от джуна до сеньора

как стать </> востребованным* разработчиком

Владимир Швец

Владимир Швец

От джуна до сеньора. Как стать востребованным разработчиком

«Альпина Диджитал»

2023

Швец В.

От джуна до сеньора. Как стать востребованным разработчиком / В. Швец — «Альпина Диджитал», 2023

ISBN 978-5-96-148425-0

Быть разработчиком – трудно, а делать первые шаги – еще труднее. Вам предстоит постоянно практиковаться, осваивать большие объемы сложной информации, вы обязательно столкнетесь с неожиданными вызовами, которые могут легко отпугнуть даже самого заинтересованного и мотивированного специалиста. «Вам придется услышать немало критики, и сразу оговорюсь: корректная критика – это то, что помогает стать лучше, не задевает самооценку и способствует профессиональному росту. Очень важно отличать критику от критиканства. Замечайте, когда вас используют, чтобы подкрепить свое нездоровое эго или самоутвердиться за ваш счет. Такие ситуации вряд ли будут частыми, но нужно быть готовым и к ним». Книга Владимира Швеца, востребованного разработчика с 15-летним опытом работы, поможет вам не сойти с пути и преодолеть все трудности с честью. Она содержит исчерпывающие сведения о проблемах каждого разработчика и способах их решения. Вы узнаете, как писать хороший, чистый код, отлаживать его и оптимизировать, настроить удобный для себя режим работы и без труда общаться с коллегами и руководителями, как справляться с усталостью, выгоранием и гордыней. Каждый раздел содержит непридуманные истории из опыта автора и его коллег, маленькие хитрости и лайфхаки, а также задания, которые помогут вам подготовиться к грядущим испытаниям на пути к новым высотам в карьере. «В реальности код большого проекта расширяется так быстро, что хорошее, продуманное именование не поспевает за ним, но это не значит, что вы не должны уделять этому внимания. Старайтесь делать по одной вещи зараз. Если вы пишете новый код, называйте элементы так, чтобы по ним можно было читать код как рассказ (или хотя бы как хокку). Если вы работаете с уже написанным кодом, будьте бдительны, потому что иногда переменная `sum` может оказаться указателем на открытый файл. Если вы уверены в своих силах, выделите немного времени и поправьте

то, что выглядит нелогичным с точки зрения чтения кода». «Первый совет, который я хочу вам дать, – притормозите. Возьмите больничный, даже если это будет стоить недовольных лиц руководства. Возьмите отпуск, пусть даже вы не будете присутствовать на релизе своего продукта. Если вы исчерпали весь свой ресурс, то можете сделать лишь одно: остановиться и обдумать ситуацию без нависающих над вами дедлайнов, ошибок и клиентов». Для кого В первую очередь для начинающих разработчиков, которые хотят найти свое место в индустрии, а также специалистов в IT, которые уже успели освоиться и теперь жаждут узнать, насколько глубока кроличья нора.

ISBN 978-5-96-148425-0

Содержание

Владимир Швец
От джуна до сеньора: Как стать востребованным разработчиком

Редактор *Ольга Бараш*
Главный редактор *С. Турко*
Руководитель проекта *А. Деркач*
Художественное оформление и макет *Ю. Буга*
Корректоры *О. Улантикова, А. Кондратова*
Компьютерная верстка *К. Свищёв*

* * *

Владимир Швец

ОТ ДЖУНА ДО СЕНЬОРА

Как стать востребованным разработчиком

альпина
ПАБЛИШЕР

МОСКВА
2023

Вступление

Меня зовут Владимир, и я хочу рассказать о том, как выжить в IT. Эта книга предназначена в первую очередь для начинающих разработчиков, которые стремятся найти свое место в индустрии, а также специалистов в IT, которые уже успели освоиться и теперь жаждут узнать, насколько глубока кроличья нора.

Коротко обо мне: более 15 лет я занимаюсь коммерческой разработкой, в основном высоконагруженными веб-системами и приложениями; работал почти на всех должностях корпоративной лестницы – от тестировщика до ведущего архитектора. За свою карьеру я накопил достаточный опыт, которым и хотел бы поделиться в этой книге. На данный момент я продолжаю заниматься разработкой, поэтому книга будет максимально приближена к реальной жизни и особенностям выживания в этой невероятной индустрии.

Книга построена так, чтобы ее можно было использовать практически, исходя из конкретной проблемы или навыка, который вы хотите улучшить. К каждой теме добавлены задания, которые помогут вам преодолеть страх изменений и сделать первый шаг.

Книга состоит из трех основных разделов. Раздел «Код» описывает самые полезные практики по работе с кодом. Раздел «Люди» затрагивает проблемы коммуникации и жизни внутри коллектива. Раздел «Я» посвящен личному росту, особенностям человеческого характера и борьбе с сомнениями и страхами, знакомыми каждому разработчику.

Я искренне считаю, что лучший способ обучения в IT – это практика. Сколько бы вы ни читали книг, ни смотрели курсов и ни слушали подкастов – все это будет бессмысленно, если вы не начнете писать код и развивать навыки самостоятельно. Ваше развитие – это постоянная практика и поиск новых знаний, освоение новых технологий и попытка выяснить, «как же оно устроено». В тексте, возможно, встретятся термины, которых вы пока не знаете, но я специально не разъясняю их, чтобы вы могли активнее участвовать в процессе собственного обучения. Если вам попался непонятный термин, название технологии или совершенно неизвестного вам языка программирования, не поленитесь и воспользуйтесь Google – это окупится. Выполнение заданий, размещенных в конце каждой темы, весьма полезный опыт, которым я бы советовал не пренебрегать. Какие-то задания будут даваться легко, какие-то покажутся сложными, но не пренебрегайте ими, и они в конце концов поддадутся, как и любой навык, которым вы стремитесь овладеть.

В конце каждой темы также будет короткая история из жизни – мой личный опыт, относящийся к описанным проблемам. Она не несет в себе обучающей информации, но, возможно, покажется вам забавной. Смело пропускайте, если вам это неинтересно.

Код

Этот раздел поможет научиться писать код так, чтобы вас не возненавидели коллеги, а новые разработчики, переходя к поддержке вашего кода, мысленно говорили вам спасибо.

Бóльшую часть времени разработчик посвящает коду, и поэтому крайне важно не только быть технически подкованным, но и писать код так, чтобы спустя несколько лет за него не было стыдно.

Темы этого раздела описывают не столько техническую сторону вашей работы, сколько правила, которые позволят содержать ваш код в чистоте и порядке, сделают его понятным для восприятия и удобным для поддержки.

Стиль

Один из важнейших элементов работы над кодом – стиль языка программирования, а точнее, следование стилю, предлагаемому авторами языка. В современном мире практически все языки программирования имеют свой guideline, который предписывает, как оформлять код, какими правилами нужно руководствоваться и как избежать типичных ошибок.

У каждого языка программирования своя специфика, область применения, особенности синтаксиса, и поэтому важно соблюдать единообразие в коде. Люди, долгое время работающие с одним языком программирования, читают его быстро и часто полагаются на визуальную интуицию. Они видели конструкции этого языка так часто, что могут заметить ошибку даже при беглом просмотре.

Если ваш код будет соответствовать стилю языка, на котором написан, это будет огромным плюсом и очень упростит жизнь и вам, и всем тем, кому придется поддерживать этот код в будущем.

Множество современных языков программирования имеет в своем составе специальные инструменты – linters, которые позволяют форматировать код, находить типичные ошибки синтаксиса и многое-многое другое. Регулярное использование этих инструментов окупается тем, что код будет выглядеть предсказуемо и узнаваемо для любого человека, знакомого с данным языком программирования.

Однако из этого правила есть исключения. Большие (и не очень) компании часто могут отступать от стиля языка по тем или иным причинам: особенности использования синтаксиса языка, договоренность среди разработчиков, особенности ведения разработки (давайте представим, что все разработчики этой гипотетической компании работают на мониторах с разрешением, позволяющим без проблем вместить только 80 символов на строке. Кошмарный, ужасный случай).

В случае если в проекте, над которым вы работаете, уже есть соглашение о том, в каком стиле должен быть оформлен код, такой стиль оформления будет более предпочтительным, даже если он расходится с рекомендациями авторов языка программирования.

Вы можете спросить у старших разработчиков, по какой причине был выбран именно такой стиль и насколько он актуален. Не исключено, что решение об отказе от guideline языка программирования было принято очень давно и требует пересмотра. В таком случае поздравляю вас, вы только что повесили на себя массу дополнительной работы по приведению кодовой базы в пристойный вид.

Тезисы

- Guideline языка программирования важен, ознакомьтесь с ним как следует.
- Правила проекта важнее, чем guideline языка программирования.
- Linters – ваши друзья и помощники, используйте их.

Задание

Найдите linters для языка программирования, на котором вы пишете регулярно, или для того языка, который используется на вашем проекте. Проверьте ими код проекта и ужаснитесь, насколько все плохо (или, наоборот, порадуйтесь, как здорово работаете вы и ваши коллеги). Попробуйте найти проблемные места и предложить исправить их. Для вас это будет хорошим опытом работы с кодом проекта, а для проекта – полезным рефакторингом.

История из жизни

На одном из своих первых мест работы я писал frontend для разрабатываемых сайтов, используя JavaScript. На эту должность я устроился, уже имея некоторый опыт работы с JavaScript и (как мне казалось) гениальный метод форматирования кода. Боюсь, что у меня не осталось примеров того самого форматирования (я очень рад, что все примеры утеряны), однако, увидев этот код год спустя, я не просто не узнал его, но еще и долго ругался на автора, создавшего такую бестолковую мешанину из пробелов и отступов. К счастью (или к сожалению), память позволила мне воссоздать, как это выглядело. Узрите же!

```
if (user.loggedIn) {
user.lastLogin=new Date();

sendNotifications([
'Welcome home, '+user.name,
]);

if (user.acl['dashboard.view'] || false)
{
nav.redirect('dashboard.view');
}
}
```

Именование и здравая логика

Основа всех языков программирования – текст программы, способ изложения идей разработчика (привет, ассемблер). Поэтому невероятно важно сохранять читаемость текста программы, простоту его восприятия. Для авторов некоторых языков программирования удобство написания текста программы было большим приоритетом (да, Python, мы говорим про тебя). Авторы других языков, видимо, считали, что вы будете в восторге от обилия скобочек и палочек (да, Objective-C, мы в курсе, что ты в комнате).

Опираясь на guideline языка, вы должны следить, чтобы то, что вы пишете, соответствовало тому, что вы хотите сказать. Если вы заводите переменную sum, в которой храните текущую температуру в фаренгейтах, – поверьте, для вас уже разогревают котел в аду. Старайтесь соблюдать баланс в выразительности: переменная activeSessionsUsersWithGuestRole тоже вряд ли кого-то обрадует. Продуманные названия переменных, классов и функций облегчат жизнь не только вам, но и вашим коллегам.

В реальности код большого проекта расширяется так быстро, что хорошее, продуманное именование не поспевает за ним, но это не значит, что вы не должны уделять этому внимания. Старайтесь делать по одной вещи зараз. Если вы пишете новый код, называйте элементы так, чтобы по ним можно было читать код как рассказ (или хотя бы как хокку). Если вы работаете с уже написанным кодом, будьте бдительны, потому что иногда переменная sum может оказаться указателем на открытый файл. Если вы уверены в своих силах, выделите немного времени и поправьте то, что выглядит нелогичным с точки зрения чтения кода.

Если вам кажется, что ваш код понятен, поставьте себя на место человека, который видит его в первый раз. Или хотя бы на свое место, если вам придется править этот код спустя 5 или 10 лет (да, такие вещи случаются). Хорошие названия – это инвестиция, которая окупается в долгосрочной перспективе. Не пожалейте минуты, чтобы назвать переменную правильно, и вам не придется тратить полчаса в будущем, чтобы понять, зачем она нужна.

Тезисы

- Названия элементов программы должны быть продуманны и логичны.
- Названия элементов программы должны быть сбалансированы, без избыточности и излишней краткости.

Задание

Попробуйте «прочитать» код вашего проекта как текст, как роман, не сосредотачиваясь на его логике. Постарайтесь понять по тексту, что должен делать тот или иной участок кода. Найдите несколько самых запутанных и непонятных мест в тексте кода проекта. Подумайте, как вы могли бы упростить его, сделать понятнее при чтении.

История из жизни

К сожалению, тот пример, который я привел в тексте, – не вымысел. Я действительно находил переменную sum, которая хранила температуру в фаренгейтах. Учитывая тот факт, что код находился в цикле агрегации данных, я потратил немало времени, прежде чем понял, что sum не является суммой всех записей о температурных изменениях. «Спасибо» тебе, неизвестный разработчик. Это не самый страшный пример дурацкого названия переменной, и я готов дать руку на отсечение, что и сам именовал свои переменные довольно идиотским образом.

Повторное использование кода

Рано или поздно вы столкнетесь с ситуацией, когда нужно будет продублировать часть кода, который уже существует в проекте, – но с небольшими (иногда о-о-очень небольшими) отличиями.

Самое простое, что вы можете сделать, – скопировать код и перенести его в новое место (сделать Copy – Paste, как это называют в индустрии), заменив те части, которые того требуют. В этот момент вам следует остановиться, проверить КАЖДУЮ перенесенную строку и задать себе вопросы: будет ли эта строка работать в этой части проекта? Не нарушена ли логика кода? Все ли части скопированного кода имеют смысл в этом месте проекта? Исправили ли вы все комментарии, названия и связи в скопированном коде?

Копирование кода и дублирование считаются дурным тоном, однако в реальности этим занимаются все разработчики, дело лишь в том, насколько они внимательны и предусмотрительны.

Если вы внезапно поняли, что переносите один и тот же код слишком часто и теперь в разных частях проекта появляются одинаковые участки кода, – будьте настороже. Такие дубликаты сложно при необходимости исправлять и улучшать во всех местах одновременно. Этот фрагмент кода стоит выделить в функцию, метод или отдельный компонент, который вы сможете вызывать в местах, где это требуется, и конфигурировать его поведение.

Этим вы спасете себя от многочасовой правки нескольких одинаковых строчек, разбросанных по всему проекту, так как сможете менять поведение этого участка кода только в одном месте.

Однако будьте бдительны: при принятии решения о выделении кода в отдельную сущность (функцию, метод, компонент и т. д.) подумайте, будет ли этот шаг логичным и целесообразным. Если вы, к примеру, пишете код, вычисляющий hash, пожалуйста, удостойте его отдельной функции. Если же вы собираетесь создать компонент, который суммирует два числа, дайте процессору отдохнуть – просто напишите операцию сложения во всех местах, где нужно.

Тезисы

- Copy – Paste – это нормально (прочь, критики).
- Тщательно проверяйте скопированный код на его новом месте.
- Если вы копируете один и тот же код слишком часто, очень вероятно, что он требует отдельного места жительства, а не сотен копий по всему проекту.

Задание

Проверьте код вашего проекта и найдите места, в которых происходят очень похожие действия. Оцените аналитически, действительно ли эти места похожи и как бы они выглядели, если бы вы решили выделить этот код в отдельную сущность. Не торопитесь с рефакторингом этих мест. Достаточно того, что вы научитесь видеть, как именно выглядит дублирующийся код. Вместе с этим придет и интуитивное понимание, когда код стоит просто копировать, а когда он достоин того, чтобы иметь свой собственный угол в проекте.

История из жизни

Как-то раз мне пришлось создать функцию для блока кода длиной 10 строк, который дублировался на проекте 37 (sic!) раз.

Изобретение колеса

Деятельность разработчика часто заключается в том, что он пишет очень похожие решения для очень похожих задач. Это совсем не значит, что такую работу может делать робот, – ничего подобного (этим я успокаиваю не только вас, но и себя). Даже в похожих решениях найдутся требования, которые заставят вас искать новые подходы. Подходы, не позволяющие вам обрастать багажом уже написанного кода, который вы будете просто копировать в каждый новый проект.

Любая задача требует качественного решения, однако чаще всего на этапе декомпозиции вы увидите, что получившиеся мелкие части этой задачи представляют собой типовые проблемы, решения для которых вы писали уже много раз. Чем опытнее вы будете становиться, тем больше подобных типовых решений будете замечать и тем легче вам будет разделять задачу на составные части.

При работе над задачей вы всегда будете стоять перед выбором: либо написать код самостоятельно, либо применить уже написанное, готовое решение. Если у вас недостаточно опыта для решения такой задачи, используйте готовое, проверенное решение. При желании вы всегда сможете сделать рефакторинг и заменить его своим. Ваши приоритеты – стабильность и простота поддержки кода, который вы производите. Не забывайте об этом.

Если вам нужно отсортировать данные, пожалуйста, не пишите свою версию сортировки, они все уже написаны; возьмите готовое решение, исходя из своих требований. Если вам нужно использовать алгоритм шифрования, будьте любезны, проанализируйте доступные решения и возьмите уже написанный и проверенный сообществом код.

Разумеется, это не относится к ситуациям, когда типового решения нет (а таких ситуаций с ростом вашего опыта будет становиться все больше и больше), однако во всех остальных случаях старайтесь использовать проверенные временем (и людьми) решения. Этим вы убережете себя от множества подводных камней, которые обязательно упустите, если начнете делать все сами. Используя стороннее решение, в качестве бонуса вы даже получите определенную поддержку от его сообщества. Иными словами, сможете ли вы поправить свежую уязвимость в алгоритме шифрования так же быстро, как это сделают авторы библиотеки? Я – нет.

Если у вас возникнет мысль, что использовать чужие решения недостойно настоящего самура… разработчика, выкиньте эту мысль на помойку. У вас ограниченный ресурс времени и сил, и вы физически не сможете написать каждое решение, которое будет необходимо. Доверяйте сообществу, доверяйте проверенным решениям. На протяжении карьеры у вас сложится достаточно ситуаций, когда нужно будет написать решение, которого пока просто не существует. Если вы все еще чувствуете неудовлетворенность, используя сторонний код, сделайте свой собственный вариант в нерабочее время. Это подарит вам новый опыт и позволит лучше понять, что именно вы использовали и как оно работает.

Тезисы

- Труд разработчика подобен сборке конструктора Lego, где блоки – типовые решения.
- Предпочитайте готовые и проверенные решения.
- Когда готового решения нет, консультируйтесь с сообществом и пишите свое.

Задание

Проанализируйте код вашего проекта, найдите места, в которых для типовых задач используются самописные решения. Постарайтесь найти аналог такого решения в open source. Попробуйте проверить, насколько

выиграл бы проект при использовании стороннего решения. Возможно, вы бы получили дополнительные функции; возможно, решение, написанное на вашем проекте, менее удобно для использования или даже содержит ошибки.

История из жизни

Мне доводилось видеть функцию получения случайного числа от 0 до 9, которая возвращала последнюю цифру текущего Unix time. Нет, я не шучу.

Экосистема

Любой проект представляет собой экосистему со своими законами, элементами и смыслом. В каждом проекте свой технологический стек, стиль, соглашения о создании кода, правила, по которым он развивается. Приходя на новый проект, вы должны будете принять его правила и придерживаться их в работе. Вам будет необходимо ознакомиться с миром этого проекта, привыкнуть к нему и использовать все, что он может вам дать.

Первые и самые важные элементы экосистемы проекта – это его функция и предметная область. У каждого проекта имеется своя функция. Было бы упрощением сказать, что любой коммерческий проект служит всего лишь инструментом для зарабатывания денег. Вы должны четко понимать предметную область, в которой работает проект, и требования, которые к нему предъявляются.

Очевидно, что некоторые проекты просто не позволят вам до конца разобраться в тонкостях области, для которой они создаются (если вы пишете систему учета налогов, то можете, конечно, попробовать стать бухгалтером, но останетесь ли вы после этого разработчиком?). Однако вы должны приложить максимум усилий, чтобы уменьшить количество белых пятен в понимании логики работы проекта.

На сложных проектах вы, вероятно, будете работать с консультантами – эти люди, разбирающиеся в предметной области проекта, будут составлять для вас рабочие требования. Любое белое пятно должно обсуждаться с консультантами – это убережет вас от гадания на кофейной гуще и досадных ошибок, которые вы можете допустить, если не учтете какое-нибудь простое правило (допустим, вы все же решили не становиться бухгалтером и потому не учли процент подоходного налога, после чего налоговая служба оштрафовала ваших клиентов).

Следующими очевидными элементами экосистемы будут технологии и языки программирования, которые используются на проекте. Независимо от того, работали ли вы уже с этими инструментами или нет, необходимо обновить знания: еще раз ознакомиться с документацией, проверить, не упустили ли вы что-то за последние пару лет, узнать, какие появились новые инструменты, что де-факто стало стандартом при работе с этими технологиями.

Третьей частью экосистемы можно считать технологический стек – все технические элементы, подсистемы, сторонние сервисы, библиотеки, программы, которые используются на вашем проекте. Очевидно, что вы не сможете досконально изучить все библиотеки, однако достаточно разобраться, какие функции они выполняют.

Технологический стек крайне важен, и чаще всего решение о том, что будет в него входить, принимает архитектор проекта исходя из текущих требований. Реальность такова, что небольшие технологические стеки гораздо проще контролировать, и небольшой набор используемых технологий позволяет сделать проект более стабильным и цельным (да, мы можем встроить виртуальную машину JavaScript в приложение-калькулятор, написанное на Python, но, пожалуйста, не предлагайте этого архитектору, если не хотите увидеть, как человек седеет за 10 минут).

То же самое справедливо и для кода, который вы пишете: если вы понимаете, что столкнулись с проблемой, которая требует стороннего решения (вы просто НЕ хотите писать свой текстовый шаблонизатор), сначала убедитесь, что в вашем технологическом стеке нет ничего, что могло бы помочь. Необдуманное добавление новых элементов в технологический стек будет очень опрометчивым решением и приведет к тому, что части системы, которые должны выполнять одну и ту же работу, будут использовать для этого разные компоненты, делая код запутанным, а рефакторинг – кошмарным.

Тезисы

■ Любой проект – экосистема.

■ У каждого проекта есть функции и предметная область.
■ Вам обязательно нужно разбираться в предметной области проекта.
■ Технологический стек проще контролировать, когда он небольшой.

Задание

Проанализируйте код вашего проекта и составьте его технологический стек: опишите технологии, компоненты и сервисы, которые в нем используются, создайте небольшую диаграмму взаимосвязей между технологиями, кратко опишите, за что отвечают те или иные компоненты. Проверьте, нет ли в проекте компонентов, которые выполняют одну и ту же задачу.

История из жизни

Для реализации одного из проектов мне пришлось досконально разобраться в картах Таро и магических практиках. Я не жалею об этом опыте.

Рефакторинг

Рефакторинг – необходимый и очень важный процесс в работе над любым проектом. Особенность почти каждого программного продукта в том, что он постоянно развивается. Меняются требования, разработчики, логика его работы. Если провести аналогию, в начале проекта вас просят сделать стул, а через год вокруг этого стула вырастает целый дом. Код вашего проекта постоянно изменяется, а значит, накапливает массу проблем. Это может быть что-то незначительное, вроде неоднородности стиля кода или отсутствия внутренней документации, но чаще всего проект обрастает неоптимальным, а иногда и устаревшим кодом. И тогда наступает время рефакторинга.

Рефакторинг необходимо проводить регулярно, однако, как показывает практика, заказчики или менеджеры не всегда понимают его важность и часто предпочитают ему реализацию какой-то новой функции, нужной клиентам. Их приоритеты понятны, но и вам не стоит забывать о своих. Ваша задача – писать качественный код и нести за него ответственность. Вы вряд ли сможете доходчиво объяснить необходимость обязательного рефакторинга (если на проекте его проведение обязательно, напишите мне, где вы работаете, – я немедленно отправлю свое резюме). Вы можете заранее закладывать дополнительное время на задачи, связанные с изменением кода. Возможно, это выглядит как попытка схитрить, при этом даже не ради собственной выгоды, но ваш профессионализм того стоит.

Перед началом рефакторинга вы должны быть уверены: то, что вы собираетесь исправить, работает неоптимально или устарело. Иными словами, не пытайтесь изменить код только ради самого процесса изменения, это отнимет у вас силы и время и не принесет ничего, кроме потенциальных ошибок, если рефакторинг окажется большим.

Планируя рефакторинг заранее, вы должны четко понимать, что именно хотите исправить. Очень часто один небольшой рефакторинг выливается в часы или недели дополнительных исправлений, которых никто не планировал. Если вы поняли, что, начав рефакторинг, уходите все глубже, захватывая все новые и новые части системы, – остановитесь и решите, где на данный момент нужно поставить точку. Вы не должны ставить под угрозу всю работу над проектом, а к рефакторингу всегда сможете вернуться и позже.

Приступайте к рефакторингу только в том случае, если сможете точно проверить, выполняет ли исправленный код те же функции, что и прежде. Лучший и самый простой способ для этого – тестирование тех частей кода, которые подлежат исправлению. Если тесты для них отсутствуют, не пожалейте времени и напишите их до рефакторинга.

При рефакторинге обращайте внимание на любые детали кода, над которым работаете. Что-то незначительное, то, что вы сочли устаревшим и ненужным, может оказаться критически важным при каком-либо из сценариев работы, поэтому проверяйте, проверяйте и еще раз проверяйте работу кода после рефакторинга.

Приступая к рефакторингу, заручитесь поддержкой кого-то из старших разработчиков и коллег или как минимум обговорите с ними, что хотите исправить. Возможно, вы видите в коде больше проблем, чем есть на самом деле. Разработчикам свойствен перфекционизм, и иногда это скорее проклятие, чем дар.

Тезисы

- Регулярный рефакторинг обязателен.
- Убедитесь, что рефакторинг определенного кода действительно необходим.
- Планируйте рефакторинг и его границы.
- Проверяйте, проверяйте и проверяйте код после рефакторинга; он должен работать точно так же, как и до него.

■ Не позволяйте перфекционизму утянуть вас на дно, вовремя останавливайтесь.

Задание

Проанализируйте код вашего проекта. Можете использовать linters или проверить код вручную, просто читая его. Найдите проблемные или устаревшие области, которые требуют обновления. Попытайтесь определить границы рефакторинга этих областей кода, проверьте, существуют ли для них тесты. Обговорите возможный рефакторинг со старшими разработчиками или менеджерами – возможно, вам выделят время, которое вы сможете потратить на улучшение проекта.

История из жизни

Однажды я сорвал запуск проекта, потому что неверно оценил время рефакторинга в 2–4 часа. В итоге на него ушло 2,5 недели.

Работает – не трогай

Вы нередко будете сталкиваться с кодом, который выглядит не так, как должен выглядеть хороший код. Возможно, он будет плохо отформатирован или сложен и запутан, и вы почувствуете непреодолимое желание провести рефакторинг. Может быть, вам захочется обновить не сам код, а версии библиотек, используемых в приложении.

Остановитесь и задумайтесь – какую проблему вы хотите этим решить? Если вами движет исключительно чувство прекрасного или желание иметь самые свежие версии, подумайте еще раз. Если ваш проект уже находится в продакшн-стадии и его используют клиенты, если он выполняет свои функции так, как надо, и вы просто хотите навести лоск, то лучше взвесить за и против.

Случаи, когда невинное обновление одной библиотеки приводило к сбою работы всей системы, не пересчитать по пальцам рук всей вашей команды разработчиков. В попытке улучшить все что можно и что нельзя вы нередко можете только все испортить.

Обновление проекта и его улучшение – сложный процесс, влекущий за собой риски, которые вы должны как минимум брать в расчет, а как максимум – иметь смелость нести за них ответственность. Золотое правило «работает – не трогай» звучит забавно, но за ним скрыты многие часы внеурочной работы и стресса разработчиков, системных администраторов, проджект-менеджеров и тестировщиков. Вам, как профессионалу, важно поддерживать проект в рабочем состоянии, даже если он слегка устарел или не использует самых новых инструментов.

Проводите обновления постепенно, дополняйте их проверками, тестами и своей здравой логикой. Старайтесь не ломать то, что работает правильно, даже если оно выглядит не так классно, как вам бы хотелось. Вы всегда сможете вернуться к этому вопросу позже.

