтаксис простейших функций, более заинтересован в освоении новой среды разработки или как минимум редактора с умным автодополнением кода.

Слабая выносливость

В определенных кругах считается подвигом программировать ночь напролет. Однако высока вероятность того, что на следующий день код, написанный в таких условиях, окажется бесполезен. Если можно себе позволить работать без мучительных дедлайнов, устанавливаемых начальниками, можно и не принуждать себя к работе над кодом. Зачастую хорошая мысль, пришедшая в голову во время отдыха у реки, может заменить целые недели с немыслимым графиком работы.

Прокрастинация

Именно плохим программистам полезно откладывать улучшение кода на максимально возможный срок. Иногда откладывание работы только играет на руку. Кроме того, рабочие инструменты, языки программирования, кодовые библиотеки и фреймворки со временем становятся только лучше. Чем больше проходит времени, тем проще становится решение, поскольку другие, опытные программисты к тому моменту уже выполнят часть работы. Довольно часто задача может и подождать, пока проблема не будет устранена в открытом проекте. Активное участие в проекте такого рода только ускорит этот процесс.

Отвращение к собственному коду

Ненавидеть собственный код естественно и хорошо. За хороший код всегда цепляются с необоснованным рвением. Плохой код всегда осознанно отправляют в мусорную корзину, как только пользователи захотят изменений или появится новый способ решения проблемы. Те, кто считает свой код совершенным, ничему новому не учатся. Совместная работа с другими программистами всегда проходит продуктивнее, если все участники проекта критически относятся к собственному коду. Чрезмерное внимание к качеству кода также может отнять много энергии, тогда как она намного нужнее для иного — осмысления целей и результатов работы. Код, в конце концов, всего лишь способ решения задачи и не является самоцелью.

Отсутствие целеустремленности

Сносный код иногда может приносить радость — если не его пользователям, то хотя бы создателям. При исследовании феномена счастья выделяют две категории людей: максималистов, которые всегда стремятся к совершенству, и конформистов, которым нужно не так много, чтобы быть счастливыми. Максималисты стремятся к высоким результатам, но конформисты чувствуют себя более счастливыми.

---

[1] Там же. — С. 114.