Тезисы

- Работает – не трогай.
- В попытке сделать лучше можно все сломать.
- Обновляйте систему постепенно, взвешивая плюсы и минусы.
- Никогда не гонитесь за самыми свежими обновлениями, если они не несут чего-то критически необходимого для вашего проекта.

Задание

Проанализируйте код вашего проекта. Можете использовать linters или проверить код вручную, просто читая его. Найдите проблемные или устаревшие области, которые требуют обновления. Попытайтесь разделить эти области на те, которые стоит поправить как можно быстрее, и те, которые можно оставить в покое.

История из жизни

Я решил обновить системные библиотеки в понедельник днем, пока ждал ответа одного из клиентов, просто чтобы провести время «с пользой». Обновленные библиотеки стерли специфические файлы конфигурации, которые я использовал для запуска продукта в своей системе. Следующие 3 часа ушли на их восстановление и извинения перед клиентом, которого я уже не мог проконсультировать.

Новый код

Начать писать новый код очень и очень непросто, особенно если вы присоединились к проекту недавно. У вас еще нет полного представления о том, какие приняты подходы к решению задач, что считается на проекте дурным тоном и какие из компонентов приложения можно использовать, чтобы достичь требуемого результата.

Лучший способ перестать беспокоиться о новом коде – ознакомиться с уже существующим кодом проекта. Если вы используете систему контроля версий, пройдите по некоторому количеству свежих коммитов, чтобы посмотреть, как выглядит актуальный код ваших коллег. Однако учитывайте, что верить коду в последних коммитах можно лишь наполовину: он совсем не обязательно будет идеальным.

Чаще всего новым разработчикам дают легкие задачи, которые помогают ознакомиться с проектом. Пользуйтесь этим временем, чтобы подробнее разобраться в структуре проекта, его подсистемах, взаимосвязях между ними. На начальном этапе у вас будет гораздо больше времени, чтобы понять основные принципы проекта, а это, в свою очередь, очень поможет всей дальнейшей работе.

На новом проекте любое задание требует чуть более подробного обсуждения со старшими разработчиками. Не стесняйтесь задавать максимальное количество вопросов о том, как сделать лучше, какой код можно использовать в качестве примера, какие компоненты проекта стоит использовать. При написании кода обращайтесь за советом или консультацией: чем больше актуальной информации о проекте вы получите сейчас, тем легче будет ориентироваться в работе в дальнейшем.

В любом случае единственный способ начать писать код, подходящий вашему проекту, – это начать его писать. Не бойтесь ошибиться или сделать что-то не так: любой опыт несравненно полезнее, чем его отсутствие.

Тезисы

- Читайте и анализируйте код нового проекта.
- Используйте время ознакомления с новым проектом по максимуму.
- Спрашивайте, спрашивайте и спрашивайте.
- Не бойтесь допускать ошибки; разработчиков, которые приходят на новый проект и моментально пишут идеально подходящий код, не существует.

Задание

Просмотрите коммиты вашего проекта и найдите те, которые реализуют что-то новое. Попытайтесь понять, чем руководствовался автор кода, и оценить, насколько его код соответствует принципам проекта, общему стилю, подходам к решению задачи.

История из жизни

На одном из проектов с очень амбициозным и своенравным главным разработчиком мне пришлось 20 раз переписывать одну из своих первых задач. Сложность была в том, что мой код должен был выглядеть как его код, а писали мы очень по-разному. Задачу я все же завершил, но сделал для себя выводы о том, что нельзя заставить людей писать так, как хочется тебе, а не им. В дальнейшем этот пример очень помогал мне находить общий язык с новыми

разработчиками на проекте. Каждый раз, когда мне хотелось, чтобы их код был похож на мой, я вспоминал эту историю и просто находил компромисс.

Код как документация

Многие из вас знают, что документирование кода – отличная и даже обязательная практика. Кто-то из вас слышал, что лучшая документация – это сам код, написанный так, чтобы он не нуждался в дополнительном описании. Оба утверждения верны, однако в реальной жизни вы чаще будете работать с кодом, документация и качество которого оставляют желать лучшего (не пугайтесь, вы же разработчик, а значит, сможете это исправить).

Документация – это очень широкий термин, охватывающий многие стороны проекта. Можно говорить о комментариях в самом коде, а также об обязательной документации в коде библиотек, если вы их создаете и предполагаете, что их будут использовать другие разработчики.

Подходы к созданию документации тоже бывают разными: это могут быть комментарии самих разработчиков или труд нескольких специалистов вашей команды, основная работа которых – составление документации проекта. Документация может писаться вручную, собираться автоматически из кода проекта или генерироваться из схем API. Сейчас мы будем говорить только о документировании кода как о наиболее частом случае.

В вашем проекте должно быть соглашение о том, что и насколько подробно надо документировать при работе над кодом. Если такого соглашения нет, следует руководствоваться здравой логикой (и любовью к коллегам). Золотое правило, на которое лучше всего опираться при принятии решения, – спросить себя, насколько код, который вы только что написали, будет понятен тому, кто находится вне контекста задачи и просто открыл файл в этом месте.

Хорошо ли названы переменные, методы и классы, которые вы создали? Позволяют ли они понять, что делают и для чего? Не будьте избыточны (Java, спасибо, что зашел): если вы написали метод add, состоящий из одной строчки кода и суммирующий два числа, не стоит писать к нему документацию. С другой стороны, если вы написали метод, который вычисляет плотность населения на квадратный метр, используя для этого анализ сигналов с мобильных телефонов, – пожалуйста, документируйте это так, чтобы за это выдали премию (а еще вы основательно наплевали на наши права и свободы).

Старайтесь комментировать код по возможности компактно. Краткие и емкие комментарии не будут отвлекать разработчиков при беглом осмотре кода, тогда как избыточная информация станет помехой и усложнит рефакторинг.

Хороший проект – это вкусный коктейль из качественно написанного кода, который не требует документации, и емких комментариев в местах, где трудно разобраться с ходу. Не каждый код можно сделать простым, иногда вам ничего не остается, кроме как описать происходящее словами (и это абсолютно нормально).

Несколько смешных примеров комментариев (надеюсь, вы не встретите подобного в вашем проекте):

```
// Dear maintainer:
//
// Once you are done trying to 'optimize' this routine,
// and have realized what a terrible mistake that was,
// please increment the following counter as a warning
// to the next guy:
//
// total_hours_wasted_here = 42

// I am not sure if we need this, but too scared to delete.
```

```
// When I wrote this, only God and I understood what I was doing
// Now, God only knows

return 1; # returns 1

// somedev1-6/7/02 Adding temporary tracking of Login screen
// somedev2-5/22/07 Temporary my ass

// I am not responsible of this code.
// They made me write it, against my will.
```

Тезисы

- Пишите код как документацию.
- Документируйте емким текстом.
- Иногда код не может быть простым – документируйте!

Задание

Если у вас уже имеется некоторый опыт работы с проектом, проанализируйте места кода, которые вы хорошо знаете, и попытайтесь поставить себя на место человека, который видит этот код впервые. Смогли бы вы понять, что этот код делает? Не показалось бы вам, что было бы проще, окажись в том или ином месте текстовое описание правил выполнения или логики приложения?

История из жизни

На одном из проектов, где я работал, было принято решение комментировать весь написанный код, независимо от того, насколько он прост и понятен. Разработчики любят предсказуемость и простоту, однако оказавшись в ситуации, где формализм существует ради формализма, вы в итоге получаете что-то такое:

```
// check if operation was a success
if (isSuccess) {
// return success marker
return MARKER_SUCCESS;
}
```

Коллаборация

Вам часто придется работать над кодом, написанным не вами. Работа с чужим кодом – дело непростое: вы должны понять, как он работает, привыкнуть к стилю автора. Код может быть написан неидеально, запутанно или неряшливо. В любом случае вы должны уважать чужой труд.

Самое важное в коде – его безошибочная работа, поэтому, если вы видите неряшливый, недокументированный код, который при этом идеально выполняет свою функцию, не торопитесь дополнять его обильными комментариями и заниматься рефакторингом. В случае если ваша задача – избавить код от ошибок, у вас развязаны руки: сделайте все возможное, чтобы он стал лучше.

Если вы столкнулись с необходимостью расширить код или улучшить существующий, постарайтесь связаться с человеком, который его написал, это сэкономит вам массу времени. Даже если автор кода не вспомнит всех деталей реализации, он хотя бы сможет объяснить общее направление своей работы и указать на основные подводные камни, с которыми вы можете столкнуться.

Полагайтесь на тесты кода, который вы исправляете, либо напишите их, если позволяют задача и время: вы должны быть уверены, что не допустите новых ошибок. В данном случае будет лучше написать тесты на существующий код, после чего приступить к его исправлению. Таким образом, закончив, вы сможете проверить код уже написанными тестами и убедиться в том, что логика работы не нарушена.

При исправлении чужого кода старайтесь не втягиваться в конфликты. Иногда люди воспринимают работу над их кодом как сомнение в их профессионализме. Возможно, вы и сами испытываете то же чувство, если видите, что ваш код исправили. Будьте честны с собой и помните: ваша цель состоит в том, чтобы код работал корректно, быстро и без ошибок. Не позволяйте эмоциям или амбициям мешать вашему профессионализму. Поступайте так же и при общении с коллегами: напомните им, что ваша задача – не критика и сомнения, а работа с кодом, ни больше и ни меньше.

При исправлении кода старайтесь подстроиться под стиль автора, даже если он вам не близок, – этим вы сохраните читаемость, не добавляя фрагментации.

Тезисы

- Уважайте чужой код.
- Корректная работа кода всегда важнее его внешнего вида.
- Если есть возможность, пообщайтесь с автором исправляемого кода.
- Напишите тесты до начала исправлений и проверьте уже исправленный код.
- Избегайте конфликтов с коллегами.

Задание

Если на вашем проекте есть практика код-ревью, постарайтесь принять в ней участие в роли как проверяемого, так и проверяющего. Если такой практики нет, уделите время просмотру свежих коммитов от коллег, попытайтесь проанализировать написанный ими код и подумайте, как вы смогли бы его улучшить. Будет здорово, если, выполняя это задание, вы найдете ошибку, – обсудите это с автором коммита, предложите помочь в исправлении.

История из жизни

Однажды ко мне обратился друг с просьбой посмотреть, правильно ли проходит его код-ревью. В то время он как раз устроился в новую компанию и его работу часто проверяли коллеги. Он пожаловался, что сам тон комментариев и претензии, которые они предъявляют, звучат агрессивно и неприятно. Сначала я не воспринял ситуацию всерьез, подумал, что проблема преувеличена, но желание помочь взяло верх и я ознакомился с комментариями к его коду. «Побивание камнями», пожалуй, самое мягкое сравнение, которое приходит мне на ум. Я не знаю, с чем был связан этот разгул немотивированной агрессии по отношению к новому человеку в команде, но в тот момент ситуация расстроила меня настолько, что я, на тот момент уже будучи lead-разработчиком, попросил друга предоставить мне слово на код-ревью. Я высказал все, что думаю о таком отношении внутри команды и о самой политике компании, которая это поощряет, после чего искренне посоветовал другу сменить место работы. Порой есть смысл тратить время и силы, чтобы попробовать изменить положение к лучшему, но эта ситуация была из разряда «собирай вещи и беги».

Отладка

Отладка по многим причинам должна занимать особое место в вашем сердце. Под отладкой мы подразумеваем любые действия, которые помогают проанализировать работу кода: от вывода сообщений от работающей программы в консоль или браузер до создания debug-сборки вашего приложения с последующим профилированием. Отладка кода – одно из самых лучших средств для анализа работы приложения и поиска проблем.

В зависимости от языка программирования проекта вам будут доступны разные способы отладки. Некоторые языки программирования предоставляют очень богатые возможности для отладки, другие, наоборот, не имеют их совсем. Но не стоит отчаиваться: даже работая с языками, чьи средства отладки минимальны, вы всегда сможете воспользоваться встроенными средствами диагностики, пусть даже это будет простой вывод на экран.

Самый простой (и старый) способ отладки – это непосредственный вывод на экран. Это может быть вывод информации о текущем состоянии кода в консоль, в браузер, в интерфейс приложения. Это быстрый и действенный способ: вы просто выводите на экран состояние переменных, результат выполнения функций и все, что сочтете необходимым. Однако я очень (ОЧЕНЬ) рекомендую настроить инструменты для отладки (отладчик), если они доступны для вашего языка программирования. Иногда настройка отладочных инструментов занимает какое-то время, но, поверьте мне, это сэкономит вам невероятное количество времени и нервов в дальнейшей работе.

Одно из самых главных табу – недосмотреть и оставить в проекте отладочный код (к примеру, вывод отладочной информации или искусственную паузу в выполнении кода). Это может не только нарушить работу приложения (вы удивитесь, насколько часто забытый в коде sleep становится в дальнейшем невероятной «оптимизацией», когда его наконец находят и удаляют), но еще и скомпрометировать данные клиентов, которые будут отображены в логах или записаны в файл и прочитаны людьми, для которых они не предназначены. Настройка отладочного инструментария избавит вас от таких опасных ситуаций.

Тренируйтесь работать с отладчиком, не идите коротким путем, решая проблему разбрасыванием выводов на экран по всему коду. Во-первых, это решение только кажется самым быстрым: в какой-то момент вы все равно столкнетесь с ситуацией, когда одного лишь вывода на экран будет недостаточно. Во-вторых, вы упускаете невероятно ценную возможность увидеть код так, как его видит интерпретатор вашего языка или операционная система при выполнении. Регулярная работа в отладчике при поиске ошибок или анализе кода позволит вам почувствовать себя машиной – видеть то, что видит она при выполнении кода, следовать тем же путем. Это поможет выработать интуицию и способность замечать потенциальные ошибки при написании кода.

Профилирование – отдельный подход к анализу кода: оно позволяет определить, насколько ваш код эффективен, обнаружить в нем медленные места, избыточные вызовы, утечки памяти или нецелесообразное использование ресурсов операционной системы. Этот инструмент используется не так часто, как отладчик, но вы должны уметь применять его, когда это необходимо (ваше приложение заняло всю оперативную память или накалило процессор так, что на нем можно зажарить стейк, – тогда закатывайте рукава и открывайте профилировщик).

Тезисы

- Старайтесь обходиться без выводов на экран.
- Найдите время на настройку отладчика, оно окупится с лихвой.
- Пользуйтесь отладкой регулярно, не позволяйте себе утратить навыки.

■ Профилируйте, чтобы понять, насколько эффективно ваше приложение.

Задание

Найдите отладчик и профилировщик для языка программирования вашего проекта, если они доступны. Настройте их, разберитесь с документацией и принципами работы, командами и возможностями. Запустите код проекта и начните отладочную сессию, проходя по коду, который вы недавно написали. Попробуйте разные возможности отладчика, изменяйте состояние вашего приложения в реальном времени, нарабатывайте опыт, чтобы при необходимости быстро найти ошибку. Запустите профилировщик и определите, действительно ли приложение эффективно. Спросите старших разработчиков, существует ли практика профилирования вашего проекта, и если нет, то стоит ли ее ввести.

История из жизни

Из всех историй отладки мне запомнилась одна, когда мне казалось, что я вот-вот лишусь рассудка. Я отлаживал свое Windows-приложение, которое, в силу условий, поставленных клиентом, должно было использовать одну из стремительно устаревающих технологий OLE (если вы пишете с использованием OLE – простите, эта претензия не лично к вам). Моя головная боль началась в тот момент, когда я понял, что у приложения «течет память». Как и любой порядочный разработчик, я, разумеется, начал подозревать в этом свой код. Спустя неделю, лишившись огромной части нервных клеток и изрядно поседев, я смог найти утечку в одной из системных DLL Windows, датированную 1995 годом выпуска. Скажу честно, я искренне попытался найти авторов и сообщить им о найденной проблеме, но безрезультатно. Меня не покидает мысль, что все, кто хоть что-то помнил об этой библиотеке, либо сняли руки с клавиатуры, либо уже покоятся с миром. Покойтесь мирно, но нервов мне, конечно, жаль.

Инструменты и автоматизация

Работая над проектом, крайне важно создать среду, в которой вы наиболее эффективны, начиная от выбранных вами инструментов разработки и заканчивая операционной системой и «железом». Все должно работать так, чтобы вам было максимально удобно.

Единственное мнение, к которому следует прислушиваться в этом вопросе, – свое собственное. Вы чувствуете, что эргономическая клавиатура вам не подходит? В мусор ее. Вы чувствуете, что монитор, повернутый, как у всех коллег, на 90 градусов, вас раздражает? Поверните его так, как вам удобно. Вы чувствуете, что максимально эффективны в Emacs, а не в полноценной среде разработки, оплаченной вашей компанией? Что ж, вы сэкономили компании немного денег – работайте в Emacs.

Выбор инструментов – долгий процесс, и вам потребуется немало времени, чтобы определить, как и чем удобнее всего работать. Если вы постоянно пробуете новые инструменты и подходы, это тоже правильно. Как можно понять, что вам что-то подходит больше, если не испробовать все варианты? Как шеф-повар, как плотник, вы должны знать свои инструменты досконально.

Не бойтесь пробовать новые подходы. Вы разработчик, в нашей индустрии вещи успевают устареть, едва появившись. Если процесс выбора доставляет вам удовольствие, уделите ему побольше времени (можно даже рабочего, но я об этом не говорил). Найдите то, что подходит вам больше всего.

Старайтесь автоматизировать рутинные задачи. Вы знаете, что за день открываете более 100 файлов? Найдите и запомните shortcut на окно открытия файлов. Вам необходимо скачать множество файлов с внутреннего сервера? Найдите для этого утилиту или напишите короткий скрипт. Это кажется мелочью, но она сэкономит куда больше времени, чем вы думаете. Кроме того, вы сможете работать с разными областями IT, впитывать больше знаний и тренироваться в технологиях, с которыми раньше не сталкивались (возможно, вы даже решите написать свой первый скрипт на Perl).

Если вы работаете со сложным проектом, требующим комплексной процедуры загрузки или настройки, – автоматизируйте. Вы будете тратить куда меньше времени на подготовку к работе и вдобавок сможете поделиться своим решением с коллегами, чем облегчите жизнь всей команде.

Помимо прочего, автоматизация нередко приводит к созданию нового проекта, иногда весьма успешного. Часто необходимость автоматизировать какой-то процесс выливается в скучные рутинные действия, которые все выполняют одинаково. И в таком контексте новый проект, позволяющий упростить этот процесс, становится долгожданным и очень желанным инструментом.

Тезисы

- Выбирайте инструменты самостоятельно.
- Учитесь работать со своими инструментами, становитесь эффективнее.
- Пробуйте новые инструменты и подходы.
- Автоматизируйте рутинные задачи.

Задание

Постарайтесь запомнить все моменты в течение рабочего дня, которые требовали от вас рутинных действий. Определите, какие из них раздражали вас больше всего и что было самым скучным. Подумайте, как бы вы могли

автоматизировать эту часть работы. Кажется ли она такой же скучной другим разработчикам проекта?

История из жизни

Мне всегда нравилось выбирать инструменты, пробовать редакторы кода, среды разработки, утилиты, инструменты автоматизации. Часто это давало преимущество в принятии решения о выборе инструмента – у меня уже был опыт и представление о том, что я хочу видеть и чем хочу пользоваться на проекте. У этого подхода есть и обратная сторона: чтобы сделать HTTP-запрос, я скорее напишу Python-скрипт, чем просто воспользуюсь cURL. Но это определенно не то, из-за чего я стану огорчаться.

Тесты

Тесты – это большая (и не всегда однозначная) часть любого крупного проекта. Тестирование предполагает проверку функций, корректности их работы, проверку ожидаемых результатов, и не всегда оно заключается только в тестировании кодом. Тестирование выполняется вручную или автоматически, этим могут заниматься коллеги-тестировщики, и (в самом худшем случае) у вас всегда будут тестировщики со стороны – пользователи вашего продукта.

Давайте сразу проясним один важный вопрос о тестировании: оно необходимо. Если у вас уже есть опыт работы, то, возможно, вы сталкивались с проектами без тестов, с разработчиками, которые относятся к тестированию и написанию тестов скептически, с руководителями и клиентами, которые не хотят тратить время и деньги на написание тестов. Но все это не имеет значения, потому что, каков бы ни был ваш проект, одно я могу сказать наверняка: ему нужны тесты.

Тема тестирования кода так же стара, как и само его написание. Никто из разработчиков не завершит работу над задачей, не проверив предварительно, работает ли она так, как требуется. В этом смысле каждый разработчик является тестировщиком.

В этом же кроется один из основных подводных камней, вынуждающих нас писать тесты или нанимать отдельных специалистов для тестирования кода. Разработчик не всегда способен корректно и полноценно протестировать собственный код. Отчасти это связано с тем, что, работая над конкретной задачей, мы упускаем из вида общую картину проекта; отчасти с тем, что, разрабатывая код, мы интуитивно осознаём его потенциальные недостатки и при тестировании пытаемся обойти их, проверяя лишь формально.

Тестирование проекта условно можно разделить на две большие категории: проверка человеком и проверка кодом. В идеальном мире нам не приходится выбирать из этих двух вариантов, так как они дополняют друг друга. В реальности же вы можете оказаться в ситуации, когда у вас нет ни тестировщиков, ни написанных тестов.

Тестирование человеком направлено на те проблемы, которые обнаружат пользователи вашего продукта (а они обнаружат, поверьте). Тестирование кодом направлено на формальный поиск ошибок, когда проверяются заранее подготовленные сценарии и данные.

Тестирование человеком всегда дает вашему проекту преимущество, потому что человек – это невероятная аналитическая машина, способная совершать фантастические глупости. Как разработчику вам никогда даже в голову не придут вещи, которые пользователь может сотворить с продуктом. Буквально. Просто поверьте мне на слово. Тестировщики, конечно, будут относиться к вашему проекту более бережно, однако, как опытные специалисты, они (как и вы в коде) интуитивно чувствуют слабые места и выявляют проблемы продукта куда быстрее и качественнее самих разработчиков. «Володя, я кликнул на это поле ввода три раза, отсоединил и подсоединил обратно клавиатуру, подпрыгнул на стуле, и наше приложение зависло» – да, готовьтесь именно к такому.

Тестирование кодом служит другим, не менее важным целям. Технические способы проверки кода весьма обширны, они включают много терминов и понятий. Не рассчитывайте испробовать все подходы к тестированию в рамках одного проекта. Задача формального тестирования состоит в том, чтобы проверить, насколько предсказуемо ведет себя код, основываясь на ваших требованиях и исходных данных.

Как правило, приступая к очередной задаче, вы, как разработчик, располагаете всеми данными, необходимыми для ее реализации (если нет – вы явно пропустили фазу анализа и сбора требований, ай-яй-яй). Следовательно, вы можете заранее описать сценарии работы еще не написанного кода. На этом принципе базируется методология разработки и тестирования TDD. Формализм и аккуратность будут вашими лучшими помощниками при написа-

нии таких тестов, однако знайте, что есть грань, пересекать которую считается дурным тоном. Однажды формализм заставит вас написать проверку того, что true является true. Встаньте, передохните и постарайтесь больше не превращаться в робота.

Не менее полезно в написании тестов то, что они помогают отслеживать регрессии кода. Покрывая тестами какую-то функцию проекта, вы с определенной долей уверенности можете на них положиться и использовать их, когда начнете эту функцию изменять. Провалившиеся тесты будут первым звоночком, что вы сделали что-то не то.

Как вы уже поняли, тестирование – это пункт, который нельзя просто пропустить. Писать тесты может быть очень-очень скучно. Тестировать функцию может быть невероятно утомительно. Но этим вы помогаете себе и проекту куда больше, чем думаете. Просто представьте: один провалившийся тест в вашей локальной консоли или один упавший продакшн-сервер с миллионом клиентов.

Используйте рабочее время, дополняя тестами код, который пишете. Включайте время на тесты в ETA задач, над которыми будете работать. Вы не сможете найти все ошибки, которые есть в коде, это правда, но, эй, мы же оба знаем, что вы хотите сделать его максимально стабильным. Это ваш профессионализм, это ваш код. Инвестиция своего времени в тестирование вернется вам вдвойне.

Тезисы

- Тесты нужны, даже если вас убеждают в обратном.
- Идеальный вариант, если ваш код будут проверять тестировщики или хотя бы ваши коллеги.
- Терпите скуку формальных тестов, это окупается.
- Один упавший тест – минус сотня недовольных клиентов.

Задание

Узнайте, используются ли на вашем проекте тесты. Проверьте их актуальность, настройте окружение для тестирования. Проведите тестирование проекта. Если тесты не выполняются или все красные от ошибок, поговорите с кем-нибудь из старших разработчиков или коллег, чтобы понять, в чем может быть проблема. Если у вас нет тестов или они не обновляются, обсудите ситуацию с коллегами, узнайте, почему работа над тестами не ведется. Выберите какой-нибудь код, написанный вами недавно, и попробуйте написать для него тесты (хотя бы для себя).

История из жизни

Вам никогда не встречался тест, который использует текущее системное время для проверки работоспособности проекта? Мне встречался. А знаете, чем будет знаменит такой тест, чем он запомнится вам, когда вы с ним встретитесь? А вы обязательно с таким встретитесь или даже напишете свой. Ваши тесты будут либо выполняться безукоризненно гладко, либо останавливаться с ошибками, и вы потратите уйму времени на то, чтобы понять: все, что меняется между их запусками, – это секунды, которые сменяют друг друга. Тик, тик, тик… Успешный тест, неуспешный тест, успешный тест…

Идиоматичность

У каждого языка программирования есть свое неповторимое «лицо», свой шарм и обаяние. Помимо синтаксиса и особенностей реализации, идиоматичность языка состоит в том, КАК на нем пишут. Сюда входят лучшие практики по написанию типовых решений, устоявшиеся в сообществе языка подходы к использованию языковых конструкций, стилистические особенности оформления кода и многое другое. Иными словами, идиоматичность языка напрямую определяется его применением. Нередко сами авторы языка описывают наиболее подходящие способы решения типовых задач, тем самым давая разработчикам подсказку, каким образом следует использовать их язык программирования.

Идиоматичность кода трудно воспринять с первого взгляда. Вам потребуются опыт и много практики для того, чтобы «почувствовать», как стоит писать на том или ином языке программирования. Скорее всего, рано или поздно вы сами придете к идиоматическому написанию кода. Однако я очень рекомендую вам ускорить этот процесс: читать определения авторов языка программирования, код стандартных библиотек и код значимых проектов, написанных на этом языке.

Нет ничего ужасного в том, чтобы писать на одном языке программирования так же, как вы писали бы на другом. Просто это непрофессионально. Не понимая, как писать код идиоматически, вы зачастую будете приходить к решениям, которые окажутся не такими эффективными, стабильными и элегантными, какими могли бы быть.

Воспринимайте написание идиоматического кода как разговор на иностранном языке. Без изучения правил и постоянной практики вы будете говорить с акцентом, и чем больше вы будете стараться говорить на иностранном языке как на родном, тем сильнее будет акцент и тем хуже вас будут понимать носители языка.

Все, что вам необходимо, чтобы начать писать идиоматический код, – опыт, знание правил и много-много практики. Наблюдайте, как пишут код люди, давно работающие с этим языком (для этого очень полезно читать исходный код стандартных библиотек, написанный авторами языка программирования). Читайте инструкции и документацию от создателей языка – они провели с его кодом куда больше времени и точно знают, как писать на нем с максимальной эффективностью. И конечно же, постоянно практикуйтесь в написании кода.

Если в течение карьеры вам пришлось многие годы писать на одном и том же языке программирования, не расстраивайтесь, что теперь пишете профессионально только на нем. При переходе к новому языку программирования вас будет тянуть писать на нем так, как вы привыкли писать на предыдущем. Это нормально, но следует четко понимать, что у разных языков разная идиоматичность и ее нужно уважать. Нередко для того, чтобы вы полностью восприняли новый язык, должен совершиться качественный переход (щелчок в голове): вы внезапно увидите язык как целое, его правила станут прозрачнее, а сама работа с ним – проще и естественнее. Не старайтесь ускорить приближение этого момента, практика и опыт, как всегда, сделают свое дело.

Тезисы

- Каждый язык программирования обладает своей идиоматичностью.
- Не пишите код на каждом новом языке программирования так же, как на предыдущем.
- Читайте код, читайте советы авторов языка, практикуйтесь.
- Если «заржавели» на одном языке, не бойтесь переключиться на другой и потратить время на его изучение.

Задание

Если у вас уже накоплен определенный опыт в работе с одним языком программирования, найдите несколько написанных на нем малоизвестных open source проектов. Проанализируйте их код: выглядит ли он идиоматически верным? Что бы вы смогли улучшить, опираясь на свой опыт работы с данным языком? Познакомьтесь с каким-нибудь новым языком программирования, совсем не похожим на привычный вам (я даже дам подсказку – берите Lisp).

Создайте свой небольшой проект и попытайтесь перенести маленький кусочек кода проекта на новый язык. Интуитивно ощутите разницу при написании кода на двух разных языках программирования, попробуйте переделать перенесенный код так, чтобы он соответствовал идиоматике выбранного языка.

История из жизни

Вместо десятка слов я покажу вам код. Здорово, если вы уже немного знаете PHP и JavaScript, вам будет легче оценить эту шутку:

```
$sum = 0;
$arr = new Array(1, 2, 3);

for ($i = 0; $i < $arr.length; $i++) {
$sum += $arr[$i];
}
```

Это JavaScript-код, но написанный так, будто это PHP. Нет, делать так не надо никогда.

Open source

«With great power comes great responsibility»[1] – это, пожалуй, самое точное описание, которое можно дать open source проектам. В своей карьере вы совершенно точно столкнетесь (и уже сталкивались) с open source решениями, которые будете использовать на благо ваших проектов. Приложения, библиотеки, сервисы, компоненты, утилиты – все, что вы можете представить, и еще немного сверх того.

С точки зрения кода open source – это самый большой помощник, который у вас есть. Современная коммерческая разработка не терпит задержек и никого не ждет. Рынок диктует условия и сроки. Продукт, который не был запущен «прямо сейчас», через месяц окажется задавлен конкурентами или потеряет актуальность. Вы не можете позволить себе потратить жизнь только на то, чтобы написать свой веб-сервер, когда захотите завести себе небольшой сайт-визитку. Поэтому все разработчики приходят к open source, используют его, стараются делать лучше, участвуя в разработке открытых проектов, берут и отдают одновременно.

Open source решения могут быть качественными или собранными на коленке, огромными или совсем крохотными, но вы будете их использовать, а значит, нужно усвоить несколько правил, которые спасут вас от хаоса открытых проектов и позволят наслаждаться их дарами.

Правило № 1. Всегда проверяйте лицензию open source проекта, который хотите использовать. Это звучит как шутка (мы же говорим об OPEN source, эй!), однако вы должны уважать автора продукта и его правила игры. Нужно быть уверенным, что этот код разрешено применять в условиях вашего проекта. Обязательно сохраните упоминания об авторе и лицензии, если это требуется. Если лицензия для open source проекта не указана, не пожалейте нескольких минут на то, чтобы спросить автора напрямую или создать тикет с соответствующим вопросом. Этот пункт особенно важен, если вы работаете над коммерческим проектом. Лучше быть уверенным в лицензии заранее, чем, спустя полгода получив повестку в суд, пытаться переписать половину кода проекта.

Правило № 2. Выбирайте для своего проекта только проверенные и поддерживаемые open source компоненты. Если вы собрались добавить к своему проекту текстовый шаблонизатор, проведите анализ, составьте список библиотек, которые подходят по лицензии и удовлетворяют вашим требованиям. Убедитесь, что проекты достаточно развиты и над ними ведется работа. Проверьте, как авторы реагируют на сообщения от своих пользователей об ошибках и не оставляют ли их без внимания. Создайте proof of concept решение с использованием нескольких выбранных проектов, чтобы проверить, какой подходит вам больше всего.

Правило № 3. Следите за обновлениями open source проектов, которые используете. Пропущенное обновление безопасности может дорого вам стоить, во всех остальных случаях следует руководствоваться правилом «работает – не трогай». Нередко open source проекты предоставляют обновления без обратной совместимости, и ваш проект может выйти из строя, пока вы не проведете необходимую миграцию или не откатитесь до предыдущей версии компонента, который обновился.

Правило № 4. Не бойтесь «заглядывать под капот». Нередко наступает момент, когда вам становится непонятно, что именно происходит в той части системы, где используется open

[1] С большой силой приходит и большая ответственность (*англ.*).

source решение. Не создавайте интригу: загляните в код (это все-таки open source!), попробуйте разобраться, что происходит не так, поднаберитесь новых идей. Чтение open source кода – замечательная практика, которая помогает знакомиться с новыми языками программирования, нарабатывать опыт интуитивного чтения кода, узнавать об оптимальных способах решения тех или иных проблем.

Правило № 5. Старайтесь участвовать в жизни проектов, которые используете. Open source сообщество живет разработчиками, их желанием создавать и делиться. Не упускайте возможность почувствовать это, заводите тикеты, участвуйте в обсуждении новых фич, внесите свой маленький вклад. Можно просто сказать спасибо, а если вы внезапно ощутили особый прилив любви к какому-то open source проекту – узнайте, принимают ли они донаты. Люди вкладывают в эту работу много сил и времени, им будет очень приятно, что вы оценили их труд.

Тезисы

- «With great power comes great responsibility».
- Проверяйте лицензию у любого open source проекта, который собираетесь использовать.
- Выбирайте только качественные, проверенные временем и людьми проекты.
- Следите за обновлениями, но не обновляйтесь без необходимости.
- Изучайте код open source проектов, читайте его, старайтесь разобраться в том, как он работает.
- Не только берите, но старайтесь и отдавать что-то open source сообществу, только так оно будет оставаться живым.

Задание

Найдите open source решения, которые используются на вашем проекте. Проверьте, актуальны ли они, как давно обновлялись, появилось ли в них что-то новое. Есть ли решения, уже отмеченные авторами как устаревшие и неподдерживаемые? Если да, попробуйте составить список альтернативных проектов, которыми вы могли бы их заменить. Попробуйте найти места в коде вашего проекта, которые реализуют что-то очень обычное (то, что можно было бы заменить на библиотеку). Проанализируйте, что может дать вашему проекту замена этого кода на open source компонент: станет ли проект гибче, получит ли новые возможности – или ваше решение не требует замены и работает слаженно именно с вашим проектом?

История из жизни

Из-за постоянной занятости у меня никогда не хватало времени на то, чтобы полностью участвовать в жизни open source, равно как и на то, чтобы писать открытые решения. Однако я все же пересилил себя и написал небольшой плагин к редактору VIM, который приводил текст комментария к табличному виду. Первый коммит на GitHub датирован 16 декабря 2012 года, а сам плагин на момент написания этой книги имеет целых (ух-х-х!) 42 звездочки. И эта микроскопическая капля в море open source невероятно радует меня каждый раз, когда кто-нибудь устанавливает себе этот плагин и пользуется им.

Серебряные пули

В какой-то момент вам может начать казаться, что для решения той или иной задачи идеально подходит конкретный язык программирования, библиотека или инструмент. И вполне возможно, что вы правы. Но реальность такова, что для идеального решения должна существовать идеальная задача. Разработка проекта ведется в контексте постоянно изменяющихся требований (как бы нас это ни раздражало). Одни задачи сменяются другими, варьируются требования и парадигмы, инструменты и подходы.

Вы должны очень гибко подходить к проблемам, которые пытаетесь решить технически. Необходимо следить (не следовать) за новыми подходами к решению старых задач. То, что вы делали два года назад определенным инструментом, может абсолютно не подойти вам сегодня. Ищите качественные и актуальные решения. Решения, которыми вы сможете гордиться через несколько лет.

Вам нередко будут встречаться разработчики, которые считают, что нашли идеальный способ что-либо сделать. Относитесь к таким утверждениям с изрядной долей скепсиса. Наша индустрия очень быстро изменяется, эволюционирует. Попытки использовать одни и те же подходы на протяжении долгого времени приведут к тому, что система будет деградировать, становиться неподдерживаемой и медленной. Многие инструменты в момент своего появления были уникальны и незаменимы, однако постепенно становились все более бесполезными (прости, jQuery, но это правда).

Будьте гибкими, не позволяйте опыту (даже личному) вставать на пути к качественному коду.

Тезисы

- Серебряной пули не существует.
- Всегда ищите лучшие варианты, но не бегите за трендами.
- Не останавливайтесь на одном решении, старайтесь сделать лучше.

Задание

Проанализируйте код вашего проекта и найдите инструменты, библиотеки или компоненты, которые используются достаточно давно. Узнайте, насколько они актуальны, поддерживаются ли, какие для них есть альтернативы.

История из жизни

Не проходило и нескольких лет, чтобы мир разработки не получал в свое распоряжение какую-нибудь новую блестящую игрушку и целую армию ее фанатов, утверждающих, что ВОТ УЖ ТЕПЕРЬ ВСЕ ПРОБЛЕМЫ БУДУТ РЕШЕНЫ. Однако чем дольше находишься в этой индустрии, тем чаще видишь одни и те же технологии, приправленные новой маркетологией. Черт, было время, когда и я думал, что C++ сможет решить любую мою проблему. Идеального кода нет, как и идеальных решений, так что отбросим невротический поиск абсолюта: «делай что должно – и будь что будет».

Код ради кода

Открою вам секрет: разработчики любят код. Можно сколько угодно делать вид, будто мы приходим на работу за зарплатой, но если вы настоящий разработчик, то знаете, о чем я говорю. Нам нравится код, нравится создавать что-то из ничего. И мы любим играть: с новыми технологиями, с новыми идеями и подходами, с новыми языками программирования, с новыми библиотеками и инструментами.

Если вы с энтузиазмом познаете новое, с удовольствием разбираетесь в подходах к разработке или обожаете писать свои pet projects, будьте осторожны: так можно перепутать работу и игру. Нет ничего лучше, чем любовь к своей работе, и ваша тяга к новым знаниям – это прекрасно (я бы вас даже обнял, но руки заняты набором этого текста). Однако работа на коммерческом проекте, с задачами, которые поставлены другими людьми, с финансовыми потерями или, наоборот, выигрышами накладывает свои обязательства.

Не поддавайтесь искушению играть с кодом на работе. Вам может захотеться встроить в проект Lua-интерпретатор или написать микросервис на Scala, но до конца ли вы уверены, что это решение – результат анализа потребностей проекта? Может, вам просто хочется испробовать новый язык или технологию? Действительно ли эта новая библиотека, которую так хочется использовать, нужна проекту, или вам просто любопытно, как она работает?

Старайтесь разделять любовь к знаниям и профессиональные навыки. Поверьте мне: даже в рамках несколько наскучившего вам проекта есть масса вещей, которые вы можете улучшить, не применяя десяток новых технологий. Не поддавайтесь желанию испробовать каждую технологию, которую знаете, на своем рабочем проекте.

Ваша работа – не супермаркет игрушек, поэтому вы должны ответственно подходить к тому, что и как использовать. Никто не станет сдвигать рассчитанные ETA, чтобы вы могли узнать, а так ли продуктивен Go в действительности. Никто не будет мириться со штрафами от клиентов, пока вы проверяете, подходит ли Rust в качестве альтернативы языку программирования проекта (разве что вам поставили такую задачу, если это так – я очень за вас рад).

Вам нужно уметь работать в рамках и интересах проекта для его коммерческого успеха. Жажда знаний и страсть к новому пригодятся вам даже в таких условиях, просто старайтесь разграничить личную заинтересованность в разработке и требования заказчиков.

Тезисы

- Разработчики любят код и новизну.
- Не играйте с кодом на работе.
- Разделяйте личный интерес и требования проекта.

Задание

Пофантазируйте. Представьте, что ваш текущий проект – «песочница» и здесь можно пробовать любые технологии и языки. Черт, да вы можете использовать по языку на каждый имеющийся у вас сервис! Подумайте, насколько сложно было бы совместить разные подходы к разработке, архитектуре, инструментам в одном проекте. Попробуйте оценить, насколько бы затянулась по времени реализация одной из ваших задач, если бы для этого использовался другой язык программирования.

История из жизни

Я люблю языки программирования, я люблю пробовать каждый, чтобы понять, каков он, чем он может быть интересен. И так как время мое не резиновое, мне всегда приходится искать способ уместить максимум

интереса в минимум времени. С какого-то момента, пробуя новый язык, я стал делать с его помощью «Франкенштейна» – небольшой проект, буквально несколько файлов, в которые старался уместить все особенности языка, используя их так, чтобы они делали хоть что-то осмысленное. Со временем это стало забавной традицией и незамысловатым способом поднять себе настроение.

Ошибки

Любой код содержит ошибки. С увеличением объема кода в нем увеличивается и количество ошибок. Вы можете быть гениальным разработчиком, но никогда не станете разработчиком, не допускающим ошибок. И это хорошо. Ошибки позволяют нам развиваться, делаться более бдительными и интуитивно чувствовать слабые места кода, потенциальные проблемы и способы их устранения.

Как разработчик вы должны писать код, содержащий минимальное количество ошибок. Пытаться классифицировать ошибки – пустая трата времени, поскольку ошибка может заключаться в чем угодно (вы поделили на ноль или на диске закончилось свободное место).

Почти все языки программирования предоставляют определенный уровень контроля ошибок, который вы должны использовать. Однако учитывайте, что во всем должна быть мера. Не нужно писать код так, чтобы на каждую строку было две проверки на ошибки. Соблюдение баланса в контроле ошибок приходит с опытом, когда вы будете предугадывать проблемные места и уделять им чуть больше внимания.

А пока старайтесь регулярно задавать себе вопросы: насколько вероятно, что данная ошибка произойдет? Какие последствия она будет иметь для продукта и пользователя?

Ответ на первый вопрос может быть достаточно непредсказуем (как вы помните, пользователи могут сделать что угодно), поэтому второй вопрос будет для вас более приоритетным. При любой ошибке желательно минимизировать урон для продукта и пользователя. Если существует вероятность, что в случае ошибки пользователь потеряет все свои данные, сделайте все, чтобы была возможность их восстановить. Если же в случае ошибки у пользователя изменится цвет заголовка его личной страницы, это тоже плохо, но вас хотя бы не сожгут на костре.

Старайтесь подходить к обработке ошибок прагматично, исходя из их потенциальной угрозы; не используйте больше контроля, чем необходимо. Возможно, это прозвучит не совсем логично, но вы ХОТИТЕ, чтобы определенные ошибки случались. Система без ошибок невозможна, и некоторые из них помогут вам лучше понять, с какими проблемами столкнулся продукт. Когда ошибки случаются, старайтесь собрать по ним всю доступную информацию, какую только можете получить (логи, stack traces, дампы данных), но, опять же, не пишите в лог каждое действие. В лучшем случае это просто оставит терабайтные файлы логов, в худшем – будет мешать работе пользователей или замедлять ваш продукт.

Тезисы

- Не бывает кода без ошибок.
- Используйте системы контроля ошибок с умом.
- Относитесь к ошибкам прагматично, некоторые из них вам нужны.
- Спрашивайте себя: как эта ошибка может случиться? Какой вред она причинит продукту и пользователю?

Задание

Проанализируйте код вашего проекта, узнайте, каким образом вы контролируете потенциальные ошибки. Оцените, какие области кода требуют больше внимания (если у вас есть подсистемы, работающие с денежными переводами пользователей, я очень выразительно смотрю в их сторону).

История из жизни

Моя самая дорогая ошибка обошлась бы мне в 3 миллиона рублей плюс сильный удар по репутации. Я занимался разработкой системы биллинга, и из-за досадной цепочки моих недосмотров были утеряны данные платежей

пользователей за сутки. Ситуация складывалась весьма неприятная, но у меня был запас времени: до момента, когда компания собиралась сделать заявление и рассылку с извинениями, оставалось два выходных дня. Это были крайне непростые два дня, за которые я все же сумел придумать и написать решение, способное восстановить историю покупок, хоть и с опозданием. Старайтесь воспринимать все свои ошибки серьезно. Какие-то обойдутся вам легко – только в то, что, обнаружив их, вы скажете «ОЙ». А другие станут очень ценным (иногда буквально) опытом, про который вы никогда не забудете.

Паттерны проектирования

Использование паттернов проектирования в коде – давний предмет споров и стычек на рабочем месте. Особое место в спорах эта тема занимает из-за того, что разработчики очень любят декомпозицию и качественные, элегантные решения. Если бы в разработке программного обеспечения был свой пророк Моисей, то, вероятно, с горы Синай он спустился бы с паттернами проектирования. Вам нужно будет хотя бы поверхностно ознакомиться со списком паттернов, чтобы понять, о чем я буду вести речь (да, вам опять понадобится Google).

Появление концепции паттернов проектирования для разработки программного обеспечения было предрешено. Эта область как никакая другая подходит для попытки классификации типичных проблем и их решений. Паттерны проектирования дают разработчикам способ унификации своего кода для ряда повторяющихся проблем, делают его более единообразным, узнаваемым и интуитивно понятным. Но до определенного предела. Опасность использования паттернов проектирования кроется именно в их универсальности и абстрактности по отношению к проблеме. Как говорится, если у вас в руках молоток, все начинает напоминать гвозди.

Начинающие разработчики частенько влюбляются в паттерны проектирования и пытаются использовать их везде, где только можно. Я постараюсь уберечь вас от этого.

Поймите меня правильно, паттерны проектирования – прекрасная практика, вы можете использовать их, и в некоторых ситуациях от вас будут ждать именно такого подхода. Однако вы должны отчетливо понимать, для чего вы это делаете и действительно ли выбранный вами паттерн подходит под вашу проблему.

Не относитесь к паттернам проектирования как к серебряной пуле, у вас есть достаточно способов решения проблемы, и только они и ваш опыт подскажут, какое решение подойдет лучше всего. Красота и элегантность кода – это не все, что требуется от качественного продукта (чуть позже мы обсудим это в теме про оптимизацию).

Будьте рассудительны. Если перед вами гвоздь – пожалуйста, беритесь за молоток, но, если вы не уверены до конца, потратьте чуть больше времени, принесите все инструменты и проверьте, что подойдет лучше всего.

Тезисы

- Паттерны проектирования – благо и проклятие.
- Не используйте паттерны проектирования для решения всех на свете проблем.
- Подходите к выбору решения прагматично, хладнокровно и осознанно.

Задание

Ознакомьтесь с паттернами проектирования и проанализируйте код вашего проекта: используются ли на нем какие-то из описанных паттернов? Выясните, есть ли в вашем проекте места, которые идеально подходят под описание проблемы одного из паттернов проектирования. Сделайте рефакторинг, реализовав выбранный вами код в рамках выбранного паттерна. Насколько новый код представляется вам лучше старого? Стал ли код более читаемым, лучше расширяемым или интуитивно понятным?

История из жизни

Мое первое знакомство с паттернами программирования было шапочным, и не могу сказать, что я был впечатлен. На сколько-то лет я забыл об их существовании, пока не понял, что многие из написанных мной

решений принимают определенную, повторяющуюся форму. После чего я решил освежить свои знания и при повторном знакомстве с паттернами хихикал, отмечая в голове те, к которым пришел с опытом. Мог бы я сберечь себе время, если бы уделил паттернам должное внимание при первом знакомстве? Безусловно. Получил бы я столько опыта, если бы не забыл их и не дошел до них практическим путем? Боюсь, что нет, не получил бы.

Переабстракции

Большинство языков программирования дает обширный набор конструкций, позволяющих при работе над кодом создавать сложнейшие абстракции. Их может быть так много, что сам анализ задачи, возможно, превратится в нескончаемый процесс подбора комбинаций из интерфейсов, поздних связываний, абстрактных классов и виртуальных функций. Ведьмин круг, из которого вы не сможете выбраться.

Идентичность языка определяется, помимо прочего, его подходом к предоставлению свободы разработчику. Некоторые языки программирования дадут вам десяток способов поразному совершить одно и то же действие (C++, как здорово, что ты зашел). От других вы дождетесь в лучшем случае двух (Go, заходи, присаживайся). Какой подход предпочтительнее, покажет опыт, но если вы уже работаете с проектом на конкретном языке программирования, то вам остается одно: следовать принятым соглашениям о способах использования возможностей языка. В худшем случае ваши коллеги никак не обсуждают этот вопрос и каждый из них пишет код так, как считает правильным (о, эта дивная кроличья нора!).

Однако у вас есть возможность помочь себе и коллегам: абстрагируйте код ровно до той степени, пока он не начнет делать то, что указано в требованиях к нему. Ваша работа – не соревнование «кто напишет самый абстрактный, гибкий (и неподдерживаемый) кусок кода на всем проекте». Этот код будут поддерживать ваши коллеги – пожалейте их, не всем нравится продираться через хрустальные замки интерфейсов и дюжины классов с множественным наследованием.

Если вы ожидаете, что ваш код будет часто и значительно меняться в ближайшем будущем, дайте себе чуть больше свободы в абстракциях, но не бросайтесь в омут с головой. В случае необходимости будет проще переписать этот код с нуля, чем тратить время и нервы на попытки понять, как же он должен работать. Во всех остальных случаях – keep it simple[2]! Если вы чувствуете непреодолимую жажду сотворить нечто невероятно сложное из всех конструкций языка, до которых сможете дотянуться, – создайте свой самый запутанный и понятный только вам pet project, он точно вас не разочарует.

Тезисы

- Разные языки программирования предоставляют различный уровень абстракций.
- Используйте ровно тот уровень абстракций, который работает для вашей задачи.
- Пишите просто – этот код придется поддерживать вам и вашим коллегам (а они могут знать, где вы живете).

Задание

Узнайте, какой уровень абстракций предлагает язык программирования вашего проекта. Проанализируйте код проекта: есть ли негласные правила использования тех или иных возможностей языка разработчиками? Попробуйте решения, которые можно выполнить с меньшим уровнем абстракций, сделав код более интуитивным и простым.

История из жизни

[2] Keep it simple, stupid («Делай проще, тупица»), или просто KISS – принцип проектирования, который утверждает: большинство систем работают лучше всего, если остаются простыми.

Однажды при найме нового разработчика я получил от него тестовое задание, которое включало в себя довольно простую задачу: в ней требовалось реализовать веб-приложение, позволяющее редактировать текстовые заметки. Обычно кандидаты ограничиваются минимальным набором необходимых классов и компонентов, но на сей раз это оказалась целая заготовка большой системы, состоящей из интерфейсов, трейтов, абстрактных классов, фабрики заметок и прочего. При личной встрече я первым делом задал вопрос: «Почему вы решили так усложнить задание?» На что получил более чем достойный и ожидаемый (в душе) ответ: «Я просто готовился к тому, что эту систему придется в дальнейшем расширять».

Оптимизация

Оптимизация – это способ сделать уже написанный код более быстрым, менее требовательным к ресурсам и более отвечающим нуждам задачи, которую он выполняет. Оптимизация возможна почти для любого написанного кода, но в большинстве случаев она нужна только в конкретных местах проекта.

Задачи оптимизации кода, его быстродействия крайне важны, и отчасти вы уже оптимизируете код, даже не осознавая этого. Многие языки программирования включают механизмы оптимизации в свои инструменты: в компиляторы, интерпретаторы и среды выполнения. Эти формы оптимизации помогают коду даже без вашего ведома (я очень рекомендую прочитать, как именно оптимизирует себя ваш язык программирования, это будет очень полезно).

Разные области разработки программного обеспечения требуют разной степени оптимизации. Есть определенный баланс: оптимизируя код, вы часто жертвуете стройностью его логики, красотой реализации, удобством архитектуры или частью функций системы.

Из этого следует первое правило: никогда не начинайте оптимизацию до того, как код будет удовлетворять всем требованиям проекта. Дональд Кнут (пожалуйста, прочитайте про этого прекрасного человека) сформулировал это правило так: «Преждевременная оптимизация – корень всех зол»[3] – и был, несомненно, прав. В вопросах оптимизации необходимо идти от обратного: не от отсутствия кода к его появлению, а от существующего кода к его упрощению и ускорению.

Второе правило оптимизации: убедитесь в том, что вы оптимизируете нужный код.

Прежде чем приступать к любой оптимизации, следует как минимум сделать профилирование кода. Необходимо знать все медленные места проекта, все бутылочные горлышки (bottlenecks, да, ознакомьтесь с этим). Чем больше вы получите информации о проекте, тем лучше сможете проанализировать и спланировать дальнейшую оптимизацию.

Как уже говорилось, чаще всего оптимизация на готовом проекте представляет собой баланс: на одной чаше весов – скорость и безотказность работы после оптимизации, на другой – простота понимания кода, удобство его использования или часть функций, от которых придется отказаться. Вы не сможете склонить чашу весов ни в одну из сторон, пока проект не будет удовлетворять всем требованиям и пока вы не будете знать, какие именно места в системе делают ее медленной и неоптимальной.

Вопрос о необходимости оптимизации также зависит от того, что именно вы собираетесь ускорить и чем можете пожертвовать, чтобы сохранить время, нервы и волосы вашего продакт-менеджера. Допустим, вы разрабатываете сайт, который позволяет одним пользователям загружать фотографии себе в ленту, а другим – просматривать их (нет, мне это ничего не напоминает, а вам?).

Вы провели профилирование и выяснили, что в системе есть два медленных места: из-за сложной обработки данных замедляется регистрация новых пользователей, а из-за некорректно выбранного формата хранения довольно медленно идет выдача загруженных фотографий. Очевидно, что для пользователей выдача фотографий гораздо важнее и оптимизация требуется в первую очередь здесь. В реальных проектах приоритет оптимизации не всегда будет определяться с такой очевидностью, поэтому необходимо тщательно подходить к проблеме выбора.

Оптимизация часто приводит код к менее логичному виду, иногда усложняет его, иногда чрезмерно упрощает, лишая его абстракций, элегантных решений. Это нормально: задача

[3] "Structured Programming with go to Statements," *Computing Surveys* (Vol. 6, № 4, декабрь 1974, с. 268).

оптимизации – сделать код максимально продуктивным, и у этого есть цена, которую вы должны заплатить.

Технически оптимизация будет напрямую зависеть от того, каким языком программирования вы пользуетесь, какие применяете библиотеки и компоненты, какие инструменты работают в проекте. Возможно, библиотека, которую вы используете, медленная. Возможно, вы неоптимально используете какой-либо из ресурсов системы. Возможно, вы где-то забыли отладочный код, замедляющий проект (ай-яй-яй).

Каждая оптимизация уникальна. В одном случае это будут некорректные индексы в базе данных, которые вы поправите одним запросом. В другом вы уткнетесь в несовершенство операционной системы, в рамках которой работает проект, и потратите не один день на поиск способа сделать код быстрее. Оптимизация – это чаще всего не набор конкретных действий, а творческий подход к коду в попытке сделать его более производительным.

Тезисы

■ Не надо оптимизировать весь код.

■ «Преждевременная оптимизация – корень всех зол».

■ Выделяйте приоритеты оптимизации.

■ Изучайте и используйте технические способы оптимизации.

■ За оптимизацию всегда надо платить (логичностью кода, удобством, потерей функций).

Задание

Профилируйте проект, определите места, которые требуют оптимизации (если их не обнаружилось, значит, либо у вас идеальный проект, либо вы плохо искали). Попробуйте расположить найденные медленные места в порядке убывания важности исходя из функций вашего проекта: что должно быть оптимизировано сразу? Что можно отложить на потом? Попробуйте оптимизировать участок кода, в котором вы разбираетесь лучше всего. Проведите профилирование еще раз, чтобы убедиться, что новое решение более производительно. Проанализируйте изменения и скажите, стал ли код более читаемым, стал ли более сложным в дальнейшей поддержке, не пришлось ли вам срезать несколько углов и избавиться от чего-то не слишком важного ради скорости выполнения?

История из жизни

История, вспоминая которую можно и усмехнуться, и всплакнуть. В одном из проектов мы действительно провели невероятную «оптимизацию», но скромно умолчали о ней. В нескольких критически важных местах системы, которые должны были выполняться очень быстро, обнаружились оставленные кем-то из разработчиков отладочные вызовы функции sleep. Эта функция останавливает выполнение программы на указанное время, тем самым замедляя любой процесс на указанное время. В официальных отчетах мы отметили, что скорость работы приложения повышена, однако предпочли не раскрывать, каким именно образом. В любом случае это был самый быстрый и качественный способ оптимизации, с которым я сталкивался. Удалить несколько строк кода и получить немедленный прирост скорости – не об этом ли мечтает каждый разработчик?

Люди

Этот раздел – об общении с людьми, о том, как выстраивать деловые отношения, как приспособиться к работе в коллективе. Я затрону социальные вопросы, проблемы баланса между работой и общением.

Разработчики постоянно общаются с разными людьми: с коллегами, начальством, а также с теми, кто максимально далек от разработки программного обеспечения и с трудом отличает браузер от операционной системы. Вы должны уметь найти верный способ объяснить им, чем вы занимаетесь. Навык общения крайне важен для профессионального (а еще в большей степени карьерного) роста.

Как разработчик вы в первую очередь должны уметь писать код. Но этого мало. Умение вести диалог, объяснять свои мысли, предлагать собственные решения и высказывать несогласие – все это даст вам огромные возможности профессионального роста.

Контекст и коммуникация

Думаю, все сталкивались с ситуацией, когда вы работаете над сложной задачей и тут вас внезапно о чем-то спрашивают. На считаные секунды вы отвлекаетесь, отвечаете на вопрос, возвращаетесь к работе и никак не можете сообразить, чем только что занимались. Вас выдернули из рабочего контекста (и вряд ли вас это обрадовало).

Ремесло разработчика сложное, это факт. В голове у вас должны умещаться абстракции, логика работы проекта, особенности требований, подходящие алгоритмы, потенциальные ошибки и многое-многое другое. Все это требует внимания и сосредоточенности, которые так легко потерять, если внезапно задуматься на постороннюю тему.

Вы должны оберегать себя от внешних раздражителей. Наушники, режимы «do not disturb» в программном обеспечении, отдельный кабинет или комната, где вас никто не станет дергать. Чем больше вы работаете, тем лучше понимаете, как обеспечить себе максимальный комфорт на рабочем месте.

На многих проектах полагаются обязательные митинги (совещания), обсуждения или иные способы коммуникации. Иногда это вызвано методологией разработки, иногда избыточным количеством менеджеров, которым больше нечем заняться. Часто отказаться от участия в них просто нельзя, а порой они даже бывают полезны, но помните: ваш основной приоритет – работа над кодом.

Если вы понимаете, что излишняя коммуникация только вредит вашей работе, обсудите это с менеджером или старшими разработчиками. Скорее всего, вы сумеете договориться о возможности избегать непродуктивного общения, чтобы сосредоточиться на работе.

И наоборот: если вы понимаете, что регулярные обсуждения приносят вам пользу, позволяют лучше видеть мотивацию команды или коллег, – участвуйте в них. Правильным решением будет только то, которое подходит лично вам. Возможно, на этом этапе вам станет яснее, в каком направлении вы предпочтете развивать карьеру: будет ли это, например, совершенствование технических навыков или организация процессов разработки.

Это же относится и к общению в рабочих чатах, мессенджерах или в офисе: определите для себя комфортный уровень коммуникации и придерживайтесь его – отключайте чаты, попросите коллег не беспокоить вас, если понимаете, что они мешают сосредоточиться.

Тезисы

■ Оберегайте свое спокойствие и рабочий контекст.

■ Проанализируйте, нравится вам общение с коллегами или оно в основном лишь отвлекает от работы.

Задание

Попробуйте определить, какой уровень коммуникации на проекте вас больше всего устраивает. Старайтесь соблюдать баланс между полезной коммуникацией и сосредоточенностью на работе. Если вас отвлекают, не стесняйтесь сказать об этом. Обратите внимание и на коллег: какой уровень коммуникации для них предпочтителен?

История из жизни

Не буду скрывать – я всегда избегал лишнего общения, будь то состязание в остроумии у кулера или еженедельные совещания (о ежедневных летучках при мне даже не упоминайте). В работе я всегда был продуктивен только в одном состоянии – когда писал код. Но когда я начал руководить

разработкой проектов, общению все же пришлось уделять больше внимания. Зато со временем я научился видеть в этом пользу.

Десять раз спроси, один – напиши

Разработчик почти всегда находится в поиске более качественного решения задачи, над которой работает. Если вы пишете код исключительно для себя, это отчасти упрощает задачу (или значительно усложняет, спасибо тебе, проклятый перфекционизм). Однако если вы разрабатываете код по чьим-то требованиям, то почти всегда будете располагать неполной информацией. У вас будет масса вопросов, непонятных или не до конца известных условий и особенностей продукта, которые необходимо выяснить.

Никогда не бойтесь и не стесняйтесь спрашивать обо всем, что считаете нужным при решении задачи. Нет ничего более полезного, чем полная информация о проблеме, которую вам необходимо решить. Любое предположение, не подкрепленное верными данными, может обернуться ошибкой. Любое белое пятно в требованиях может привести к необходимости выкинуть половину готового кода и начать писать его заново.

Обилие вопросов может смутить менеджеров, заказчика или старших разработчиков, но помните: вы занимаетесь разработкой не для того, чтобы о вас хорошо или плохо думали, а ради качества продукта, ради профессионализма и карьеры. Хирург не начинает операцию без знаний и опыта, точно так же и вы не можете приступить к работе, не выяснив досконально, с чем придется столкнуться.

Старайтесь упростить себе работу. Не заданный вовремя вопрос замедлит выполнение задачи либо в процессе самого решения, либо позже, когда вам надо будет вносить дополнения и изменения, которые, в свою очередь, могут спровоцировать новые ошибки.

Тезисы

- Спрашивайте.
- Спрашивайте.
- Спрашивайте.
- ОБЯЗАТЕЛЬНО спрашивайте.

Задание

Для следующей сложной задачи составьте максимально подробный список вопросов, постарайтесь представить общую картину проблемы и ее потенциального решения. Задайте эти вопросы тому, кто поставил задачу. Попробуйте заранее определить, какие дополнительные сложности могут вас ждать.

История из жизни

Один раз я взял небольшой (и потенциально быстрый) заказ на разработку сайта с астрологическими прогнозами. Сказать, что я задолбал заказчика вопросами о том, как устроены гороскопы и от чего будет зависеть выдача пользователям, значит не сказать ничего. В силу гороскопов я так и не поверил, но у меня еще долгое время на столе лежал ворох бумаг с выкладками по знакам зодиака и воздействию на каждый из них положения Луны.

Критика и критиканство

Если вы начинающий разработчик или только собираетесь вступить на эту стезю, будьте готовы к критике и к тому, что в ваших решениях станут сомневаться. Это закалка, которую вы должны получить. Не все ваши решения в начале карьеры будут правильными и качественными, это нормально. Вы накапливаете опыт, который в итоге и сделает вас профессионалом.

Вам придется услышать немало критики, и сразу оговорюсь: корректная критика – это то, что помогает стать лучше, не задевает самооценку и способствует профессиональному росту. Очень важно отличать критику от критиканства. Замечайте, когда вас используют, чтобы подкрепить свое нездоровое эго или самоутвердиться за ваш счет. Такие ситуации вряд ли будут частыми, но нужно быть готовым и к ним.

Любую критику необходимо воспринимать только как стимул улучшить свои навыки. Никогда не принимайте ее близко к сердцу. Ваша личность никак не связана с профессиональными качествами, работа не связана с вашим «я». Берите в расчет только то, что относится к качеству работы, и пропускайте мимо ушей все, что каким-либо образом затрагивает вашу личность.

Если вы понимаете, что в коллективе из вас делают козла отпущения или же критика в ваш адрес относится не к качеству работы, а к вам как к человеку, постарайтесь обсудить это на более высоком уровне, непосредственно с руководством. Если это не приносит результата, увольняйтесь. Опыт вы получите в любой компании, но никогда не стоит работать там, где вас используют только для того, чтобы тешить свое больное эго.

В некоторых случаях критику будут оправдывать тем, что это «жесткая любовь»: дескать, вас заставляют работать на износ, критикуя практически каждый шаг, чтобы сделать из вас «настоящего спартанца». Никогда не верьте в это. Любая компания, практикующая такое, – не ваша компания (бегите, срочно бегите!). В таком месте вы не сможете развиваться как профессионал. Все, что вы получите, – невроз и комплекс неполноценности.

Тезисы

■ Слушайте и воспринимайте только конструктивную критику, а не критиканство.

■ Меняйте работу, если понимаете, что из вас делают козла отпущения.

■ Никакой «жесткой любви», вы достойны большего. Или поступите как «настоящий спартанец»: сбросьте вашего менеджера со скалы.

Задание

Проанализируйте критику в свой адрес: всегда ли она направлена на объективные проблемы, связанные с кодом, над которым вы работаете? Очевидно, что в начале карьеры какие-то вещи вам придется принимать на веру. У вас еще недостаточно опыта, чтобы оценить дальние перспективы, но всегда можно понять, когда критики переходят на личности. Сравните, как люди оценивают вашу работу и работу ваших коллег: есть ли разница? Критикуют ли вас чаще или реже, чем остальных?

История из жизни

В начале своей карьеры я работал в компании, которую возглавлял бывший военный. Человек он был очень резкий, стремившийся контролировать все и вся и весьма неадекватный. К сожалению, тогда у меня не было примера хорошей компании и адекватного руководства, и я проработал полтора года в атмосфере постоянной муштры и напряжения,

о чем теперь жалею. Полтора года жизни – это много, не стоит сжигать свое время в топке чужих комплексов.

Пользователь всегда прав

Пользователи. М-м, как много в этом слове. Если вы не разрабатываете автоматизированный сервис, взаимодействующий только с другим программным обеспечением, то у вашего продукта есть пользователи. Это могут быть крупные компании или только ваши родители – неважно. В любом случае вы должны о них заботиться.

Давайте я сразу раскрою основную мысль этой темы: пользователь всегда прав. Всегда. Даже тогда, когда он не прав. Поймите меня правильно: при общении с пользователями в 8 случаях из 10 вы будете сталкиваться с тем, что именно пользователь сделал какую-то глупость, а код работал так, как и должен был работать. Однако вы ВСЕГДА должны считать, что пользователь прав, и проверять каждую ошибку как можно внимательнее.

Пользователи часто будут к вам несправедливы (черт, да они даже могут оскорблять вас в баг-трекерах). Они станут писать глупости или не смогут объяснить, какую именно ошибку получают. В любом случае пользователи – это ваши самые верные помощники. Пользователи – это те, ради кого вы создаете продукт, это ваш фронт, ваша зона боевых действий. Только они имеют дело с продуктом так, как это было задумано при разработке.

Вы должны с особым вниманием относиться к комментариям, которые пишут пользователи, к сообщениям об ошибках, которые они вам отправляют. Необходимо тщательно собирать и анализировать информацию: как они используют ваше приложение, с какими неудобствами сталкиваются, чего им не хватает, что их раздражает.

Реальность будет расходиться с тем, о чем я здесь пишу. Пользователи будут вас раздражать, доставать дурацкими просьбами, сообщениями об ошибках, которых вы не сможете воспроизвести, и просто посторонними вопросами, но постарайтесь быть сильным. Помните: именно какой-нибудь пользователь однажды сообщит вам о серьезной проблеме, которую проглядели и вы, и ваши тесты, и тестировщики. Он один будет стоить сотни пользователей, которые сделали глупость сами.

Когда вам понадобится улучшить UX вашего продукта, соберите аналитику по разным пользователям, узнайте их мнение. Да, вы работаете над продуктом каждый день, и вам КАЖЕТСЯ, что вы лучше всех знаете, как использовать его наиболее эффективно, но это всего лишь иллюзия.

Вы удивитесь, насколько по-разному можно работать с одной и той же системой со стороны разработчика и пользователя. Доверьте право выбора удобства пользователям: ваша зона ответственности – код, их – использование конечного результата.

Тезисы

- Пользователь всегда прав.
- ВСЕГДА.
- Внимательно относитесь к каждому сообщению об ошибке от пользователей, прислушивайтесь к их комментариям.
- Доверяйте пользователям в вопросах удобства работы с вашим продуктом.

Задание

Если у вас есть система баг-репортов от пользователей, проанализируйте список обнаруженных ошибок. Проверьте: возможно, некоторые ошибки имеют между собой что-то общее. Постарайтесь исправить одну из них. Если у вас есть система фидбеков, попробуйте поработать с ней, отвечая пользователям на их запросы или сообщения об ошибках.

История из жизни

Я был очень упертым разработчиком. Долгие годы я считал, что точно знаю, как работает мой код и как удобнее всего использовать мой продукт. Отучили меня от этих заблуждений опыт и болезненные щелчки по носу. Ты несколько дней игнорируешь сообщение от пользователя, считая, что он делает что-то неправильно, а потом выясняется, что это у тебя глупейшая ошибка, – ЩЕЛК. Ты приходишь к клиентам настраивать софт и видишь, как пользователи тратят время на неудобный интерфейс, который ты считал весьма подходящим для себя, – ЩЕЛК. Доверяйте пользователям, следуйте их пожеланиям, если это возможно.

Это МОЙ код

С ростом опыта и продвижением карьеры необходимо развить в себе способность (и уверенность) отстаивать свой код и выбранные решения.

Очевидно, что на первых порах вам придется больше слушать и писать, чем предлагать и отстаивать. Но в любом случае настанет момент, когда вы будете знать, что вы правы, а коллеги – нет.

Отстаивать свои решения одновременно и легко, и сложно. Легко, потому что у вас, как у опытного специалиста, всегда будет объективная и корректная аргументация, помогающая доказать предпочтительность вашего подхода. А сложность состоит в том, что люди не всегда готовы прислушиваться к кому бы то ни было. Вы можете быть десять раз правы, а аргументация идеальной, но если вас не хотят услышать, все это будет зря.

Как и в любых вопросах общения, старайтесь соблюдать здоровый баланс: последовательно отстаивайте свою позицию, подавляя желание сделать «розочку» из бутылки и начать угрожать коллегам. А если говорить серьезно, не позволяйте обесценивать ваш труд. Вы вложили в свой код время, силы, личный опыт; вы уверены в его качестве. Не стоит все это предавать. В спорной ситуации будьте настойчивы, используйте объективные данные: цифры, сравнения, ссылки на статьи и лучшие практики.

Но не менее важно понимать, когда стоит остановиться в споре. Помните: ваша задача – создавать качественный код, и если кто-то предлагает более подходящее и рациональное решение, не упрямьтесь, будьте честны с собой. Если предложенное решение лучше вашего, проанализируйте его и примите с благодарностью. Так вы получите куда больше опыта и возможностей роста, чем если будете спорить просто ради спора.

Тезисы

- Отстаивайте свой код.
- Будьте гибкими, но не предавайте свой труд.
- Выбирайте лучшее решение, даже если оно не ваше.

Задание

Задание для этой темы найдет вас на работе само, поверьте. Будьте гибкими, но никому не позволяйте пренебрегать вашим трудом и опытом.

История из жизни

За годы работы меня бросало из крайности в крайность. Вначале я не умел и не считал нужным отстаивать свой код, а став профессионалом, не принимал ровным счетом никакой критики. Мне стало легче, когда я понял, что не отделяю себя лично от работы, которую выполняю. Любая критика или комментарий к коду воспринимались как личный упрек, как личное оскорбление. В моем случае это была плата за полноту, с которой я отдавался профессии. И по сей день мне приходится напоминать себе, что я – это я, а не моя работа. Это помогает.

Это МОИ деньги

Если говорить о карьере, совершенно невозможно обойти вниманием вопрос оплаты труда. С какой бы любовью вы ни относились к своей работе и коду, вам нужно что-то есть, одеваться, оплачивать свои хобби, и да, мы все еще живем в капиталистическом мире.

Вопрос оплаты труда актуален не только с точки зрения достатка и возможности оплачивать счета. Ваша ставка – это то, как вы оцениваете свою работу, то, насколько доверяете себе как профессионалу. Оценивать себя всегда непросто, и, чтобы объективно «измерить» свой профессионализм, необходим опыт. Не ждите, что уверенность и объективность в этом вопросе придут к вам сами по себе.

В начале карьеры у вас будет очень мало возможностей влиять на размер зарплаты. Ваш небольшой опыт играет на руку компаниям, а они обычно заинтересованы в вас не настолько, насколько вы заинтересованы в них. Это нестрашно. Ваша задача на данном этапе – найти по-настоящему комфортное место работы, где вы сможете развивать профессиональные навыки, набираться опыта, улучшать резюме и получать новые знания.

Главное, что должно занимать вас в начале карьеры, – это профессиональный рост. Необходимо использовать любое место работы или проект для получения новых знаний и навыков. Это даст толчок всей вашей дальнейшей деятельности, если вы вправду хотите стать профессиональным разработчиком.

Должность начинающего разработчика выглядит не очень завидно, однако в ней есть большой плюс: с вами действительно будут делиться знаниями – иногда в силу желания ощутить превосходство, иногда искренне. Минус же в том, что компании не особо заинтересованы в начинающих разработчиках, если только не собираются оставить их надолго и «вырастить» под себя. Последнее, кстати, не самый лучший сценарий, потому что компания будет формировать ваши знания под свои нужды, проекты и требования. Маловероятно, что это станет для вас полезным опытом (если только речь не идет о какой-нибудь очень специфической индустрии, где требуется очень конкретный набор знаний).

Итак, опыт – вот чем должна полностью определяться ваша зарплата. Поначалу придется мириться с тем, что вам диктуют, сколько вы «стоите». Но это не повод для огорчения. С каждым годом ваши профессионализм и опыт будут расти, и со временем вы сами сможете определять условия работы, потому что теперь уже компания будет заинтересована в сотрудничестве. Ваши знания, умения и опыт – тот ресурс, который вы продаете.

Как только вы почувствуете, что можете объективно назвать стоимость своей работы, ни в коем случае не соглашайтесь на меньшее. Вы абсолютно не заинтересованы в том, чтобы ваш труд недооценивали. Если в компании есть регулярные повышения оклада – прекрасно, но в первую очередь доверяйте себе. Реальность такова, что компании нет никакого смысла платить вам больше, если вы выполняете свою работу за те деньги, которые уже получаете.

Ведите учет своих успешных проектов и профессиональных достижений. Этот список будет важным аргументом при обсуждении оплаты труда на собеседовании или при повышении на текущем месте работы. Вы требуете больше денег не просто потому, что так хотите, а предоставляете объективные аргументы в пользу того, что ваш труд ценен.

Тезисы

- В начале карьеры придется набраться терпения: вам нужен опыт.
- Ваша «стоимость» на рынке будет определяться навыками и опытом; как только вы поймете свою истинную цену, ни за что не соглашайтесь на меньшее.
- Ведите учет своих профессиональных достижений.

Задание

Представьте, что завтра состоится обсуждение качества вашей работы в прошедшем году и повышения зарплаты. Какие аргументы вы приведете, чтобы вопрос о повышении не вызывал дополнительных сомнений? Какие профессиональные достижения за последний год сможете перечислить? Как вы построите диалог, если вам предложат совсем незначительное повышение или скажут, что недовольны вашей работой?

История из жизни

Достаточно много лет я не умел оценивать свою работу, не знал, как правильно выстраивать диалог об оплате труда. Отчасти из-за того, что много сомневался в своих силах и опыте, отчасти потому, что деньги никогда не были для меня основным мотиватором. Я всегда любил разработку, любил программировать, любил сам процесс. Бывали месяцы, когда я забывал отправить платежные документы, чтобы получить заработанные деньги, просто потому что был слишком увлечен проблемой, которую решал. Это не самое лучшее отношение к своему труду и времени. Наше время очень ограниченно, а деньги позволяют проживать его с комфортом. Если бы я мог вернуться в прошлое и дать совет себе начинающему, этот совет был бы таким: никогда не бояться говорить о деньгах и соглашаться только на ту сумму, которая соответствует моему опыту.

Сильные и слабые стороны

Будем честными: никто не состоит из одних достоинств (и спасибо Вселенной, что это так, иначе было бы чудовищно скучно). Впрочем, сейчас я хочу поговорить не о том, насколько вы хороши как личность, а о вас как о профессионале.

Навыки коммуникации с коллегами и клиентами определяют ваш профессионализм не меньше, чем качество кода, который вы создаете. Понимание своих личностных особенностей – непростая задача. Мы биологически запрограммированы на желание нравиться окружающим (да, даже если вы носите ирокез и считаете себя отчаянным нонконформистом).

Трудно честно ответить себе на вопрос, каковы ваши сильные стороны, а каковы слабые. Ваше внутреннее «я» будет сопротивляться, выдавая желаемое за действительное. Вы станете обманывать себя: превращая сильные стороны в нейтральные или даже слабые. Не торопитесь, дайте себе время. Постарайтесь анализировать отстраненно: что бы вы сказали про себя, будь вы своим коллегой?

Отвечайте себе честно. Если будете лукавить, пользы это занятие принесет мало. Вы полностью уверены в своих технических знаниях и умениях, но легко идете на конфликт и общаться с вами порой невыносимо? Прекрасно, теперь вы знаете, что вам лучше сосредоточиться на работе, а не на общении с коллегами. Вы поняли, что лучше всего умеете сглаживать конфликты внутри коллектива, но не способны отличить абстрактный класс от интерфейса? Отлично, значит, вы знаете, каких технических знаний вам не хватает, а может, даже задумываетесь о смене профиля деятельности, раз ваши слабые стороны мешают получать удовольствие от работы.

Используйте свои сильные стороны – полагайтесь на них, доверяйте им. Вы умеете писать очень качественный код? Наверняка об этом знаете не только вы, поверьте. Используйте это как козырь, развивайте этот навык. Вы отлично умеете декомпозировать задачи, видеть частности в целом? Применяйте это в своей работе, помогайте улучшить продукт, претендуйте на голос в принятии решений по проекту и его архитектуре.

Возьмите листок бумаги и честно запишите свои плюсы и минусы, составьте список тех черт, которыми вы гордитесь, и тех, которыми недовольны. Попробуйте внимательно посмотреть на них: что они говорят о вас?

Возможно, в списке слабых сторон окажется и нежелание разбираться в собственных слабостях. Тогда можно решить, чем вам нравится и хочется заниматься, исходя из списка сильных сторон.

Список может помочь вам лучше понять, чего вы хотите от своей карьеры и профессионального роста. К примеру, если список слабых сторон отчетливо демонстрирует недостаток технического мастерства, но при этом из списка сильных сторон явствует умение общаться с людьми, задайте себе вопрос: действительно ли вы хотите продолжить строить карьеру технического специалиста? Не стоит ли попробовать себя в роли продакт-менеджера или руководителя собственной IT-компании?

Если же ваши слабые стороны – это просто список вопросов, которым вы уделяли мало внимания, тогда все отлично: можете совершенствоваться.

Невозможно сохранять баланс, имея только сильные или только слабые стороны. Более того, в попытке избавиться от слабых сторон мы можем потерять и сильные. Не пытайтесь быть идеальным, это путь к неврозу и постоянному стрессу, но честно признавайте свои достижения и провалы. Относитесь к этому отстраненно. Это не то, за что следует винить или превозносить себя. Это просто черты, которые вы в себе видите. То, что вы можете использовать в своих интересах, и то, чего можно избежать, зная о существовании проблемы.

Тезисы

■ Для определения своих сильных и слабых сторон нужно время.

■ Постарайтесь оценивать себя как бы со стороны.

■ Будьте максимально честны по отношению к себе.

■ Соблюдайте баланс: подтягивайте слабые стороны, не запускайте сильные.

■ Используйте свои сильные и слабые стороны в собственных интересах.

Задание

Попробуйте проанализировать последние полгода своей работы. Есть ли то, с чем вы справляетесь лучше, чем коллеги? Все что угодно: технические задачи, общение в коллективе, организация событий внутри компании, способность убедить заказчика в чем-либо и т. д. А с чем вы испытывали сложности? Постарайтесь записывать все успехи и трудности за этот период. Есть ли что-то, объединяющее их?

История из жизни

В моей карьере был период, когда я настолько измотался, что начал спрашивать себя: а этого ли ты хотел? И доспрашивался до того, что мне захотелось попробовать себя исключительно в роли руководителя, так чтобы не вникать в технические аспекты работы. Скажу сразу: меня хватило на три недели совещаний, создания отчетов в Word и просиживания на встречах с потенциальными клиентами. Больше я себе таких глупых вопросов не задавал.

Интервью

Собеседования при приеме на работу – неотъемлемая часть жизни любого человека. Интервью будут случаться у вас не так уж часто (я надеюсь), но к ним нужно быть готовым. Обычно поиск вакансии и прохождение интервью занимают несравнимо меньше времени, чем дальнейшая работа в компании, в которой вы окажетесь. Однако важность интервью нельзя недооценивать, особенно если вы хотите попасть в конкретную компанию на определенную должность.

Интервью – процесс сложный, и, к сожалению, единого алгоритма его успешного прохождения нет. Общая атмосфера на интервью зависит от десятка факторов, начиная от страны вашего проживания и заканчивая тем, с каким настроением сегодня проснулся интервьюер (да, я знаю, что это непрофессионально с его стороны, но не хочу врать: мы говорим о реальном мире). Я просто перечислю важные пункты, относящиеся к любому интервью, но не могу сказать, какой из них будет решающим в вашем случае, поэтому постарайтесь учесть их все.

Внешний вид. Да, вы должны выглядеть опрятно и соответствовать принятому в компании дресс-коду. Однако если вам делают замечание по поводу внешнего вида или вы поняли, что он чем-то не устроил интервьюера, вряд ли стоит идти в такую компанию. Вас нанимают за знания и навыки, а не за внешний вид.

Соответствие требованиям. Если вы идете на интервью по вакансии, то уже знакомы с требованиями работодателя. Не пытайтесь хитрить или приукрашивать свои умения. Это может сработать, если у вас есть значительный опыт в индустрии и большой набор знаний в разных областях IT. Однако если вы только начинаете карьеру разработчика, просто постарайтесь иметь представление обо всех требованиях, перечисленных в объявлении о вакансии. Собеседования с начинающими разработчиками проходят достаточно быстро, и, чтобы сразу отсеять кандидата, не нужно даже привлекать технического специалиста: вам просто зададут стандартный набор вопросов по технологиям, перечисленным в вакансии. Если вы не знаете их на достаточном уровне, это выяснится либо сразу, либо позднее, в другой части интервью, где вас будет собеседовать технический специалист.

Подготовка. К любому интервью стоит подготовиться. Помимо того, что вы должны освежить свои знания по требованиям, указанным в вакансии, необходимо также ознакомиться с основной информацией о компании. Узнайте, какие продукты она выпускает, какие технологии при этом используются. Составьте базовое представление о компании, это будет дополнительным плюсом на интервью. Не относитесь к подготовке спустя рукава: если вы действительно хотите получить должность, то техническая подготовка к интервью – первый шаг по направлению к цели.

Уверенность в себе. На какую бы должность вы ни претендовали, неуверенность и сомнения в себе будут заметны с порога. Да, вы будете испытывать стресс, и некоторые компании только подстегнут его (в их воспаленных фантазиях это может лучше раскрыть кандидата, вот же странные люди). Не позволяйте эмоциям завладеть вами. Это просто работа, вы пришли не на суд, а рассказать о себе и своих знаниях. Если не знаете, что отвечать, – скажите об этом и попытайтесь предложить решение в той области, в которой разбираетесь. Большинство интервью направлено не на то, чтобы услышать от вас правильные ответы. Обычно работодатели хотят узнать, как вы подходите к проблеме, как пытаетесь решить ее, основываясь на своем опыте и умениях, – учитывайте это при подготовке к интервью. Не забывайте о своих сильных сторонах, равно как и о слабых (поверьте: когда вы говорите, что ваш самый большой минус – желание всем угодить, никто этому не верит, даже вы сами).

Помните: даже если вы идеально подготовились, вам могут отказать. И это не должно вас расстраивать. Совсем. В крупных компаниях имеется отдельный штат сотрудников, отвечаю-

щих за наем персонала, а в небольших вас может собеседовать просто старший разработчик; в любом случае отказ – не повод для огорчения. Может быть, интервьюеру просто не понравился ваш внешний вид (не уверен, что стоит приходить в эту компанию еще раз), а возможно, вам не хватило знаний или опыта – отлично, значит, есть куда расти. Можете попробовать пройти интервью в той же компании через год, с новым опытом и знаниями (и более высоким запросом по заработной плате).

Тезисы

- Ваш внешний вид на собеседовании должен соответствовать должности, на которую вы претендуете, и дресс-коду компании.
- Если вас оценивают только по внешнему виду – бегите.
- Изучите требования к кандидатам и не приукрашивайте свои знания.
- Не пожалейте времени на подготовку к интервью.
- Будьте уверены в себе, вы просто пришли устраиваться на работу.

Задание

Найдите компанию, в которой вы действительно хотели бы работать. Проверьте, есть ли у них подходящие вакансии. Попробуйте подготовиться к интервью: узнайте больше о требованиях и используемых в этой компании технологиях, составьте список своих сильных сторон и последних достижений на текущем месте работы. Если чувствуете в себе дух авантюризма и уверенность, договоритесь об интервью. Никто не заставит вас бросить текущий проект, но сам опыт прохождения интервью и, возможно, потенциальная должность как минимум поднимут настроение и придадут уверенности.

История из жизни

Я никогда не любил интервью и собеседования. Мне не нравились вопросы, задания, я никогда не мог вспомнить нужную сортировку и особенности ее реализации в нужный момент. Я не любил конкретные вопросы о том, что я «должен» помнить наизусть, это напоминало мне скучное заучивание, без вариативности и возможности придумать что-то на ходу. Стоит ли говорить, что с таким подходом я редко оказывался желанным кандидатом для самых разных компаний. Но был и плюс: благодаря такому отношению я попадал в другие компании, которые так же не принимали формализма в найме разработчиков, отдавая предпочтение их практическим навыкам. Моя нелюбовь к формальным собеседованиям привела меня к тому, что уже находясь по другую сторону стола я с куда большим пониманием и симпатией отношусь к разработчику, который хочет получить работу, и не пытаюсь выдавить из него реализацию алгоритма, которую он должен написать ПРЯМО СЕЙЧАС на бумажке (да, увы, я побывал и на таком собеседовании).

Если начальник – идиот

Вряд ли вы хотите, чтобы ваш будущий начальник оказался идиотом. Но это реальный мир, и эта ситуация так же вероятна, как и десятки других.

Следует оговориться: когда я пишу, что ваш начальник, возможно, идиот, я совершенно не беру в расчет то, какой он человек. Платит ли алименты своим бывшим женам и всегда ли придерживает вам дверь. Я говорю только о его профессиональных навыках и взаимодействии с подчиненными. Старайтесь исключать личное отношение, когда речь идет об общении в коллективе. Да, вы можете дружить с коллегами, но нельзя считать людей непрофессионалами только потому, что они не соответствуют вашим представлениям о хорошем.

Слово «начальник» можно понимать достаточно широко. Это может быть любой человек в коллективе, от которого зависит ваша работа, – директор, старший разработчик, проект-менеджер и т. д.

В начале карьеры у вас будет еще недостаточно опыта, чтобы с уверенностью сказать, насколько хорошо руководитель разбирается в вашей работе. К тому же многие из них умеют идеально маскировать свое незнание. Моя рекомендация по поводу авторитарных начальников, в знаниях которых вы сомневаетесь, проста: не работайте с такими людьми. Да, возможно, это будет стоить вам места, но поверьте мне: если вы цените свои нервы, рабочую мотивацию и хорошее настроение, избегайте подобных людей. Они никогда не признают ваших заслуг, будут критиковать вашу работу, ничего не понимая в ней, и даже вставлять палки в колеса, если вдруг у них появится такое желание. Уважайте себя, цените свой труд и ментальное здоровье.

В нашей индустрии немало отличных специалистов, которые ведут себя как первостатейные говнюки (я бы назвал пару имен, но хочу еще немного пожить). Да, работать с ними сложно, и да, они очень хорошо умеют делать свою работу. Иногда чаша терпения окружающих переполняется и все достоинства этих прекрасных специалистов меркнут перед их неумением (или нежеланием) общаться с людьми. Будьте уверены: в течение карьеры вам непременно выпадет поработать с несколькими такими людьми (или с несколькими десятками, если ваша карма так себе).

Тезисы

- Если вы поняли, что ваш начальник – идиот, уходите.
- Абсолютно серьезно – уходите.

Задание

К этой теме у меня для вас только одно задание. Всегда и везде избегайте идиотов, от которых зависит ваша работа. Любыми способами.

История из жизни

К счастью, в жизни мне попалось только несколько руководителей, с которыми я не смог работать. Вот, пожалуй, самая глупая из связанных с этим историй: в 3 часа ночи мне звонил начальник, находившийся в другом часовом поясе, и требовал объяснить, что делает код, который мы написали в течение дня (он не был разработчиком и о коде имел такое же представление, как я о глубоководной фауне). Нужно ли говорить, что время этого разговора не учитывалось при оплате в конце месяца? Как и в отношении денег, я довольно долго считал, что мне нужно подстраиваться под людей, чтобы иметь шанс быть частью какого-то продукта, но каждая такая ситуация в 3 часа ночи

хорошо отрезвляла и напоминала, что мой запас времени и нервов весьма ограничен.

Поиск виноватых

Итак, этот день настал. Ваше приложение упало, и упало сильно. Тысячи расстроенных пользователей, потеря данных, денег, громкий плач, доносящийся из отдела по работе с клиентами. Да, такое случается и однажды обязательно случится с вашим продуктом. То, что произойдет после, определит уровень вашей компании и людей, которые в ней работают.

Начнем с правила, которое вы просто должны запомнить. Поиск виноватых и охота на ведьм – дерьмо, которое никогда и никому не помогало. Если после сбоя продукта весь менеджерский состав и старшие разработчики заняты поиском того, кто допустил эту ужасную ошибку, чтобы покарать негодяя, я бы предложил вам серьезно задуматься о смене места работы.

Ошибки – это постоянный побочный эффект нашей работы. От них никуда не деться, их не избежать. Все, что вы можете сделать как профессионал, – постараться минимизировать их последствия. Именно поэтому найти и покарать виновного – самое глупое, что можно сделать, когда ошибка уже произошла.

Первым делом вы и ваши коллеги должны понять, что нужно сделать, чтобы сейчас, в данный момент, когда все уже случилось, свести к минимуму урон, нанесенный компании и клиентам. Необходимо выяснить, работает ли на данный момент хоть что-нибудь (нет ли еще дна, которое можно пробить); оценить размер ущерба и приступить к анализу того, что нужно сделать, чтобы исправить ошибку; восстановить потерянное (данные, доверие клиентов, нервы).

Кризисы раскрывают все самое неприятное в людях, поэтому постарайтесь даже в очень сложных ситуациях оставаться профессионалом: не поддавайтесь эмоциям и панике. Полагайтесь на факты и свое понимание ситуации. Что вы реально способны предложить или сделать, чтобы минимизировать урон от сбоя вашего продукта? Как вы можете помочь своим пользователям и клиентам?

Когда вы локализуете проблему и приступите к ее решению, наступит следующая фаза – нет, не поиск виноватого (эта фаза вообще должна быть исключена). Следующая цель – определить, как избежать подобных ситуаций в будущем. Возможно, это будут дополнительные тесты, изменения в коде или пересмотр архитектуры. В любом случае это уже техническая сторона вопроса.

Кризис – это проблема, и необходимо найти ее решение. Если вы подскочите и начнете, сшибая коллег на своем пути, носиться по офису с воплями «Все пропало!», это не поможет. Не поддавайтесь общей панике, работайте в обычном режиме – просто ищите решение.

И, разумеется, необходимо упомянуть случай, когда именно ваш код стал причиной такого крупного и громкого сбоя (ха-ха, смотрите, я нашел виноватого!). Если серьезно, не берите в голову. Делайте то, что вы бы сделали, найдя у себя ошибку при более спокойных обстоятельствах. Сделайте выводы, разберитесь, почему все пошло не так, извлеките пользу из этого происшествия. Черт, вы можете даже немного гордиться им. Если ваша ошибка отключила и сожгла сервер, можете называть себя Повелитель Огня.

Не позволяйте даже самым катастрофическим ошибкам ставить крест на всей вашей работе. Нельзя достичь профессиональных высот, ни разу не сорвавшись вниз. Опыт крупных ошибок и падений будет для вас одним из самых ценных.

Тезисы

- Поиск виноватых – пустое и глупое занятие.
- Если ошибка уже случилась, займитесь ее исправлением.
- Не упрекайте себя за ошибки, набирайтесь опыта, делайте выводы.
- Чем крупнее ошибка, тем ценнее вынесенный урок.

Задание

Попробуйте оценить, как ваша компания переживает кризисные моменты, крупные ошибки или проблемы с продуктом. Можно ли сказать, что ваши коллеги и начальство относятся к возникающим проблемам прагматически? Не случается ли так, что разработчиков, допустивших ошибку, осуждают публично? Если компания предпочитает искать и наказывать виновных, а не решать проблемы, подумайте: а так ли ценна работа там, где вы нужны ровно до того момента, когда допустите ошибку?

История из жизни

Одна история оказалась настолько неприятной и неожиданной, что особенно остро врезалась мне в память. Я был в составе команды, которая занималась очень большим и амбициозным продуктом для российского медиарынка. В конце обычной рабочей недели выяснилось, что небольшая часть данных клиентов (история просмотров) была утеряна при переносе с одного сервера на другой. Ситуация неприятная, но в меру обыденная. Однако один из представителей клиента, человек, и ранее проявлявший себя довольно гадким образом, созвал общее совещание с привлечением всех руководителей, требуя провести внутреннее расследование и предоставить ему имена разработчиков, допустивших ошибку, после чего предложил пересмотреть договор сотрудничества и ввести штрафы за допускаемые в коде ошибки. Сказать, что мы были шокированы таким отношением и подходом, значит не сказать ничего.

Холивары

Ничто так не занимает некоторых разработчиков, как возможность зацепиться с кем-нибудь языком на тему, почему технология A лучше технологии B и почему язык программирования X подходит для решения любых задач, а язык Y – полное дерьмо. Споры между людьми, существующие столько же, сколько существует человечество, с появлением IT-индустрии получили новую жизнь и новых последователей.

Я не буду ходить вокруг да около и скажу сразу: холивары – для людей, у которых слишком много свободного времени. Либо для тех, кто выбрал себе не ту профессию. Возможно, этим людям стоило бы попробовать себя в политике или телемаркетинге, кто знает. Я искренне советую вам не начинать холиваров и не участвовать в них.

Каждый опытный разработчик знает, что нет плохих или хороших технологий. Они бывают только живыми или мертвыми. Либо технология используется – и, значит, она жива, либо не используется – и уже мертва.

Люди эволюционно запрограммированы собираться в стаи. Они чувствуют себя спокойнее и увереннее, если разделяют принципы какой-либо группы и придерживаются их (да, огромное количество преступлений против человечества было совершено по этой причине). Старайтесь руководствоваться здравой логикой. Пусть вы терпеть не можете технологию X, но скажите, по какой причине вы бы стали отговаривать кого-то от ее использования? Вам доплачивают за это? Кто-то угрожает вашей семье? Прелесть несостоятельных технологий в том, что время решит все за них. Равно как и в эволюции: характеристики, которые не сработали, перестанут воспроизводиться, а те, что оказались полезными, дадут начало новым.

Если у вас уже есть опыт работы с несколькими языками программирования (и вы использовали их не только в своих pet projects), то вы представляете, о чем я говорю. Такое многообразие языков существует не просто так. Каждый из них занимал какую-то нишу, каждый пытался наследовать лучшее от предшественников и избавиться от их ошибок. Старался следовать за новыми технологиями, упрощал жизнь разработчиков и пользователей. И нет, это не делает старые технологии плохими или непригодными (давайте, расскажите C, что он мертв).

Не вовлекайтесь в бесконечные споры о том, почему один язык лучше/быстрее/функциональнее другого, не участвуйте в досужих рассуждениях о том, почему какая-либо новая технология или методология разработки обречена на провал. Лучше уделяйте время тому, что действительно важно, – повышайте свои навыки, увеличивайте опыт.

Да, вам будет полезно узнать о двух языках, о которых спорят коллеги, но совсем не для того, чтобы примкнуть к какой-либо из сторон и с пеной у рта доказывать ее правоту. Ознакомившись с обеими технологиями, вы получите двойной опыт – двух разных подходов, двух парадигм, двух видений, которым следовали авторы этих технологий. Это во много раз ценнее словесного пинг-понга, из которого никто и никогда не выходит победителем.

Любая технология существует не просто так. Любая технология пытается решить некие проблемы, которые уже назрели, но, возможно, не так очевидны со стороны. Относитесь к ним с уважением, старайтесь понять, какую именно проблему пытались решить авторы, какие новые подходы для этого использовали.

Тезисы

- Холивары – для людей, которым нечем заняться.
- Все технологии нужны.
- Придерживайтесь своего мнения, но не пытайтесь никого переубедить.
- Собирайте знания, не воюйте с технологиями, абсорбируйте их.

Задание

Дождитесь очередного холивара (а если вы работаете в коллективе, он будет, уверяю вас) и попробуйте узнать о двух технологиях, о которых ведется спор. Ознакомьтесь с каждой из них, не пытаясь оценить их полезность или функциональность. Попробуйте разобраться, почему эти две технологии считаются конкурентами, а затем определите, в чем они таковыми не являются. Попробуйте представить, как данные технологии можно совместить в работе.

История из жизни

Помню, сразу после института я был совершенно уверен, что Java – медленный язык программирования, и даже старался кому-то это доказать – только для того, чтобы несколько лет спустя в поте лица писать на Java систему, работающую в реальном времени. Рад я только одному: что достаточно быстро усвоил урок о ненужности бесполезных споров.

Оценка задач

Приступая к работе на любом проекте, вы должны уметь оценивать задачи по времени их выполнения. Любая компания заинтересована в том, чтобы получить продукт как можно быстрее. Упущенное время всегда станет преимуществом для конкурентов.

Существует множество методологий оценки времени задач; возможно, одна из них используется и в вашей компании. В этом случае у вас уже есть набор правил, по которым вы будете действовать, однако перечисленные ниже пункты обязательны при любой оценке задачи, постарайтесь их запомнить. Со временем это войдет в привычку, и вы сможете давать более точные оценки, учитывая скрытые риски, незаметные на первый взгляд.

Требования. Первое, что вам нужно сделать при оценке любой задачи, – досконально разобраться в требованиях к ней. Любая мелкая деталь может кардинально изменить сроки выполнения. Задайте максимум вопросов по задаче, проведите анализ кода, который будет изменен, поговорите со старшими разработчиками.

Декомпозиция. Если перед вами объемная, сложная задача, разбейте ее на несколько небольших, проанализируйте каждую из них и решите, стоит ли разделить и их на более мелкие. Однако не увлекайтесь. Процесс декомпозиции затягивает, как и любой аналитический вопрос, и вы можете разложить сложную задачу на такие маленькие подзадачи, что потратите больше времени, обновляя их статусы, чем работая непосредственно над основной проблемой. Определите для себя максимальное время работы над задачей: оно и будет для вас индикатором того, что она требует декомпозиции. Чаще всего в крупных компаниях есть некоторое соглашение: к примеру, если предполагается, что задача займет больше 16 часов работы, то ее стоит разбить на подзадачи.

Риски. Определение рисков при оценке задач – довольно сложный этап, особенно если у вас мало опыта или вы еще не очень хорошо знаете проект. Понимание рисков будет складываться из двух факторов: опыта решения таких задач в прошлом (некоторые задачи буквально кричат: «нас можно выполнить за час работы», но на самом деле потребуют нескольких суток) и особенностей проекта, над которым вы работаете (то, что поначалу выглядит как замена надписи на кнопке, вполне может вылиться в исправления в ядре приложения). Отчасти помочь с рисками, которых вы не видите, могут коэффициенты, речь о которых пойдет ниже, но все же учитесь предсказывать (нет, вам не понадобится хрустальный шар, но, возможно, таинственный взгляд будет нелишним) требования к задаче, которые только выглядят простыми, но скрывают под собой целую гору дерьма.

Тестирование. Всегда закладывайте дополнительное время на написание тестов для реализуемой задачи и ее проверки. Компании часто не считают тестирование важным пунктом задачи (ровно до момента, пока все не упадет). Я не могу изменить политику вашей компании, но могу сказать вам: отстаивайте свое право проверять код, который написали. Время, выделенное на тесты для вашей задачи, – это огромный плюс для вас, продукта и компании.

Взаимодействие. Не самый очевидный, но от этого не менее важный пункт. Любая компания и любой проект обладают внутренним распорядком, будь то совместные обеды, ежедневные митинги или трехчасовые лекции директора о том, почему ваша компания лучшая на рынке. Проще говоря, любой коллектив требует, чтобы часть вашего рабочего времени уходила на действия, никак не связанные с написанием кода. Если при анализе задачи вы понимаете, что как минимум два часа будет «съедено» митингом, час уйдет на шуточки продакт-менеджера, которые вам придется слушать, потому что уйти будет неловко, и еще час – на растаскивание в разные стороны коллег, сцепившихся, чтобы выяснить раз и навсегда, можно ли писать на Lisp в продакшн, то учитывайте и это время. Разумеется, вы не сможете указать его как официальное, так как никакой менеджер не признается в том, что потратил

час на шутки, отвлекая коллег, а разработчики сделают вид, что холивары о Lisp – это ваши фантазии.

Коэффициенты. Финальное действие – это применение коэффициентов. Строго говоря, коэффициенты могут понадобиться вам лишь в ряде случаев, но для начинающего разработчика они весьма желательны, так как дают запас времени, даже если изначальная оценка была проведена некорректно (пока вы только набираетесь опыта, так и будет, можете мне поверить). После того как вы оценили задачу, руководствуясь предыдущими пунктами, необходимо выбрать коэффициент и умножить на него полученную оценку. Это дополнительное время – ваша страховка на случай непредвиденных обстоятельств. Выбор коэффициента будет зависеть от того, насколько такая практика принята в вашей компании, готов ли заказчик платить за скрытые риски, а также от того, как вы оцениваете свой опыт в задаче. Если попытаться свести величины к числам, коэффициент будет варьироваться от 1 (вы считаете оценку точной и не предвидите никаких рисков, начальное время оценки после умножения на 1 остается тем же) до 1,5 (вы предполагаете, что задача таит в себе большое количество скрытых рисков и дополнительной работы). Допустим, вы оценили задачу на 10 часов, но догадываетесь, что даже с тестированием и заложенными рисками можете потратить дополнительное время на обсуждение и доработку от дизайнеров, и выбираете коэффициент 1,2. Таким образом, ваша конечная оценка будет 12 часов (10 × 1,2), где дополнительные 2 часа – ваша страховка.

Не пытайтесь никого впечатлить низкими оценками. Никогда. Вы добьетесь лишь одного: создадите себе кучу проблем и будете ночевать на работе. Я не встречал проджект-менеджеров, которые предложили бы вам повышение за то, что вы ставите задачам низкие оценки, а потом выбиваетесь из сил, чтобы уложиться в сроки. Недооценивая задачи, вы ставите под угрозу и качество кода, который пишете, – вы будете торопиться, ошибаться, упускать из виду риски и требования.

Запомните: переоценить задачу гораздо лучше, чем недооценить. Если вы выполните задачу за меньшее время – прекрасно, руководство будет счастливо. Но стоит вам затянуть с работой над задачей, которую вы недооценили, – упреки от коллег не заставят себя долго ждать.

Тезисы

- Перед оценкой выясните все требования к задаче.
- Дробите большие задачи на более мелкие.
- Учитывайте потенциальные риски; запоминайте, когда вы ошибались в оценке.
- Добавляйте время на тестирование.
- ОБЯЗАТЕЛЬНО добавляйте время на тестирование!
- Учитывайте время на «потрындеть», оно такое же реальное.
- Добавляйте коэффициенты оценки, не пренебрегайте своей безопасностью.
- Низкие оценки никого не впечатлят, а вам придется ночевать на работе.

Задание

Задание к этой теме найдет вас само, можете не сомневаться. Практически каждая задача будет полем для практики, но, если хотите проверить себя, попытайтесь придумать для своего продукта новую подсистему, новый компонент или новую функцию. Задача должна быть достаточно большой, чтобы вы могли попробовать декомпозицию, оценку рисков и остальные пункты данной темы. Придумайте требования

к этой задаче, постарайтесь сделать их достаточно размытыми, чтобы потренироваться в оценке рисков и скрытых сложностей.

История из жизни

Всех историй о том, как я недооценил задачу, не перечислить и в трех книгах, но одна запомнилась особенно хорошо – правда, по иным причинам. Я уже не вспомню, что конкретно надо было сделать, но задача выглядела солидно, предполагала время на анализ, на дополнительные тесты. Суммарно выходило порядка 20 часов. Задача была оценена, я отложил ее на поздний срок и вернулся к ней через несколько недель. Работа моя продлилась 2 часа, после чего еще полчаса ушло на то, чтобы понять, как я ухитрился нафантазировать себе оценку в 20 часов.

Общий код

При совместной работе над проектом код почти никогда не бывает написан кем-то одним. Чаще всего код, который вы видите, – результат работы множества людей. Над одной небольшой функцией могли работать десятки человек, и каждый из них имел свое видение кода, свой опыт, свои предпочтения в организации логики.

Общий код проекта – это вопрос договоренностей и соглашений. Код ваших коллег не всегда будет вас радовать. Вы разные люди с разным опытом и подходом к разработке. Возможно, вы в восторге от «условий Йоды», но это совсем не значит, что коллеги разделяют ваши предпочтения. Возможно, вас не восхищает коллега, который предпочитает оформлять в функцию даже небольшие блоки кода, используемые только в одном месте.

Какие-то из этих вопросов вы сможете обсудить на код-ревью, если в вашей компании есть такая практика. Но обычно то, что не относится к корректной работе кода, не служит поводом для обсуждения. Вы, как разработчик, должны уметь читать и воспринимать любой код, даже если он вам не особо нравится.

Проработав в коллективе какое-то время, вы с коллегами так или иначе придете к негласной (или очень даже гласной) договоренности о том, что возможно допускать при работе над кодом, а что – нет. Не расстраивайтесь, если оказались в меньшинстве, и не прыгайте от радости, если ваш стиль приняло большинство. Вы же не станете ловить коллегу в коридоре только для того, чтобы прижать к стене и пытаться убедить, что любить котиков неправильно и он должен срочно завести собаку, а кота выставить за порог.

Старайтесь относиться к чужим предпочтениям с пониманием, этот навык очень пригодится, когда со временем вы станете старшим разработчиком или руководителем отдела. Тогда вам придется постоянно иметь дело с десятками людей, каждый из которых будет писать код так, как нравится ему. И каждый код будет хорош по-своему. Единственным исключением будет такой код, в котором нарушены правила оформления, принципы работы компании или он просто запутан настолько, что понять его может только автор. Да, у всех разработчиков есть собственный стиль, и он не должен становиться препятствием для других.

Тезисы

■ Код проекта – это совместный труд его разработчиков.

■ Вам не обязан нравиться чужой код, равно как и ваш код не обязан нравиться вашим коллегам.

■ Умейте читать и воспринимать любой код.

■ Договаривайтесь с коллегами, как должен выглядеть и работать код вашего проекта.

Задание

Возьмите любой проект и проанализируйте его код. Попробуйте угадать непосредственно по коду, какие части были написаны разными людьми. Используйте git blame, чтобы узнать, правы ли вы. Найдите код, который вам визуально не нравится: он может быть написан с использованием необычных нотаций из прошлого (да, Венгерская нотация, я никогда тебя не забуду), иметь нетипичный подход к размещению блоков и т. д. Попробуйте хотя бы немного привыкнуть к нему.

Представьте, что разработчик, написавший этот код, – ваш подчиненный. Достаточно ли у вас оснований сказать, что код не работает, и отправить его на доработку?

История из жизни

Помню, на одном из проектов мы долго обсуждали с коллегой, как именно должен выглядеть код на языке программирования, на котором мы в тот момент писали. Спустя некоторое время в обсуждение вовлекся весь отдел. До ссоры не доходило, но и общего согласия на горизонте видно не было. В итоге мы решили вопрос анонимным голосованием (и да, для этого быстро написали маленькое приложение, с чего бы нам просто кидать бумажки в шляпу!); принятое решение пошло на пользу всем.

Одно кольцо, чтоб править всеми

В любом проекте наступает момент, когда необходимо сделать важный выбор. Это может быть техническое решение, которое отразится на архитектуре продукта, или решение стратегическое, определяющее судьбу продукта на рынке. И нередко это решение будет приниматься либо конкретным человеком, либо очень ограниченным кругом лиц.

Для разработчика, может быть, не так важно, как продукт позиционируется на рынке и какие преимущества имеет перед конкурентами. Однако вы должны ясно понимать, как дорого может стоить неверно принятое архитектурное решение. В зависимости от величины компании технические решения могут приниматься главным разработчиком, архитектором, техническим директором или уборщиком – уж как повезет.

И все было бы здорово, если бы не одно но. Иногда человек, который должен принять ключевое решение, либо отсутствует на проекте совсем, либо он есть, но не обладает для этого достаточными знаниями и опытом.

В случае если в вашей команде нет человека, отвечающего за решения по архитектуре и развитию проекта, вам будет очень-очень непросто. Любой проект проходит стадии развития. Он растет, эволюционирует, крепнет. И он будет изменяться, подстраиваясь под задачи, которые должен выполнять, или под требования клиентов. Иногда он будет меняться кардинально, а это требует смелости и умения смотреть в будущее (у вашего архитектора есть хрустальный шар, точно говорю).

Разработчики достаточно амбициозны, и если у вас нет сотрудника, принимающего важные решения по развитию проекта, кто-нибудь из ваших коллег попытается занять это место. Это может быть большой удачей, в случае если человек обладает достаточным опытом и изрядным запасом интуиции. И это же может стать провалом, если сотрудник просто хочет признания, не обладая при этом знаниями и способностью выбрать правильный курс.

Если в какой-то момент карьеры именно вы окажетесь тем, кто принимает ключевые решения, я заранее поздравляю вас. Поздравляю с тем, что отныне каждый ваш выбор будет делить людей на две или больше групп. Часть из них будет согласна с вашим решением, а часть – сопротивляться ему так, будто вы пытаетесь поджечь их дома. Это нормально, опирайтесь только на объективные критерии выбора. Да, когда вы объясняете, что необходимо сделать, старайтесь находить правильную аргументацию и слова. И не нужно угождать всем, ваша задача – сделать проект сильнее и лучше.

Тезисы

- Каждый проект – это совокупность больших и маленьких решений.
- Человек, не обладающий достаточным опытом принятия решений, так же плох для проекта, как и отсутствие человека, принимающего решения.
- Если решения принимаете вы, будьте готовы столкнуться с сопротивлением.

Задание

Если в вашей команде есть человек, отвечающий за принятие ключевых решений по развитию проекта, – прекрасно. Попробуйте анализировать его подходы, его рассуждения при принятии решений. Находясь на позиции наблюдателя, вы сможете оценить, были ли принятые решения верны в долгосрочной перспективе. Поставьте себя на его место: будет ли вам интересно принимать такие весомые решения? Готовы ли вы нести за них ответственность?

История из жизни

Моя история, связанная с этой темой, довольно печальна. Я был middle-разработчиком и работал с техническим руководителем, который имел очень приблизительное представление о разработке и технических решениях. Собственно, он был из тех специалистов, которые окончили платные курсы и по случайности получили должность, позволявшую им принимать ключевые решения по развитию продукта. Я не обладал достаточной уверенностью или опытом, чтобы оспаривать его решения, но видел, что на дальней дистанции с ними возникнут большие сложности. В итоге я просто отмалчивался и продолжал им следовать, наблюдая, как проекту становится все хуже. Молча смотреть, как продукт, в который ты вкладываешь так много себя, страдает и теряет аудиторию – болезненный опыт, повторения которого я бы не хотел.

Обсуждения

Если бы мы стали делить разработчиков на типы (а глупее занятия не придумать), то можно было бы сказать, что они делятся на тех, кому нравится обсуждать свою работу, и тех, кто предпочитает работать молча. Принадлежность к одному из этих типов не делает разработчика хуже или лучше, это всего лишь стороны характера, особенности личности. Кому-то по душе обсуждать все нюансы своей задачи с коллегами, кому-то удобнее разобраться со всеми вопросами один на один.

Обсуждение проекта в коллективе – очень важная часть рабочего процесса. Но ровно до тех пор, пока оно не начинает мешать работе над проектом. К сожалению, соблюдение баланса в обсуждении задач и рабочих процессов – дело очень-очень непростое. И чем больше людей вовлечено в обсуждение, тем сложнее оно становится. А если в команде нет человека, который может принять окончательное решение, процесс становится бесконечным.

Поверхностное или слишком сжатое обсуждение в рабочем процессе грозит тем, что разработчики не смогут увидеть общую картину. Задачи будут выполняться не полностью, с ошибками, без учета всех требований, а иногда абсолютно некорректно, просто потому что у разработчика нет полного понимания того, что он делает. Замечательно, если разработчики в вашей команде способны на самостоятельный анализ, оценку рисков и четкое понимание требований без долгих разговоров о них.

Другая сторона медали – чрезмерное обсуждение, разговоры по делу и без, многочасовые митинги и совещания с короткими перерывами на разработку. Такая проблема свойственна большим компаниям, где, согласно корпоративной этике, хороший разработчик – тот, кто присутствует на ежедневных митингах и с восторгом рассказывает о том, что он уже сделал и чем планирует заняться дальше. Поймите меня правильно: я не считаю, что совещания с коллегами – это плохо. Обсуждение работы над проектом – прекрасная вещь, но только если оно не ворует ваше рабочее время.

Иногда способы, которыми руководство и менеджеры решают проблему контроля за разработкой, буквально парализуют компанию. В попытке получить больше информации о том, чем занимаются сотрудники, вас будут раз за разом собирать в залах и комнатках, где нужно будет рассказывать, что и как вы делаете. Если вы работаете над большим продуктом, то, разумеется, такие встречи необходимы. Отделы должны понимать свои задачи, контактировать и узнавать друг друга в лицо: это упрощает коммуникацию внутри компании. Но если вы понимаете, что из восьми часов рабочего времени потратили пять на разнообразных обсуждениях, значит, в вашей компании что-то идет не так.

К сожалению, рецепта идеального баланса обсуждений нет, и каждая компания спустя некоторое время приходит (или не приходит) к тому количеству разговоров, которое устраивает и руководство, и сотрудников. Для вас, как для профессионала, важно понимать, что ваша задача и приоритет на проекте – разработка качественного кода. И только вы можете знать, делают ли многочасовые разговоры ваш код лучше.

Тезисы

- Мало обсуждений – плохо.
- Много обсуждений – тоже плохо.

Задание

Если в компании проводятся регулярные митинги, попробуйте ответить честно: насколько полезны они для вас как для разработчика? Если вы чувствуете, что они полезны и вы получаете больше опыта, – прекрасно, я могу за вас только порадоваться. Если же вам кажется, что на этих обсуждениях

вы лишь теряете время, поинтересуйтесь у руководства, нельзя ли реже принимать в них участие.

История из жизни

Я настолько не любил пустые ежедневные обсуждения, что это стало для меня одной из мотиваций, чтобы развиваться как разработчик. Я хотел получить возможность сказать «нет, мне некогда», однако это сыграло со мной злую шутку. Поднимаясь вверх по карьерной лестнице, ты становишься тем самым человеком, который и инициирует подобные обсуждения, и обязан на них присутствовать. Любви к пустым обсуждениям у меня, разумеется, не появилось, поэтому я стараюсь не отвлекать разработчиков, если только для этого нет по-настоящему веских причин.

Бюрократия

Как бы печально это ни звучало, но бюрократия на сегодняшний день – неотъемлемая часть мира IT. Она проникает в нашу индустрию потому, что руководству компаний нужна четкая структура и понимание продукта (с какой скоростью и с какими перспективами идет его разработка). Это благое желание, которое часто выливается в замедление или усложнение разработки.

Возможно, я начал на печальной ноте и конкретно в вашей компании бюрократия не имеет такого размаха. Однако велика вероятность, что вы попадете в компанию, где любое взаимодействие между отделами или новая идея проходят слишком много согласований, так что под конец вы уже толком не помните, с чего, собственно, начинали.

Нельзя сказать, что бюрократия совсем бесполезна. Нередко случается, что спонтанная идея после долгого обсуждения и хождения между отделами действительно оказывается никчемной. Но бывает и так, что хорошая идея для продукта получила одобрение слишком поздно и компания уже не успевает за конкурентами, упустив время.

Вы, как разработчик, вряд ли будете напрямую влиять на рабочие процессы в компании (и скажите за это спасибо, вы НЕ хотите лезть в эту банку со змеями).

В первую очередь не следует принимать происходящее близко к сердцу. Да, возможно, именно вашу идею задвинули в угол или сочли неудачной. Да, случается, что из-за медленного взаимодействия с дизайнерами вы уже две недели не можете получить макет главной страницы приложения. Воспринимайте такие ситуации нейтрально.

Бюрократия – это неповоротливая машина, которая часто дает сбои, но, кроме этого, она может уберечь вас от потенциально провальных решений, упростить взаимодействие с коллегами или обеспечить качественный отпуск (ощутите на языке вкус бананового коктейля).

Тезисы

- Бюрократия – неотъемлемая часть больших компаний и продуктов.
- Редкие плюсы бюрократии теряются среди ее многочисленных минусов.
- Не позволяйте сухим бюрократическим формальностям расстраивать вас.

Задание

Если вы работаете в компании, известной своими бюрократическими проволочками, и вас это БЕСИТ, начните тренироваться. Воспринимайте каждое вовлечение в череду ненужных (на ваш взгляд) обсуждений как прогон тестов. Да, это медленно, да, вы в это время не можете толком заняться ничем другим, но, возможно, именно на этом прогоне вы заметите какую-нибудь очень неприятную, очень опасную ошибку. Используйте время, которое теряете в бюрократической бездне, с пользой для себя: подтяните свои знания по языкам программирования, займитесь рефакторингом вашего приложения (да-да, если вам одобрили эту задачу, подписав 12 бумажек на 6 разных этажах и прислав почтового голубя с письмом-подтверждением).

История из жизни

Одна из компаний, в которых я работал, требовала тотальной отчетности по рабочему времени, разбитому на десятиминутные интервалы, так что если в течение часа вам приходилось заниматься разными делами, вы должны были составить примерно такой список:

10 минут – обсуждение с коллегой ошибки #123
20 минут – анализ требований к задаче #125
10 минут – поиск библиотек, подходящих под требования задачи #125
10 минут – общение с тестировщиком по ошибке #123
10 минут – работа по внедрению найденной библиотеки в задачу #125

Все это отнимало уйму времени, и на отслеживание и запись своих занятий нужно было тратить куда больше усилий, чем непосредственно на сами занятия.

Идеальный продукт

Людям свойственна тяга к прекрасному, а перфекционисты – люди, ставшие рабами этой тяги. Желание сделать работу как можно качественнее – важная и замечательная черта любого специалиста. Когда мы начинаем новый проект или новую работу, мы горим (мама, неси огнетушитель!). Мы мотивированны, возбуждены, ушки у нас на макушке. Мы хотим сделать все правильно. Мы хотим сделать все качественно.

Извините, но сейчас я спущу вас обратно на землю. Мир – это хаос. Мир программного обеспечения, при всей кажущейся структурированности и логичности, – еще больший хаос. И мы становимся заложниками желания привнести в этот хаос смысл и логику. Мы пытаемся написать идеальный код. Дизайнеры пытаются создать идеальный, понятный и практичный интерфейс. Тестировщики пытаются написать идеальные системы проверки продукта.

И все это разбивается о пользователя, который решил отправить форму, оставив вместо имени цифру 0. Да, я утрирую, но лишь слегка. Мы так отчаянно стараемся создать что-то незыблемое и прекрасное, что отрываемся от реальности, от хаоса, в который погрузится наш продукт, как только попадет на рынок, к пользователям. И я не хочу сказать, что в этом есть чья-то вина: наша как создателей или пользователей как потребителей. Просто так устроен мир, и мир программного обеспечения не исключение.

Для примера понаблюдайте за опытными разработчиками. Со стороны их работа может показаться достаточно небрежной. И дело не в том, что они нарушают стиль кода или вместо того, чтобы писать код, играют в пинг-понг. Нет, «небрежность» заключается в самом их отношении к своему коду. Вы редко увидите, что профессиональный разработчик корпит сутки напролет, выстраивая монструозную систему отлова всех ошибок и исключений. Вы вряд ли увидите разработчика, который тщательно напишет все возможные тесты для только что созданного им кода.

Эта кажущаяся небрежность – не более чем показатель большого опыта, полученного на множестве проектов. Каждый разработчик, приобретя достаточный опыт в IT, начинает понимать, что идеального кода не существует. Есть код, который работает и делает то, что должен делать. С одной стороны, профессиональный разработчик должен писать качественный код, с другой – может быть достаточно небрежен в работе. Но как?! Опыт и интуиция. Если за свою карьеру вы написали 10 калькуляторов, то вряд ли станете писать сотню юнит-тестов по проверке сложения (assert(2 + 2, 4), какая красота, вы только посмотрите). Вероятнее всего, вы напишете десяток тестов, но таких, которые будут включать самые нелепые, самые глупые входные данные, какие только могут быть: сложение со строкой, деление на ноль, возведение в объект. Опыт поможет вам почувствовать, в каком месте код встретится с хаосом реальности и когда он падет смертью храбрых. Но даже тогда вы не предугадаете всех вариантов, поэтому позвольте себе наслаждаться процессом. Не делайте из кода или продукта культ. Пусть работа приводит вас в восторг, а мотивация зашкаливает так, что не дает уснуть по ночам. Однако всегда помните о том, что хрустальные замки идеалов – в облаках, а внизу – хаос реальности, от которого нельзя избавиться, просто отказавшись о нем думать.

Тезисы

- Мир программного обеспечения – хаос.
- Старайтесь выполнять свою работу качественно, но не кладите ее на алтарь перфекционизма.
- Идеальных решений, как и идеального кода, не существует.

Задание

Вспомните самое начало своей карьеры. (Если вы начинающий разработчик, вернитесь к этому заданию чуть позже – уверен, оно вас впечатлит.) Вспомните, с какой дотошностью вы относились к новому коду, который впоследствии приходилось писать не один десяток раз. С точки зрения вашего опыта на сегодняшний день можете ли вы сказать, что по-прежнему так же педантичны? Что пытаетесь учесть все-все варианты и исходы кода? Что предпочитаете ставить обработчики ошибок и логирование через каждые три строчки, чтобы уж точно не упустить ни одной опасной ситуации?

История из жизни

Ощущение, будто я и правда могу создать что-то идеальное, никуда полностью не уходит, но мои ожидания становятся реалистичнее. Само словосочетание «идеальный код» теряет свою магическую недостижимость и означает вполне конкретные критерии, которыми я буду доволен, когда достигну их выполнения. Ощущаю ли я, начиная новый проект, что хочу создать идеальный продукт? Разумеется, да. Просто теперь «идеальный» для меня звучит как «надежный», «удобный для пользователей», «расширяемый» и «продуманный». На такие идеальные качества я вполне могу ориентироваться без того, чтобы усложнять бесконечным перфекционизмом и без того сложную работу.

Код-ревью

Качественный способ улучшить код, найти ошибки на ранней стадии, ознакомить членов команды с компонентами системы, с которыми они раньше не работали, – прекрасно. Возможность разобрать по косточкам код коллеги, которого вы недолюбливаете, чтобы найти ошибку, – бесценно. Шучу. Не обижайте коллег, вам с ними делать релизы.

Код-ревью – это прекрасная практика, которая помогает держать код проекта в чистоте и порядке, придерживаться изначально задуманной архитектуры, находить ошибки, давать подсказки новым разработчикам и т. д. Описывать прелести код-ревью можно долго, а еще проще найти об этом сотню статей.

В этой теме я не буду подчеркивать достоинства ревью кода. Просмотр кода полезен, и если в вашей компании его практикуют – прекрасно.

Код-ревью вовлекает разработчиков в диалог, в ходе которого один из них должен занимать оборонительную позицию. Может быть, и существуют идеальные команды, где каждый разработчик уважает своих коллег и то, как они пишут код, но давайте будем реалистами.

Формальные правила код-ревью предполагают, что написанный для проверки код должен соответствовать правилам проекта, которые нельзя проверить автоматизированно. Иными словами, код-ревью подразумевает проверку логики работы написанного кода, целесообразность использования механизмов, алгоритмов и компонентов.

Код-ревью поможет понять, соответствует ли код архитектуре и смыслу проекта, не используются ли средства, которые не принято использовать на проекте, и многое-многое другое. Отчасти с такой проверкой могут справиться тесты, но чаще всего тесты пишутся тем же разработчиком, что и код, попавший на ревью, а следовательно, они неспособны выявить все потенциальные проблемы.

Код-ревью требует времени. Нередко это раздражает, потому что не во всякой компании на ревью выделяется отдельное время и часто код-ревью требуется совмещать с работой над задачами. Просмотр кода не просто предполагает беглое ознакомление с написанным, но требует понимания поставленных задач и логики работы кода. В случае если мы говорим о ревью добавленной кнопки с одним действием, все кажется не таким сложным. Но если вы смотрите на код нового компонента аудита действия пользователей системы – да поможет вам Вселенная.

Код-ревью требует опыта и отстраненности. Умение донести свою позицию, не перегибая палку и не пытаясь переделать чужой код в свой, требует большого опыта. С годами у разработчиков вырабатывается свой стиль, свое видение кода. Да, мы используем один и тот же язык с одинаковыми правилами синтаксиса и выполнения, и тем не менее у каждого разработчика есть личный подход. Это может быть свой метод именования переменных, своя логика при работе с массивами, свой способ контролировать ошибки и исключения – все что угодно.

А теперь сосредоточьтесь. Не позволяйте своим привычкам проникать в код-ревью. Вы НЕ делаете код лучше, когда просите другого разработчика переименовать переменную. Вы НЕ делаете код лучше, когда просите заменить for на while просто потому, что привыкли обходить циклы так. Более того, даже если ваш подход обеспечивает мизерную оптимизацию на конкретной версии вашего компилятора, придержите эту ценную информацию при себе.

Ревью кода – это не схватка идеальных машин, это способ держать проект в балансе между разными людьми, у каждого из которых есть на него уникальный взгляд.

Код-ревью потребует от вас крепких нервов. Да, возможно, вы ветеран многих холиваров и закалены в интернет-дуэлях. Но скорее всего (особенно если вы начинающий), вы нервничаете каждый раз, когда ваш код отправляется на ревью старшим разработчикам. Это нормально. Хорошо сомневаться в своем коде, отлично, если вы хотите, чтобы он был лучше.

Проходя код-ревью, постарайтесь забыть, что этот код ваш, и объективно воспринять комментарии, которые вам напишут. Представьте, что сами присутствуете на этом код-ревью, а код принадлежит другому человеку. Хорошим советам и замечаниям стоит только радоваться: код станет только лучше.

Относитесь к код-ревью как к полезному инструменту, как к плагину для вашей среды разработки. Смогли бы вы прожить без него? Да. Но если что-то может сделать ваш вклад в проект более качественным, почему бы не воспользоваться такой возможностью?

Тезисы

- В ходе код-ревью не обороняйтесь и не нападайте, это не битва.
- Уделите время код-ревью.
- Абстрагируйтесь от кода и стиля, сосредоточившись на логике написанного.
- Воспринимайте рекомендации как добрый совет, не ищите в них упрека.

Задание

Попробуйте найти в системе контроля версий вашего проекта задачу, которую вы делали какое-то время назад и уже успели забыть. Возьмите ее на ревью. Оцените свежим взглядом собственные решения, выбор механизмов и подходов. Проверьте, соответствует ли ваш код архитектуре проекта. Достаточно ли много общего он имеет с похожими компонентами в системе? Какие рекомендации по его улучшению вы бы дали, руководствуясь своими знаниями на сегодняшний день?

История из жизни

Как-то раз я проводил код-ревью начинающего разработчика и делал это по большей части автоматически – комментировал, писал, почему так не стоит делать и как сделать лучше; словом, это было самое обычное ревью, по крайней мере для меня. Через некоторое время мне передали, что разработчик, чей код я проверял, работает через силу, подавлен, сомневается даже в простых решениях, которые пишет. Я решил поговорить с ним, узнать, что случилось, нет ли каких-то личных причин, которые он хотел бы обсудить. Выяснилось, что мое код-ревью было обычным только для меня, а разработчик воспринял его как отповедь, как прямую критику в свой адрес. Я извинился и попытался объяснить, что ничего из того, что я говорил, не может и не должно восприниматься как упрек или сомнения в его профессионализме: это всего лишь комментарии, которые должны подготовить код к тому, чтобы он стал частью проекта. Мы хорошо поговорили в тот день и достаточно долго поддерживали дружеские отношения, иногда вспоминая эту историю.

Методологии разработки

Тема, занимающая умы тысяч продакт-менеджеров. Тема, заставляющая директоров компаний потирать руки в ожидании невероятной продуктивности сотрудников. Тема, обязывающая разработчиков готовить доклады о том, как прошла их рабочая неделя.

Когда-то разработка программного обеспечения была узкоспециализированной необходимостью. Научной, математической, интеллектуальной. Развиваясь, разработка стала требовать большего, чем просто умение создавать программы и вычислять результаты. Итеративность процесса разработки, необходимость предсказуемости результата, формирование четкой структуры и фаз разработки требовали создания чего-то нового. Алгоритма по созданию программного обеспечения. Методологии.

Мы не будем погружаться в пересказ всех доступных (и уже покойных) методологий разработки. Если вам действительно этого захочется, можете найти их самостоятельно. Методологии позволяют структурировать процесс разработки программного обеспечения, составлять формальные правила по оценке задач, регламентировать обсуждения и планирование, оценивать эффективность людей и команд внутри компании.

Насколько методология разработки, предпочитаемая в компании, повлияет на вас как на разработчика? Это зависит от вас.

Для кого-то сам набор правил и регламентов позволяет чувствовать себя безопаснее и предсказуемее (многие разработчики любят структурированность не только в своем коде, но и в жизни).

Кому-то выбранная методология разработки будет отравлять жизнь необходимостью совершения рутинных действий, присутствия на пустых и обременительных обсуждениях и т. д.

На ком-то наличие методологии не скажется никак. Совсем.

Говоря о методологиях разработки, с большой долей уверенности можно сказать, что:

- наличие выбранной методологии разработки лучше ее отсутствия (плюс для компании);
- выбранная методология не должна мешать вам работать (плюс для вас).

Отчасти методологии защищают и самих разработчиков, помогая предупредить неверные решения, спрогнозировать лучший подход или верно оценить время, которое потребуется. Однако в первую очередь методологии нужны компаниям, и на то есть причины. Да, каждый разработчик способен на многое, но, как и любая совокупность людей, участвующих в одном деле, мы частенько теряем представление об общей картине происходящего. Не потому, что каждый человек в толпе глуп или недальновиден, просто сама толпа становится менее предсказуемой и эффективной.

Методологии разработки появлялись тогда, когда в них возникала необходимость. Часто методологии, описанные максимально подробно и применяемые максимально точно, абсолютно не приживались в одних компаниях и давали невероятный толчок к производительности в других.

Применение методологии разработки – не панацея и не проклятие, это всего лишь инструмент, полезность которого определяется методом проб и ошибок. На ваше счастье, если вы работаете в крупной компании, там уже есть люди, которые ломают над этим голову. Относитесь к методологии разработки как к временам года. Они могут вам нравиться или нет, но они есть, и вам нужно извлечь из этого плюсы.

Тезисы

- Методология разработки – попытка структурировать и повысить производительность разработки программного обеспечения.

■ Чаще всего методологии создавались для конкретных компаний и с определенными целями.

■ Верная методология в любой компании определяется методом проб и ошибок.

■ Извлекайте из методологии разработки плюсы, даже если ее минусы мешают вам работать.

Задание

Определите, какая методология разработки используется в вашей компании (если используется вообще). Ознакомьтесь с историей происхождения этой методологии и оцените, насколько корректно компания следует изначальной идее. Если вас интересует структурирование взаимодействий внутри компании, попробуйте проанализировать слабые места в организации работы, предложите что-то из других методологий, которые, возможно, лучше бы подошли вашей компании.

История из жизни

Из всех моих пересечений с методологиями разработки я вынес для себя только одну, самую полезную и нужную технику. Вопреки тому, что я могу воспользоваться мириадами программ для трекинга задач, событий и важных уведомлений, я все еще предпочитаю клеить САМЫЕ ВАЖНЫЕ напоминания на монитор стикерами с написанным от руки текстом. Обычно мой монитор похож на рождественскую елку, разве что не зеленый и не мигает огнями гирлянды. А еще я веду письменные заметки во время технических обсуждений, чем вызываю немало шуток про бересту и глиняные таблички.

Я

Этот раздел написан про вас. Ну хорошо, хорошо, он написан про меня, потому что я не могу описать здесь ваш опыт, ваши будущие решения и профессиональный путь. Возможно, темы этого раздела покажутся вам полезными, а может быть, вы поймете, что в некоторых ситуациях поступили бы иначе. В любом случае я опишу то, что на протяжении моей карьеры казалось мне самым важным. То, что беспокоило или радовало лично меня. В конечном счете мы все удивительно похожи, и, возможно, вы сумеете найти в моем опыте что-то полезное.

Забота о себе

Внутреннее состояние оказывает огромное влияние на вашу работу.

Пожалуйста, остановитесь и еще раз прочитайте предыдущее предложение. Я серьезно. Вам может казаться, что работа – это исключительно набор навыков, технический склад ума или просто опыт, но, поверьте мне, самочувствие сказывается на ней больше, чем вы можете себе представить.

Отмечайте то, что вас радует. Возможно, это новые проекты, сложные задачи, какие-то конкретные технологии или направления разработки. А может, вы радуетесь, когда гладите страуса или готовите домашнюю пиццу. Какая-нибудь мелочь, которая делает счастливым лично вас, может дать вам мотивацию на неделю работы вперед. Замечайте это, запоминайте то, что дает вам силы.

Разумеется, вы не будете чувствовать себя всегда на все сто, и это нормально. Не корите себя, если замечаете, что недостаточно удовлетворены работой. Не упрекайте себя, если понимаете, что внутреннее состояние не позволяет вам работать так же качественно и быстро, как раньше. Отнеситесь к себе с пониманием, вы не можете обязать себя быть постоянно на подъеме. Дайте себе время, позвольте восстановиться.

Старайтесь радовать себя даже тогда, когда работа занимает все ваше время. Выделите час в процессе работы, найдите время вне работы, чтобы сделать то, что поднимет вам настроение. Это может быть все что угодно: возможно, вам давно хотелось попробовать написать 'Hello world' на Scala; а может, вы мечтали пострелять из арбалета или погладить енота.

Заботьтесь о себе во внерабочее время, это даст вам куда больше энергии для профессионального роста, чем кажется.

Это же касается и физического самочувствия. Я думаю, мне нет смысла объяснять, что писать качественный код, когда раскалывается голова или живот сводит болью, практически невозможно. И не нужно.

Быть разработчиком сложно – это требует очень больших затрат энергии как тела, так и мозга. Не пренебрегайте сигналами от организма, не забывайте нормально питаться и спать; следите за тем, как и сколько вы отдыхаете от новой информации.

Я не буду говорить про йогу, спортзал и абдоминальное дыхание. Это вещи, отношение к которым вы определите сами. Моя задача – донести до вас, что ментальное и физическое состояния неразрывны и напрямую сказываются на вашей работе.

Тезисы

- Ваше внутреннее состояние – залог качественной работы.
- Вы должны заботиться о себе.
- Ваши ментальное и физическое состояния тесно связаны, не надейтесь удерживать на плаву что-то одно.
- Находите и создавайте поводы для радости.

Задание

Составьте список из 10 занятий, которые вас всегда радуют. Любимая еда? Отлично. Тайная страсть к написанию кода на ассемблере? Замечательно. Храните этот список и заглядывайте в него, когда понимаете, что ваша мотивация под угрозой, или чувствуете, что работа перестает вас радовать, как раньше.

История из жизни

Увы, я всегда знал, как надо заботиться о себе, но редко следовал своим же правилам. Переработки? Да. Работа месяцами без выходных? Да. Черт, я как-то посчитал свое рабочее время за один очень насыщенный год, просуммировав общее количество рабочих часов и разделив на количество дней. Вышло, что в том году я работал по 8 часов КАЖДЫЙ день. Не будьте как я, будьте здоровее, чем я, заботьтесь о себе и своем здоровье.

Усталость и выгорание

Хотел бы я сказать, что вы никогда не будете уставать так, что вас будет тошнить от одной мысли о работе, и никогда не столкнетесь с профессиональным выгоранием. Но это было бы ложью.

В какой-то момент способы, описанные в теме «Забота о себе», не сработают так, как раньше. Вы почувствуете, что все, чем вы занимаетесь, – бессмысленно, что ваши решения в коде – некачественные и глупые. Вы начнете допускать грубые, детские ошибки, станете хуже принимать решения, хуже анализировать новые задачи и требования.

Поздравляю, вы стали профессиональным разработчиком. И это только отчасти шутка. Любой опытный разработчик сталкивается с личным кризисом: в силу внешних или внутренних причин ему кажется, что его работа бессмысленна и ничего не стоит.

Я мог бы попробовать описать причины таких кризисов, но в реальности их слишком много, чтобы перечислить. Это могут быть личные причины, не связанные с работой. Причиной может быть ваша компания, продукт, депрессия, начальник-идиот. Чаще всего причин множество, в какой-то момент происходит надлом и работа начинает страдать вместе с вами.

Первый совет, который я хочу вам дать, – притормозите. Возьмите больничный, даже если это будет стоить недовольных лиц руководства. Возьмите отпуск, пусть даже вы не будете присутствовать на релизе своего продукта. Если вы исчерпали весь свой ресурс, то можете сделать лишь одно: остановиться и обдумать ситуацию без нависающих над вами дедлайнов, ошибок и клиентов.

Не обвиняйте себя и не обесценивайте свою работу. Подобные мысли будут посещать вас очень часто. Вам будет казаться, что вы неудачник, ничего не умеете и не можете сделать хорошо. Помашите этим мыслям рукой, они просто обманывают вас и не имеют никакого отношения к реальности. Это всего лишь ваш усталый мозг дает сигналы, что ему пора отдохнуть.

Дайте себе столько времени, чтобы передохнуть, сколько необходимо. Вместе с выгоранием приходит потеря интереса ко всему, что вас радовало, мотивировало, интересовало. Ваша задача – сделать паузу, чтобы эти ощущения вернулись. Если вы снова чувствуете интерес, захотелось попробовать что-то новое в разработке, заинтересовала какая-то новая технология, – поздравляю, вы на правильном пути.

Однако есть и другой сценарий. То, что я сейчас скажу, прозвучит безжалостно, но я должен это сделать: возможно, взяв перерыв, вы поймете, что у вас уже нет желания работать в этой индустрии. И я не имею в виду конкретную компанию или проект. Возможно, вы почувствуете, что не хотите больше заниматься разработкой. Не пугайтесь этой мысли, дайте ей право на существование. Даже самая успешная карьера не стоит того, чтобы быть несчастным.

Выгорание и депрессия часто идут рука об руку, и вам нужно понимать, что не всегда потеря интереса к работе связана только с самой работой. Если вы понимаете, что ваше ментальное состояние ухудшается и потеря интереса к работе лишь следствие того, что вы несчастны как человек, пожалуйста, обратитесь к специалисту, начните психотерапию. Кислородную маску сначала надевают на себя, затем на ребенка. Работа никогда не будет важнее вас как человека, позаботьтесь в первую очередь о себе.

Тезисы

- Выгорание однажды случится и с вами.
- Остановитесь.
- Не смейте себя упрекать и дайте себе столько времени, сколько нужно.
- Помните, что вы не тождественны своей работе: вы значительно важнее, чем она.

■ Выгорание и депрессия – частые спутники; если перестаете справляться сами, обязательно обратитесь за помощью.

Задание

Никогда не упрекайте себя из-за работы и давайте себе столько времени на передышку, сколько вам требуется.

История из жизни

Я выгорал, разумеется. Я впадал в депрессию, я попадал в больницу, все это было. И в то же время я могу назвать себя счастливчиком, потому что ни один из этих спадов не закончился для меня ненавистью к разработке. Да, это было очень тяжело, но я никогда не задумывался о том, чтобы делать что-то другое. Моя работа для меня – спасение и проклятие, так было и есть. С каждым годом я только лучше понимаю, как извлекать плюсы из одной крайности и нивелировать минусы другой.

Винтик в механизме

Долгая работа на одном проекте или в одной компании заставляет наши мозги ржаветь. Мы используем одни и те же технологии, работаем по одним и тем же установленным в компании правилам, применяем одни и те же методологии и подходы. В какой-то момент это может даже начать доставлять удовольствие: все становится достаточно знакомым, чтобы работу можно было выполнять быстро, просто и шаблонно. И ровно в этот момент наш мозг начинает ржаветь.

Труд разработчика многогранен и интересен. Нам приходится решать сложные задачи, находить новые подходы и разбираться в порой совершенно безумных требованиях заказчиков. Однако постоянное повторение одних и тех же решений, работа в одной и той же области требований, с одними и теми же технологиями нередко приводит к тому, что наши навыки становятся узкоспециализированными, заточенными под один набор задач. В некоторых областях разработки программного обеспечения это станет плюсом, но если вы понимаете, что теряете профессиональную хватку, – пришло время помогать себе.

Ощущение того, что вы лишь винтик в механизме, характерно для разработчиков больших компаний, работа каждого из которых сосредоточена на конкретной подсистеме или части функций проекта. Разработчики могут годами видоизменять одни и те же части кода, теряя представление о том, как система функционирует в целом. Кому-то из вас такая работа будет по душе, но кто-то может начать чувствовать постоянное давление и растущее ощущение своей бесполезности.

Давайте оговоримся сразу: любая работа, которую вы делаете, важна и нужна. Даже не пробуйте спорить с этим и предполагать «другие варианты». Примите это как безусловный факт. Однако вы должны помочь себе. Ваши амбиции (я употребляю это слово в самом положительном смысле), ваш опыт требуют нового. Они требуют, чтобы вы развивались, им нужны ваша помощь и участие.

Знания и опыт – это то, что делает вас профессионалом. Помогите себе: начните заново пополнять их. Пробуйте направления, которыми раньше никогда не занимались. Вы занимаетесь веб-разработкой? Научитесь писать на ассемблере. Создаете мобильные приложения? Попробуйте написать свою мини-игру под Linux. Однако замечу: выбирайте то, к чему испытываете интерес. Нет ничего хуже, чем заставлять себя заниматься тем, что не по душе. Знания никогда не бывают бесполезными, особенно в нашей индустрии. Вы думаете, что разобрались в том, как работает сетевой стек, но это вам никогда не пригодится? Просто подождите. Думаете, что вам совершенно ни к чему знать, в чем разница между stack и heap? Года не пройдет, как вы столкнетесь с этим в работе.

Есть вероятность, что, даже получая новые знания, вы все равно будете чувствовать себя как в клетке. Такое ощущение часто возникает, если компания не дает пространства для роста и развития. Возможно, руководство не считает это важной или нужной частью вашей рабочей деятельности; возможно, ему выгоднее задействовать вас прицельно на конкретном проекте или части проекта. В любом случае, если вы постоянно чувствуете себя в западне, подумайте, а так ли полезна для вас работа в компании, которая не хочет видеть профессиональный рост и новый опыт своего сотрудника.

Тезисы

- Не позволяйте мозгам ржаветь.
- Поглощайте новую информацию.
- Следите за тем, чтобы не останавливаться в развитии.

Задание

Попробуйте честно ответить себе на вопрос: вы чувствуете, что на текущем месте работы получаете новые знания и опыт? Не ощущаете ли вы себя в западне рутинной, повторяющейся работы? Если понимаете, что начали буксовать в профессиональном развитии, попробуйте изучить что-то новое: язык программирования, подход к разработке – что угодно, что вызывает у вас интерес. Если же вы пробуете снова и снова, но все равно возвращаетесь к работе, где вас используют как инструмент для выполнения одних и тех же задач, задумайтесь: заслуживают ли ваш профессиональный рост и карьера такой стагнации?

История из жизни

Самой большой отдушиной для меня становилось понимание того, что вся магия, которую я могу воплотить в коде, только в моих руках. Ничто извне не может помешать мне написать новое, создать что-то интересное из ничего, из набора слов и цифр. Когда мне начинало казаться, что я просто занимаю определенное место в скучном, большом механизме, я начинал писать для себя, по вечерам, после работы, ночью. И это всегда помогало. Я старался никогда не позволять себе быть узником собственных мыслей.

Кроличья нора

Любой профессиональный разработчик знает: лучшее решение получается только тогда, когда вы располагаете максимумом информации о проблеме, которую пытаетесь решить. Каждое требование к задаче, каждый элемент архитектуры новой системы, каждая ошибка в проекте требуют от вас владения полной информацией. Ваш профессиональный рост требует абсолютно того же.

Мир разработки программного обеспечения огромен. Еще совсем недавно он был достаточно обозрим и у разработчиков был шанс знать о программном обеспечении если не все, то почти все. В наши дни эта цель уже недостижима.

Начинающим разработчикам часто становится не по себе, когда они понимают, что им открывается только малая часть того, что ждет их впереди. Им кажется, что разработка программного обеспечения необъятна. Информации много, слишком много для восприятия, и хуже того, она устаревает быстрее, чем мы успеваем ее усвоить. Новые языки программирования, новые технологии, новые подходы, новые инструменты – в какой-то момент этот шквал новизны может сбить с ног. Не отчаивайтесь, это пройдет. Честно.

В самом начале карьеры вы должны сосредоточить свое внимание только на одном: на самообразовании. Учитесь, читайте, пробуйте, ошибайтесь. Если вы боитесь высоты, но при этом идете по узкой горной тропинке, не позволяйте себе смотреть вниз. У вас всего одна задача: шаг за шагом двигаться вперед. Шаг вперед, еще шаг, еще шажок.

Выберите технологию, которая вам нравится. Занимайтесь ею, изучайте ее. Интерес к этой технологии будет поддерживать вашу мотивацию. Не переживайте и не думайте, что упускаете что-то в мире IT, – наоборот, с каждым шагом вы становитесь опытнее и профессиональнее. Если в какой-то момент привычные рамки покажутся вам тесными, выйдите за их пределы: выберите новый язык программирования, поучаствуйте в open source проекте. Расширяйте границы своего опыта.

Каждый следующий шаг принесет вам новые знания, узкая тропка будет расширяться, а страх перед огромным и неизвестным – ослабевать. В какой-то момент вы сможете с уверенностью осмотреться вокруг и увидеть, что весь этот огромный мир, который изначально так пугал, на самом деле работает как цельный механизм с уже понятными вам законами и правилами.

Вы увидите, как новые технологии используют подходы старых, добавляя что-то свое. Как языки программирования наследуют удачные решения своих предшественников. Вы поймете, что каждый опыт познания на вашем пути добавлял крупицу понимания того, как устроен мир разработки программного обеспечения.

Ни один ваш шаг не окажется лишним. Сегодня вы узнаете что-то новое, а потом сможете наблюдать, как оно изменяется, эволюционирует. Как свежая идея, добавленная в новый язык программирования, спустя 10 лет становится стандартом индустрии. Как маленькая библиотека, которая должна была облегчать работу одного разработчика, превращается в полноценный инструмент, которым пользуются разработчики по всему миру. То, что вы узнаёте, облегчает весь ваш дальнейший путь. Эти знания позволят вам с каждым шагом двигаться все увереннее и быстрее. Вы начнете видеть связи между технологиями, единые правила, по которым они работают, и наконец почувствуете радость от пребывания в таком огромном и интересном мире.

Никогда не останавливайтесь в изучении нового. Если мы не развиваемся в сфере разработки программного обеспечения, весь наш опыт, все наши знания начинают растворяться. Это постоянно изменяющийся мир, и вы должны всегда двигаться вперед вместе с ним.

Тезисы

- Мир разработки программного обеспечения огромен.
- Весь этот огромный мир работает по очень похожим правилам.
- Впитывайте знания, собирайте пазл.
- Никогда не переставайте узнавать новое.

Задание

Найдите язык программирования, который вам понравится, но с которым вы еще никогда не имели дела. Вас может привлечь синтаксис, оформление документации – все что угодно. Попробуйте уделить этому языку программирования неделю своего времени (не в ущерб работе, но искренне пытаясь разобраться в нем). Попытайтесь увидеть в нем принципы, которые уже вам знакомы (я не про синтаксис языка, нас с вами не удивит наличие в нем конструкции if). Если во время изучения вам действительно понравится этот язык, попробуйте использовать его для себя, присоединиться к его сообществу.

История из жизни

Мое знакомство с программированием началось в детстве, когда я нашел дома книгу по языку Си. До сих пор не знаю, откуда она взялась: никто в моей семье не увлекался ни компьютерами, ни программированием. Мне было, кажется, 12, и я очень любил все связанное с компьютерами. Своего компьютера у меня не было, и я начал читать найденную книгу как эзотерический трактат, представляя, что я мог бы изменить в коде той или иной программы, если бы это было возможно. Мне казалось, что с годами это любопытство, жажда узнать, что там, на следующем шагу, ослабнет, но это ощущение так и остается при мне. Каждый день мне по-прежнему не терпится узнать, насколько глубоко ведет эта кроличья нора.

Пройдет и это

На своем пути разработчика вы будете попадать в самые разные истории и сталкиваться с самыми разными ситуациями. Взлеты и падения, большие удачи и большие разочарования, конфликты и внезапные озарения. Вас будет удивлять код, вас будут удивлять люди (не всегда приятно, к сожалению), вы будете удивлять себя сами (временами очень сильно).

Возможно, вы не поверите в то, что я сейчас скажу, но хотя бы запомните эту мысль и постарайтесь пожить с ней, применить ее на практике, дать ей шанс. Происходящие с нами события не определяют наше дальнейшее поведение. Приведу пример. Допустим, у нас есть два сотрудника, которых решили уволить. Само событие увольнения для них абсолютно одно и то же, но один расценит его как проблему, необходимость поиска новой работы, профессиональную неудачу. Второй же увидит возможность найти более интересную компанию, проект и должность, получить новый опыт работы.

Одно и то же событие может вызывать совершенно разную реакцию у разных людей. Дело тут, разумеется, не в самих людях, а в том, что они видят в этом событии. Кто-то расценит увольнение как трагедию, повод усомниться в своих силах. А кто-то увидит в этом новые возможности. Решить, как относиться к ситуации, можете только вы сами.

Я не предлагаю вам составить карту желаний и каждое утро перед зеркалом повторять себе: «Я замечательный». Я просто не верю в подобные техники. Но верю в то, что мы способны выбирать, как реагировать на события, происходящие в нашей жизни. Поначалу это будет непросто. ОЧЕНЬ непросто. Но если вы действительно постараетесь, уверяю вас, с каждым разом у вас станет получаться все лучше и лучше.

Помните: не все, что происходит в нашей жизни, по-настоящему важно. Но современный мир настолько требователен и стремителен, что кажется, будто любое событие ставит нас перед выбором: жизнь или смерть. Это не так. Вы в ответе за свою жизнь, за свои эмоции. Не давайте событиям управлять вами.

И не забывайте: абсолютно все события – хорошие, плохие, неожиданные или запланированные – дают вам новый опыт и новое видение жизни. Некоторые события помогут вам, некоторые вы просто пропустите как незначительные, но многие станут невероятно полезной практикой и толчком к чему-то новому.

Тезисы

- Мы в ответе за то, как реагируем на жизненные события.
- Учитесь реагировать так, как хотели бы, а не так, как получается.
- Не позволяйте событиям управлять вашей жизнью, какие бы трудности они за собой ни влекли.

Задание

Возьмите листок бумаги и напишите список ситуаций или событий, которые регулярно вас расстраивают. Коллега любит подшутить над вашим любимым языком программирования? Добавляем. Продакт-менеджер забывает сбросить капсулу из кофе-машины, когда заходит к вам в офис? Пишем. Рядом с каждым событием запишите свою обычную реакцию, которой вы недовольны. Вспомните, как часто вас расстраивали такие события. Каждый раз вы реагировали одинаково, закрепляя этот опыт. Теперь необходимо решить, что вы измените в своем поведении, когда эта ситуация повторится. Вы каждый раз огорчаетесь, что коллеге не нравится ваш любимый язык программирования. Почему? Зачем вам нужно, чтобы коллега непременно полюбил этот язык? Может, с каждой новой шуткой вы начинаете

сомневаться в себе и в том, что этот язык программирования действительно хорош? Выберите адекватную реакцию, в которой не будет места огорчению. Можно надеть наушники и включить музыку, можно сходить и выпить кофе, а то и придумать блестящую остроту про язык программирования коллеги. Самое главное – прекратить реагировать так, как вы привыкли.

История из жизни

Я очень долго не мог принять реальность в том виде, в каком она есть. В виде хаоса, неопределенности, непостоянства. Возможно, это прозвучит глупо, но мне потребовались именно годы, чтобы понять, что реальность не станет сама собой меняться мне в угоду. А если я стану сражаться с ней как Дон Кихот, то просто буду терять все больше и больше сил – с каждым годом, с каждой новой схваткой. Моя работа оказалась плохим помощником в осознании этого факта: она была логичной, она была воспроизводимой, от нее я мог ожидать одних и тех же результатов при идентичных начальных данных. Попытки провернуть тот же трюк с жизнью всегда заканчивались провалом. Отчасти я стал заложником того, что умею лучше всего: я просто проецировал то, чем восхищаюсь в программировании, на хаос. На этот конфликт я потратил немало лет и нервов, но, кажется, наконец научился разделять два разных мира: восхищаться красотой кода и с радостью принимать неопределенность бытия. И это очень приятные ощущения.

Хвали себя

Неспособность похвалить себя – одна из самых страшных болезней профессионалов. Если вы еще не слышали про синдром самозванца, самое время прочитать про него. Я не претендую на знание статистики, но практически все IT-профессионалы, которых я встречал, страдали этим недугом. Отчасти это связано с тем, что перфекционизм – неотъемлемая часть нашей профессии. Отчасти – с тем, что любой профессиональный разработчик постоянно развивается, и этот нескончаемый поток новых знаний не дает ему времени прерваться и сказать себе, что он (вообще-то!) молодец.

Многие зараженные этой болезнью считают, что проблемы не существует, потому что были воспитаны на фразах «пусть тебя хвалят другие», «хвалить себя стыдно» и т. д. Многие считают, что строгость и даже жестокость к себе мобилизуют их, делают лучше как специалистов, помогают прыгнуть выше головы. Это самообман. Если мы не хвалим себя, то лишаемся гордости за свой труд, самоуважения за свои достижения и успехи.

Проблема гораздо шире, чем эмоции отдельных людей. Компаниям, как правило, невыгодно хвалить своих сотрудников, а уж делать это персонализированно – тем более. Компании предпочитают поощрять сотрудников деньгами, но деньги никогда не станут достаточным мотиватором. В конце рабочего дня (недели, месяца, пятилетия – подчеркните нужное) мы остаемся один на один с собственной усталостью и неуверенностью. Эти ощущения растут с каждой сложной задачей, с каждой переработкой и дедлайном, приближая нас к выгоранию и депрессии.

Выкиньте в мусор установки родителей о том, как и за что вас надо хвалить. Выкиньте в мусор фразу «то, что не убивает нас, делает нас сильнее». Вы не перестанете быть профессионалом, если признаете свои заслуги. Хвалите себя. Вы каждый день создаете что-то из ничего. Вы каждый день двигаете человечество вперед (нет, это не высокопарные слова, вы точно внимательно читаете эту тему?). Ваши достижения – результат невероятного труда и опыта. Если вы не будете хвалить себя сами, то придется вечно ждать этого от кого-то другого.

Начните с себя, но не останавливайтесь на этом. Когда вы достаточно продвинетесь в своей карьере и будете работать на руководящей должности, продолжайте следовать этому правилу. Хвалите коллег, когда они хорошо выполняют работу, хвалите их, когда понимаете, что это им нужно. Делайте это искренне по отношению к себе и к другим людям.

Тезисы

■ Тот, кто сказал вам, что хвалить себя некрасиво, соврал.

■ Делайте паузу и говорите себе, что вы молодец, даже если не чувствуете этого.

■ Умение похвалить себя – это умение похвалить и другого.

Задание

Думаю, вы догадываетесь, что я попрошу вас сделать. Хвалите себя! Написали классный компонент – скажите себе спасибо. Разработали биллинг для своего проекта – немедленное спасибо. Не бойтесь, что вы себя перехвалите. Если вам трудно похвалить себя за хорошую работу, то перехвалить себя вам не удастся. Будьте честным с собой, уважайте и цените свой труд.

История из жизни

Я долго учился хвалить себя. Если быть совсем точным, то я учился хвалить себя ОЧЕНЬ ДОЛГО И МУЧИТЕЛЬНО. Потребовались годы, чтобы

я перестал считать, что чем больше буду себя упрекать за неидеальную работу, тем сильнее буду становиться как разработчик. Наглая, отвратительная ложь, которой я пичкал и пичкал себя, пока не становилось тошно. Нет, я не встаю перед зеркалом каждое утро и не произношу: «Да что же это у нас тут за замечательный программист, да ты же мое солнышко». Для меня умение хвалить себя стало концом постоянных самоупреков, концом вечной неудовлетворенности своей работой. Я научился воспринимать ее отстраненно. С уважением, со вниманием, с конструктивной критикой, если в ней есть что улучшить.

Перфекционизм (и как от него не спятить)

Я думаю, ни для кого не секрет, что разработчики и перфекционизм неразрывно связаны. Можно долго спорить о том, делает ли разработчика перфекционистом профессия или же, наоборот, перфекционисты часто становятся разработчиками, но это не главный вопрос. Главный вопрос заключается в том, как жить и работать, не позволяя перфекционизму отравлять наше существование.

В стремлении сделать свою работу качественно не может быть ничего плохого. Однако перфекционист перестает понимать разницу между «качественно» и «идеально». Каждое решение, каждая новая строчка кода начинает проходить безжалостную, жестокую цензуру. Идеальное ли это решение? Учел ли я абсолютно все ошибки, которые могут возникнуть в процессе работы этого кода?

Возможно, кто-то из вас не согласится со мной («Эй, а разве это не ты писал, как важно делать свою работу качественно?»). Поймите меня правильно. Есть огромная разница между тем, чтобы выполнять работу качественно, и тем, чтобы с сумасшедшим упорством доводить ее до несуществующего совершенства.

Вероятно, вас интересует, как отличить в себе здоровый подход к работе от фанатичной погони за недостижимым идеалом. Для этого потребуется определенный опыт и умение быть честным с самим собой. Общее правило можно сформулировать так: если вы раз за разом возвращаетесь к написанному решению, переделываете его без четкого понимания, что именно вы улучшили, – бдите. Если вы тратите огромное количество времени на выбор одного из вариантов, притом что ни один из них не имеет объективных преимуществ, – будьте настороже.

Не пытайтесь искоренить свой перфекционизм. Если это черта характера, то избавиться от него полностью невозможно, да и не нужно. В основе перфекционизма лежит возведенное в абсолют желание делать свою работу максимально качественно и профессионально. Оно становится неврозом, мешает вам трезво оценивать свой труд, заставляет беспричинно переживать и терять время. Учитесь контролировать перфекционизм, используйте его себе во благо. Старайтесь найти лучшее решение, а когда нашли, остановитесь на нем. Если выбираете один из вариантов, объективно оцените каждый и выберите тот, который считаете лучшим, не возвращаясь к решению снова и снова.

Наша сила не только в победах, но и в поражениях. Пытаясь создать нечто идеальное, вы отрываете себя от реальности, от ее плюсов и минусов. Вы стараетесь найти то, чего не существует. Дайте себе шанс сделать все отлично, но и не лишайте себя права на ошибку. Ошибки часто учат нас куда лучше, чем успехи. Перфекционизм будет преследовать вас не только в работе, он может осложнять любую сферу жизни. Попробуйте приручить его, начав с работы, взять его под контроль, извлекать из него пользу.

Тезисы

- Желание выполнить работу качественно – прекрасная черта.
- Перфекционизм – неспособность увидеть разницу между качественным и идеальным решением.
- Не пытайтесь избавиться от перфекционизма, прибегайте к нему в разумных пределах.
- Держитесь за реальность, не позволяйте фантазиям об идеальных решениях сбить вас с толку.

Задание

Постарайтесь честно ответить себе на вопрос: перфекционист ли вы? Замечаете ли вы за собой, что тратите слишком много времени на поиск

идеального решения? Испытываете ли почти физическое напряжение, когда нужно выбрать один из очень похожих вариантов? Поступайте от обратного. Нашли работающее решение, которое вас устроило первым? Используйте его и заставьте себя не возвращаться к этому вопросу. Вам необходимо выбрать из нескольких вариантов, и ни один из них не имеет объективных преимуществ? Бросайте монетку, делайте выбор и уже не возвращайтесь к нему. Это может показаться контринтуитивным, но, поверьте, это лучшая помощь, которую вы можете себе оказать. Старайтесь бороться с привычкой погружаться в поиск идеала, разрушайте ее быстрыми и уверенными решениями. Это будет сложно, но с каждым разом вы будете делать выбор все увереннее и сможете наконец использовать весь потенциал перфекционизма без его минусов.

История из жизни

Это может показаться забавным, но от перфекционизма мне помогло избавиться то, что я слишком много работал. Сюр, казалось бы, но когда нужно вести несколько проектов, попадать в дедлайны и при этом прыгать на одной ножке, чтобы клиент радовался, на невротический перфекционизм попросту не остается времени. И мне пришлось довольствоваться принципами «да, это сделано достаточно хорошо», «этого хватит» и «мы сделаем это сейчас, у нас нет времени улучшать до бесконечности». В сутках всего 24 часа, и я предпочту сделать хорошо сегодня, вместо того чтобы дать волю перфекционизму и закончить через неделю.

Гордыня

Любой опытный разработчик в какой-то момент начинает чувствовать себя профессионалом. Уверенность растет с каждым успешным проектом, с каждым удачным выпуском продукта на рынок, с персональными достижениями в работе. Чувствовать себя уверенно и ощущать гордость за себя – прекрасно. Однако в этом есть ложка дегтя, которая может отравить всю вашу радость.

Речь идет о гордыне – чувстве, которое граничит со здоровой гордостью за себя и способностью принимать и уважать свой опыт и заслуги. Гордыня заставляет нас ставить себя выше других людей, считать, что только мы знаем, как сделать что-то лучше, как добиться того, чего другие (по нашему мнению) не смогут. Вы определенно можете быть бриллиантом своей компании и действительно разбираться во многих вопросах лучше коллег. Но само чувство иллюзорного превосходства над другими способно отравить не только вашу жизнь, но и жизнь людей вокруг вас.

Гордыня появляется настолько незаметно, что мы часто даже не отдаем себе отчет в том, что она уже сидит у нас на плечах. Ваши замечания коллегам становятся острее, участие в обсуждениях превращается в ваш монолог, где все должны слушать только вас. Вы начинаете ощущать, что все, что происходит на проекте, – результат вашей работы, ваших идей и ваших решений.

Гордыня – это башня из слоновой кости, на которую мы поднимаемся благодаря своему опыту и профессионализму. Поднимаемся так высоко, что забываем, как выглядела лестница и как много прекрасных людей встретили по пути. Гордыня поджидает нас наверху, и как только мы попадаем в ее цепкие лапы, мы становимся хуже, в первую очередь как профессионалы. Гордыня заставляет нас терять объективность, ведь теперь самое важное для нас – наш собственный опыт. Гордыня заставляет нас игнорировать мнения других людей: зачем они, если мы можем лишний раз послушать себя!

Я бы мог сказать, что гордыня обойдет вас стороной, но это будет неправдой. Если вы преданы своему делу и отдаете ему себя целиком, то однажды почувствуете себя непогрешимым. Пожалуйста, отнеситесь со вниманием к этой внутренней слабости. Вспомните, какой долгий путь вы прошли. Вспомните, какую помощь вам оказывали и сколь многому вы научились у других людей. Вы можете быть десять раз гением, но навсегда стать одиноким гением в башне – сомнительная радость.

Если вас не убедили мои гуманистические замечания (да что с вами не так?!) или вы не желаете ни с кем делить свою гордыню, то помните еще об одной опасности, которая ждет вас на вершине. Вы просто не сможете всегда быть правым. И в какой-то момент (возможно, это будет очень скоро, а возможно, через много лет) вы ошибетесь. Ошибетесь очень-очень сильно. И покатитесь по ступеням этой башни вниз. Подаст ли кто-нибудь вам руку, чтобы остановить падение, зависит только от ваших сегодняшних поступков. Оберегайте и уважайте свою гордость, но всегда помните, что понадобится всего лишь шаг, чтобы она превратилась в жгучий и разрушающий вашу жизнь и карьеру яд.

Тезисы

- Гордость за себя – прекрасное чувство, гордыня – безусловное зло.
- Гордыня возникает в нас незаметно.
- Гордыня нанесет вам удар в спину в самый неподходящий момент.

Задание

Отмечайте, достаточно ли вы гордитесь своими достижениями. Способны ли остановиться после решения сложной задачи и сказать себе, что

гордитесь собой и своей работой? Заведите себе привычку отмечать каждый важный этап своей работы, хвалить себя и ценить труд, в который вы вложили столько сил и знаний. Если в какой-то момент вы почувствуете, что начинаете превращаться в диктатора, что коллегам трудно обсуждать с вами рабочие вопросы, что вы чаще апеллируете к своему опыту и авторитету, чем к фактам и событиям, – остановитесь и признайтесь себе, что в вас созрела гордыня. Постарайтесь хотя бы просто принять этот факт (я гарантирую, ваша гордыня будет не в восторге). Начните общаться с людьми, прислушивайтесь к их идеям и подходам. Забудьте о том, что можете протолкнуть свое решение благодаря авторитету. Вспомните, почему вы вообще стали разработчиком, какие чувства переполняли вас тогда – интерес, любопытство, восторг от создания чего-то стоящего из ничего. Не лишайте себя этих прекрасных ощущений в угоду гордыне, которая остановит ваш профессиональный рост и оставит вас в одиночестве.

История из жизни

За каждый момент гордыни я всегда платил. Разрывом рабочих отношений, ошибками в программных продуктах, ударом по репутации, финансовыми потерями. Я бы и рад сказать, что быстро усваивал уроки, но нет. Признаться себе в том, что ты возгордился, нелегко, равно как и исправлять свои ошибки. Однако это единственный путь, который не усыпан граблями и разочарованием. Наше эго – это определенно не то, что надо взращивать за чужой счет.

Pet projects

Будем откровенны: даже лучшие рабочие проекты, с которыми вы столкнетесь, могут надоесть. Возможно, они потребуют от вас длительной монотонной работы или будут использовать технологический стек, который вас не вдохновляет. А возможно, вы уже проработали над этими проектами так долго, что они вам просто смертельно наскучили.

Не стоит отчаиваться! Когда почувствуете, что вас начинает тянуть к чему-то новому, но бросать свою работу вы, естественно, не планируете, найдите себе pet project. Возможно, у вас давно зреет идея, которую хотелось бы реализовать. Может, вы поработали с каким-то open source проектом, пришли от него в восторг и хотите поучаствовать в его развитии. Или просто жаждете попробовать язык программирования, синтаксис которого приснился вам в эротическом сне (не могу сказать, случалось ли такое с кем-нибудь, но и обратного гарантировать не могу).

Pet projects, как и любая деятельность, не связанная с вашей основной работой, – это огромный новый опыт и расширение навыков. Особенностью таких проектов (будь то создание личного меганавороченного калькулятора расходов или участие в работе над open source продуктом) является то, что вы испытываете огромное желание заняться ими «прямо сейчас». Вы мотивированны, вам интересно, вы испытываете положительные эмоции от работы с такими проектами. Эти положительные эмоции крайне полезны и помогают воспринимать новые знания быстрее и качественнее.

Главная проблема, связанная с pet projects, – время. Ваше бедное свободное время. Если соблюдать баланс между трудом и отдыхом (А Я ОЧЕНЬ НАДЕЮСЬ НА ЭТО!), то у вас в распоряжении остается не так много свободного времени, которое можно посвятить чему-то очень похожему на вашу основную работу. Я совру, если скажу, что не проводил выходные и вечера после рабочего дня за новым языком программирования или проектом, который был мне очень интересен, поэтому не стану отговаривать и вас. В конце концов, то, как вы отдыхаете и от чего получаете удовольствие, – это только ваше дело, ничье больше.

Pet projects могут быть очень полезны для вас. Они снимают умственное напряжение от основной работы, помогают мозгам не ржаветь, дают новые знания, новые подходы и опыт, радуют, когда вы добиваетесь прогресса. Черт, они могут даже помочь справиться с выгоранием, если вы найдете проект, вызывающий те же чувства любопытства и восторга, с которыми вы пришли в эту индустрию.

Не расстраивайтесь, если прямо сейчас слишком загружены работой, чтобы заниматься дополнительным проектом. Интерес и тяга к новому всегда с вами, просто дайте себе время, чтобы закончить дела.

Тезисы

■ От любого проекта можно устать.

■ Pet projects – простой способ поднять себе настроение и вернуть мотивацию.

■ Работая над проектами, которые доставляют удовольствие, вы значительно быстрее получаете новые знания.

■ Если у вас не хватает времени на свой pet project, не расстраивайтесь, просто включите его в свое расписание заранее.

Задание

Составьте список из 5 или 10 технологий, языков программирования или сфер IT, которые вам хотелось бы узнать получше. Держите этот список под рукой, обновляя его, если необходимо. Когда вы почувствуете, что устаете

от рабочего проекта, достаньте список, подумайте, что бы вы могли сделать, используя эти технологии, какой проект могли бы написать, к примеру, за выходные. Выделите себе немного времени после работы, чтобы составить план будущего проекта. Если вас начало затягивать, если вы чувствуете мотивацию – пробуйте!

История из жизни

Я уже не перечислю всех мини-проектов, которыми занимался многие годы, но мне запомнился один, ставший для меня отдушиной в череде очень сложных проектов, когда мне казалось, что я полностью заржавел. Я настолько устал от стека технологий и языков программирования, которые мы использовали, что мне нужна была «свежая кровь» – область, в которой я мало разбирался, то, что стало бы для меня испытанием. Я решил написать эмулятор CHIP8 на языке Rust. Да, я был вымотан и очень устал, но даже полчаса работы над этим маленьким проектом в день давали мне невероятное чувство радости, которое я помню до сих пор.

Аврал! Свистать всех наверх!

Почти в любом IT-проекте чередуются стадии спокойной, размеренной разработки и авральной, бешеной работы. Чаще всего количество и продолжительность таких стадий напрямую зависят от качества управления проектом, и в большинстве случаев вы никак не сможете на это повлиять. Если вы работаете в игровой индустрии, то я даже не буду объяснять, что такое crunch time, вы очень скоро познакомитесь с этим сами.

С логической точки зрения мы понимаем, что спокойная, размеренная работа – это то, к чему надо стремиться в разработке программных продуктов. Вдумчиво, последовательно, без спешки и с учетом требований. Однако реальность коммерческой разработки никогда такой не была и вряд ли станет. Конкуренты всегда будут дышать в спину вашего проекта, а пиар-отдел, продвигающий конкретную дату выхода продукта на рынок, не будет менять ее в процессе рекламной кампании. Да, авральная работа над проектом – зло, но чаще всего оно неизбежно, и вам, как профессионалу, надо быть к этому готовым.

Работая в экстренном режиме вместе с командой, вы сможете почувствовать необычайный прилив адреналина – ощущение, что работаете «на грани», чувство сплоченности в погоне за результатом. Это чувство напоминает охоту, где добыча – ускользающая цель. Такой целью может стать объявленная дата очередного релиза, запуск проекта для конкретного клиента или первый выход продукта на рынок. Иногда команде действительно надо сделать последний рывок, который даст жизнь проекту или откроет широкие возможности для его роста.

Переживание этой гонки, этой охоты может быть приятным, но вы должны помнить, что оно временно. На смену ему придут усталость, растерянность и нередко разочарование. Охота должна заканчиваться поимкой добычи, а если погоня длится и длится, значит, вы просто потерялись в лесу.

Другое свойство авральной работы – то, что, если практиковать ее часто, накапливается усталость, поначалу незаметная. В общей атмосфере близости дня икс вы не ощущаете проблемы, а тем временем усталость растет, ожидая момента, когда вы наконец сможете остаться наедине с собой. И тогда – хлоп! Она вас настигнет.

Авральная работа может привести к хронической усталости, к депрессии, но чаще всего она служит причиной профессионального выгорания. Помните об этой темной стороне: даже во время адреналиновых забегов перед новым релизом умейте остановиться и передохнуть.

Стоит упомянуть, что существуют компании с нездоровым (больным на всю голову) менеджментом. С начальством, которое считает, что авральная работа стимулирует сотрудников на большие достижения. В таких компаниях авралы создаются и поддерживаются искусственно, вынуждая сотрудников работать на износ, без отдыха, в непреходящем стрессе. Если вы поняли, что находитесь в такой компании, я искренне советую задуматься о смене работы. Да, какое-то время вы будете чувствовать адреналиновый подъем и мотивацию, но, поверьте, это вас сожрет. Постоянный стресс и работа на пределе возможностей не делают вас лучшим специалистом, они делают вас вымотанным невротиком.

Тезисы

- В реальном мире разработка программного продукта редко бывает размеренной и спокойной.
- Выброс адреналина от авральной работы может быть очень приятным, но это иллюзия.
- Подъем сил и радость от «охоты» сменяются горечью и разочарованием.
- Авральная работа – первый шаг к выгоранию и депрессии.

Задание

Когда на проекте в очередной раз случится внезапная, экстренная необходимость поработать в выходные или сверхурочно, попробуйте оценить, насколько это критично. Если вас ждет важный релиз или продукт должен вот-вот выйти на рынок, все понятно, вопросов нет. Однако если вы видите, что авралы случаются каждую неделю, а менеджеры продолжают рассказывать сказки о том, как важно добавить эти 13 новых функций в продукт именно сейчас или ВСЕ, КОНЕЦ, ПРОЕКТУ КОНЕЦ, – подумайте, насколько с вами честны.

История из жизни

К сожалению, я именно тот человек, которого будят в 4 утра из-за того, что рабочие сервера перестали отвечать. Не потому, что я хорошо разбираюсь в DevOps или мне придется их немедленно реанимировать (для этого разбудят администраторов), но мне нужно присутствовать, потому что это мои проекты, моя ответственность, мои решения. Рад ли я таким побудкам? Конечно, нет, однако их частота напрямую зависит от того, хорошо ли я делаю свою работу каждый день.

Свободное время

Если, придя в IT-индустрию, вы думаете, что всегда сможете работать по жесткому графику, подумайте еще раз. Я не хочу сказать, что в любой компании и на любой должности вам придется задерживаться в офисе, а иногда работать по ночам, но не могу сказать и обратного. Иногда вы искренне захотите потратить свободное время на работу – возможно, на новый проект или новую идею, которую планируете реализовать.

А теперь поговорим о вашем свободном времени. О его наличии или отсутствии. Шутки в сторону, у вас должно быть свободное время. Время, принадлежащее только вам. Время, не занятое основной работой, даже если вы любите рабочий проект как собственного ребенка (так бывает, да). Если, закончив работу, вы продолжаете думать о ней до начала следующего рабочего дня, это верный признак того, что вы попали в ловушку и нужно искать выход.

Многие разработчики пренебрегают свободным временем, полагаясь на адреналиновый раж, который возникает при работе над крупным проектом в сжатые сроки. Предпочитают потратить любое доступное время на то, чтобы сделать свой код лучше, стабильнее, написать новое решение. Я мог бы сказать, что это прямой путь к выгоранию, если бы сам не поступал так же бесчисленное количество раз.

И только поэтому я настаиваю, чтобы вы внимательно следили за тем, как тратите самый ценный ресурс. Да, вы не напишете этот алгоритм сегодня, да, вы не почините этот мерзкий гейзенбаг, который сводил вас с ума последнюю неделю, но вы сможете дать своим мозгам (своим прекрасным, вкусным мозгам) передохнуть. И эта короткая передышка поможет вам куда больше, чем бессонная ночь за монитором.

На что тратить свое свободное время, решать только вам. Постарайтесь радовать себя, насколько это возможно. Труд разработчика сложен, он требует огромного количества энергии и концентрации. Поэтому совершенно очевидно, что вашему мозгу (как и телу) нужно время, когда его не используют как загнанную лошадь на бесконечном поле, которое надо перепахать.

Справедливым будет сказать, что совершенно нормально, если в свободное время вы решите изучить что-то связанное с разработкой программного обеспечения. Not great, not terrible, просто постарайтесь честно ответить себе на вопрос: доставляет ли вам удовольствие то, как вы проводите свободное время, или в вас так много адреналина от работы, что вы чувствуете, что просто ОБЯЗАНЫ заниматься саморазвитием в любую свободную минуту?

Тезисы

- Фиксированный график – это замечательно, но часто несбыточно.
- Вам критически необходимо время, свободное от работы.
- Только вы знаете, на что тратить свободное время.

Задание

Попробуйте проанализировать, как вы проводите время вне работы. Занимаетесь ли вы тем, что позволяет разгрузить мозг? Даете ли вы себе возможность отвлечься от постоянного анализа, от поглощения новых знаний, от размышлений о том, какое решение выбрать или как написать особо сложный компонент? Если вы не можете перестать думать о работе, даже лежа в джакузи, тормозите. Пресекайте любые попытки мозга переключиться на рабочую частоту. Если дома в воскресный день вас посещает внезапная мысль о том, что выборку из базы данных нужно было переделать на использование курсоров, – пойдите и засуньте голову под прохладный душ. Используйте физические раздражители, чтобы переключить внимание.

Отправьтесь на пробежку, сыграйте в дженгу, попробуйте научиться жонглировать.

История из жизни

Возможно, я бы рассказал вам чуть больше о своем свободном времени, если бы мне прямо сейчас не надо было бежать и формировать список задач на новый релиз! Это только отчасти шутка. Эту книгу я писал по большей части ночами, нередко от бессонницы. Я все еще стараюсь занимать любое образовавшееся у меня свободное время. Нередко я заполняю его программированием, но советовать вам поступать так же у меня не повернется язык.

Я работаю ради…

Денег! Шутка. Отчасти. Разумеется, мы устраиваемся на работу, чтобы применять свои знания на практике, развивать навыки, приобретать новый опыт и получать деньги. Но деньги никогда не станут единственным мотиватором для работы. Разработчики, которых интересует исключительно зарплата, не задерживаются в индустрии надолго.

В начале карьеры вы, скорее всего, будете неотчетливо представлять, для чего именно вы работаете. Вас будет захватывать новый опыт, новая должность, новый уровень дохода. На время это может стать мотивацией, но чем больше лет вы проведете в индустрии, тем чаще будет возникать вопрос: а зачем я вообще занимаюсь разработкой?

Нужно быть честным с собой, и ответ, который вы найдете, – только ваш. Он может быть любым. Возможно, вам нравится решать сложные задачи и чувствовать себя победителем, а может, вы работаете ради людей, которым необходим ваш продукт. Со временем вы станете лучше понимать себя и свое место в IT. Вы уже будете знать, что вас восхищает и что расстраивает, из-за чего вы можете задержаться на работе и провести там все выходные.

Когда вы сумеете понять, что вами движет, доставляет удовольствие, мотивирует получать новые знания и опыт, вы станете сильнее как личность и профессионал. Понимание того, что для вас ценно, даст больше внутренней свободы и удовлетворения от работы. Вы сможете уделять больше внимания тому, что считаете важным, и меньше обращать внимания на неудачи.

Тезисы

- Существует то, ради чего вы работаете на самом деле.
- Ваши мотиваторы – только ваши и больше ничьи.
- Уважайте то, что движет вами.

Задание

Попробуйте обращать внимание на то, что вас действительно радует в работе. Вы счастливы, когда пользователи пишут положительные отзывы? Вы радуетесь как ребенок, когда ваш новый алгоритм оказался на 12 % быстрее предыдущей версии? Вы в восторге, что у вас новая должность и 15 разработчиков в подчинении? Будьте честными с собой, это только ваша правда, ваш опыт.

История из жизни

Ответ на вопрос «ради чего?» для меня всегда был простым. Я ужасно любопытный. Мне всегда хотелось узнать, как вещь устроена изнутри. Наверное, я бы не смог найти себе лучшего ремесла, чем разработка, даже если бы постарался. Да, мне важно знать, насколько успешен будет мой продукт. Да, я хотел бы, чтобы он помогал пользователям. Но то, ради чего я вообще занимаюсь разработкой, – это эгоистичное (в самом хорошем смысле) желание постоянно удовлетворять свое любопытство и каждый день узнавать, «а как же это работает?».

Удаленная работа

Давайте я скажу это сразу: удаленная работа подходит не всем. Все мы слишком разные, чтобы нам подходили одни и те же вещи. Я не буду пытаться классифицировать людей (это всегда провальная затея), потому что они чересчур сложны для какой-либо классификации, основанной на чертах личности. Однако совершенно очевидно, что у каждого человека есть набор предпочтений. Кто-то хочет находиться в гуще событий и среди людей. А кому-то трудно поддерживать социальные взаимодействия в коллективе. Ни то ни другое не делает вас лучше или хуже как специалиста, но определяет уровень комфорта и радости от работы.

Если вы уже какое-то время занимаетесь разработкой, это значит, что вы работаете либо в офисе, либо удаленно. Если вы попробовали оба варианта работы – мои поздравления, обладая таким опытом, вы уже знаете, какой формат ближе именно вам.

Работа в офисе располагает к общению, даже если вы замкнутый человек. Обсуждения, обеды, шутки, мероприятия – все это вовлекает вас в жизнь компании и коллег. Это замечательно, но только если вы не чувствуете, что тратите на это больше энергии, чем хотели бы. Не переживайте, если то, что я описываю, не относится к вам лично. Поверьте мне: хотя бы один из ваших коллег сейчас мечтает о том, чтобы продолжить работать, устроившись дома на диване (и не только потому, что у него на коленях уляжется милый котик). Кто-то будет чувствовать мотивацию и азарт от работы плечом к плечу с коллегами. Кого-то угнетает необходимость приходить в офис каждый день и пытаться участвовать в социальной жизни компании.

Как показал 2020 год (я не буду здороваться с коронавирусом, обойдется), у нас не всегда есть возможность выбирать, в каких условиях работать. Огромное количество разработчиков перешло на удаленную работу. И я могу только представить, насколько болезненно некоторые восприняли этот переход. Люди очень социальны, мы ищем взаимодействия с другими, кто-то больше, кто-то меньше, но ищем. Люди, привыкшие работать рядом с коллегами, иметь возможность прерваться на дружеский разговор или отвлечься на шутку, оказались замкнуты в пространстве, где никого нет. При этом было необходимо продолжать работать так же продуктивно, как и раньше. Именно в этот период многие окончательно определили, что им подходит больше: работа в социальной среде или работа в уединении.

Да, современная удаленная работа предполагает постоянный контакт с коллегами. Это могут быть конференции, обсуждения или просто дружеский чат, но, я думаю, вы отчетливо понимаете разницу между тем, как я вижу вас в метре от себя и как – на мониторе.

У удаленной работы не так много особенностей, которые стоило бы здесь перечислить, однако те, что есть, очень важны. Удаленная работа требует больше самоконтроля, концентрации и внимания. Если вы пробуете себя в удаленной работе после офисной, то сразу поймете, о чем я говорю. Вам необходимо будет принимать больше самостоятельных решений, нести за них больше ответственности.

Общение. Да, даже учитывая все, что я сказал выше, – общение, общение, еще раз общение. Рабочее общение. Вы должны четко понимать, что от вас требуется, и незамедлительно сообщать коллегам, когда в рабочем процессе что-то идет не так. Если чувствуете, что запутались, что начинаете отставать, – поговорите со старшими коллегами. Один короткий разговор может спасти вас от многих дней фрустрации и сорванных сроков.

График и переработки. Если вы перешли на удаленную работу недавно, то вам, вероятно, кажется, что рабочий день длится бесконечно. Время – очень коварный механизм, поэтому нужно внимательно следить за тем, чтобы ваш рабочий график совпадал с офисным. Вы начинаете работать в одно и то же время и заканчиваете через N часов. То, что вы работаете, сидя не в офисном кресле, а под домашним пледом, совсем не значит, что вы тратите на это меньше

сил. Следите за тем, чтобы прерываться на обед и разминку. Работая дома, слишком легко потерять ощущение времени.

Дом и работа. Достаточно распространенная проблема, особенно для людей, которым не приходилось подолгу работать удаленно. У вас может пропасть понимание, что дом – это дом, а не офис. Для людей, работающих дома, нет буферной фазы, когда можно переключиться с рабочего настроения на домашнее (этим буфером может быть поездка домой на метро, бар после работы или вечерний забег по магазинам). Если у вас есть такая возможность, организуйте в квартире отдельное рабочее пространство. Это может быть просто стол, главное, чтобы вы понимали, что стол – это работа, но стоит вам отойти от него, и вы дома, где царят приятные эмоции и уют.

Тезисы

■ Удаленная работа подходит не всем.

■ Попробуйте работать в офисе и удаленно: у вас должно быть представление о том и другом формате.

■ Исходите из особенностей своей личности и характера.

Задание

В качестве задания я бы попросил вас испробовать оба типа работы: в офисе и дома. Возможно, это покажет, какую роль играет для вашей работы окружение. Позволит узнать, насколько вы эффективны в разных условиях и насколько важны для вас социальные связи. Кому-то этот опыт поможет принять решение, где работать – в офисе или дома. Большинство современных компаний понимает проблему этого выбора и благосклонно относится к тому, что сотрудники хотят изменить формат работы. Не бойтесь говорить об этом со своими руководителями. В конечном итоге от вас требуется профессионализм, а то, насколько вам комфортно и как вы мотивированны, влияет на него напрямую.

История из жизни

Я очень люблю удаленную работу. Она позволяет мне писать код в тех условиях, которые обеспечивают мне максимальную продуктивность. Без офисных шуточек, без переключения внимания, без прокрастинации. Это очень субъективные характеристики, я не могу оценивать потребности каждого человека по себе, это было бы крайне глупо, но свой выбор я точно сделал. Да, порой удаленная работа требует куда больше усилий и силы воли, но, эй, хорошая работа всегда этого требует.

Это надо поправить

«Это место нужно будет доработать». «У клиента изменились требования». «Похоже, нам это не понадобится, надо убрать».

Большинство этих фраз будет преследовать вас постоянно. Требования клиентов будут меняться, реализованные функции не будут устраивать руководство, зеленый на главной странице сайта будет недостаточно зеленым, а алгоритм хеширования – недостаточно хешировать. Иными словами, однажды к вам придет человек и скажет: то, что вы выстрадали потом и кровью, больше не нужно. И это будет больно.

Дело в том, что если вы действительно любите разработку (я очень надеюсь на это), то, создавая что-то новое (проект, код, функцию), вы не просто пишете на экране буковки в определенной последовательности. Вы вкладываете в код часть своей любви к этому ремеслу, часть себя. Насколько болезненным может быть момент, когда вас просят убить эту часть? Стереть, будто ее и не было, либо переписать так, что вас там уже не останется? Иногда это бывает ОЧЕНЬ болезненно.

Возможно, вы считаете себя монументом спокойствия и подобные просьбы для вас – как с гуся вода. Я не хочу вас расстраивать, но это дело времени. Однажды у вас появится любимый проект, лучший алгоритм или самый прекрасный код, который вы написали, и вас попросят его «поправить».

В самой просьбе исправить код нет ничего особенного. Наверняка вы и сами просили кого-то об этом или участвовали в open source проекте, где авторов каждый день просят что-то изменить. Однако все меняется, когда речь идет о ВАШЕМ коде. О том, в который вы вложили столько сил, времени, личного опыта, энергии.

И… вы должны с этим смириться. Да, вам придется что-то поправить в коде, отказаться от каких-то своих решений и переписать их. Не относитесь к этому как к сомнениям в вашем профессионализме. Не относитесь к этому как к личному оскорблению. Постарайтесь взглянуть на проблему со стороны. Если у вас качается стул, потому что одна ножка подломилась, что вы предпочтете: починить его или продолжать сидеть, рискуя упасть и облиться кофе? Если вам перестали нравиться обои, будете терпеть их день за днем или поменяете на такие, которые вас радуют?

Да, ваш код важен для вас. Однако просьба изменить его – это не просьба забыть о нем, сделать вид, что его не существовало. Вы сделали его максимально качественно, но теперь он должен приобрести другую форму. Скорее всего, он станет только лучше – кому, как не вам, знать, что с ним сделать, чтобы он стал лучше. Не поддавайтесь унынию, если приходится удалять прекрасное, на ваш взгляд, решение. Да, придется пересмотреть его, но, возможно, именно тогда вы увидите еще лучшее решение.

Если требований об изменении кода слишком много или ваш клиент не может спокойно спать, пока не внесет хотя бы одно замечание по реализованным функциям, найдите себе отдушину. Заведите свой pet project, где только вы будете решать, как он будет выглядеть и каким правилам подчиняться.

Тезисы

- Однажды вас обязательно попросят изменить то, что вы создавали с любовью.
- Ваш код – это не вы сами; пока есть вы сами, вы напишете код еще лучше.
- Если ваш код постоянно подвергается нападкам, найдите себе отдушину.

Задание

Попробуйте понять, насколько болезненны для вас просьбы изменить или убрать ваш код. Как сильно вас задевает необходимость пересмотреть свои решения? Если считаете, что ваша реакция на такие просьбы неадекватна, поработайте над этим. Участвуйте в код-ревью, учитесь воспринимать просьбы об исправлениях не как попытку усомниться в вашем профессионализме, а как шанс сделать код еще лучше.

История из жизни

Мне было очень тяжело принимать исправления в своем коде, я долгое время воспринимал это именно как личную критику, раз за разом попадая в эту ловушку. Даже когда мне казалось, что я перешагнул через эту слабость, она снова давала о себе знать. Спасло меня то, что такое отношение никак мне не помогало. Мне все равно приходилось исправлять код, но каждый раз это стоило мне немалой нервотрепки. И я стал относиться к этому более прагматично, разделяя работу и связанный с ней стресс. Работа должна оставаться работой, а ваши эмоции – только вашими; эти пласты жизни определенно не стоит смешивать в очень горький коктейль.

Специалист широкого профиля

Разработка программного обеспечения многогранна. И чем больше развивается IT-индустрия, тем больше новых областей в ней появляется. Каждая область одновременно и похожа на другие, и представляет собой нечто совершенно особенное, со своими правилами, секретами и решениями. Разработка сетевого программного обеспечения чем-то похожа на создание игр, разработка драйверов – на программирование для встраиваемых систем. Однако чем дольше вы работаете в какой-нибудь специфичной области, тем больше отличий будете замечать.

Выбор направления работы полностью зависит от вас. Да, вначале вы, вероятно, будете заниматься всем подряд, чтобы получить базовый опыт, привыкнуть к разработке, индустрии, познакомиться с ее основными принципами и законами. Возможно, вам повезло и вы уже точно знаете, чем хотели бы заниматься. Это может быть разработка игр, криптография, большие данные, а может быть, вы всегда хотели написать свой язык программирования. В любом случае это должно быть тем, что вас восхищает. Тем, что вам бы хотелось изучить, разобраться в каждой мелочи, узнать, как все устроено.

Если вы пока не уверены в том, какое направление разработки вам нравится, потратьте некоторое время, поработав в разных областях. Участвуйте в open source проектах, выбирайте работу, которая может показать разные стороны индустрии. Возможно, вы сами удивитесь, поняв, к чему испытываете неподдельный, искренний интерес.

Разработчики программного обеспечения нередко меняют профиль работы: так люди меняют специальность, когда чувствуют, что больше не могут работать как раньше. Разработчик, занимающийся высоконагруженными веб-системами, вдруг понимает, что очень хочет писать игры. Разработчик игр внезапно осознает, что мечтает заняться низкоуровневым программированием для ядра Linux. Наше мировоззрение трансформируется с ростом опыта, мы начинаем находить интерес в областях, которые раньше казались скучными.

Все сказанное выше совершенно не означает, что вы должны выбрать один профиль и держаться за него руками и ногами. Нет ничего плохого в том, чтобы иметь представление о множестве сфер разработки. Более того, знания, полученные из разных направлений, делают вас куда опытнее, осведомленнее и профессиональнее. Однако справедливо и обратное. Знания, полученные в определенной предметной области, становятся очень ценными (и очень дорогими – да, я про денежки). Если вы выберете что-то действительно специфическое, то вполне можете оказаться одним из 10 человек на планете, у которых есть этот особый набор знаний и опыта.

Исходите из того, что вас восхищает. Руководствуйтесь тем, что вам интересно. Да, в какие-то моменты это будет непросто. Если вы захотите попробовать что-то новое, то придется искать другое место работы или проекты, к которым вы могли бы присоединиться. Первое время вас будет накрывать волнами новой информации, которую достаточно трудно воспринимать и запоминать. Но если вы действительно этого хотите, то обязательно добьетесь своего.

Тезисы

- В IT всегда найдется направление разработки, которое будет вам интересно.
- Не жалейте времени, попробуйте разные направления, узнайте, от чего получаете удовольствие.
- Путь к новым направлениям всегда открыт, вы в любой момент можете попробовать что-то другое.
- Следуйте за своими желаниями.

Задание

Выпишите 5 направлений разработки, которые вам интересны и не связаны с тем, чем вы сейчас занимаетесь. Расположите их в порядке убывания интереса. Посмотрите на полученный список и попробуйте описать шаги, которые приблизят вас к получению опыта по каждому из пунктов. Возможно, это будет участие в open source проекте, получение дополнительного образования или покупка онлайн-курса. Сделайте первые практические шаги к тому, чтобы попробовать себя в чем-то новом. Viam supervadet vadens[4].

История из жизни

Благодаря своему бездонному любопытству я успел покопаться в большинстве направлений IT, о чем никогда не жалел. На данный момент я занимаюсь архитектурой высоконагруженных, распределенных b2b-продуктов, но иногда спрашиваю себя: чем бы я занялся через 10 лет, если бы захотел сменить профиль? Ответа я еще не придумал. Возможно, захочу попробовать встраиваемые системы, микроконтроллеры и наконец научусь хорошо паять.

[4] Дорогу осилит идущий (*лат.*).

Новый проект

В карьере каждого разработчика рано или поздно наступает волнующий и очень важный момент: ему поручают создание нового проекта. И этот проект, который вы начали создавать с нуля, несомненно, запомнится вам на долгие годы.

Прежде всего – без паники. То, что необходимо сделать, вы делали уже много раз. Вы будете писать код (ура!). Просто не вкладывайте в эти страшные слова («новый проект») больше смысла, чем они содержат. Относитесь к новому проекту как к новому коду – ответственно, вдумчиво, но без страха.

Вы абсолютно точно будете чувствовать напряжение и груз ответственности, но постарайтесь запомнить следующее.

Новый проект на самом деле ничем не отличается от тех задач, которые вы получаете регулярно. Эта задача может быть более масштабной, но от этого она станет только интереснее.

Вы сможете работать так же, как и раньше. Сам факт, что вам поручили создание нового проекта, говорит о том, что ваша работа была профессиональной, качественной и эффективной. Вам доверяют, но доверяют не чему-то неожиданному, а тому, чем вы занимаетесь уже продолжительное время. Вы на правильном пути, просто делайте то, что и делали всегда.

Не позволяйте себя торопить. Специфика коммерческой разработки очевидна – продукт нужен «вчера», однако не разрешайте манипулировать собой и не поступайтесь качеством работы. Ваша задача как профессионала – написать качественный продукт, не стоит пытаться играть по чужим правилам и срезать углы. Хороший продукт проработает много лет, слабый будет закрыт – вот так все просто.

О требованиях к продукту: собирайте их, группируйте, работайте с экспертами в предметной области проекта, изучайте ее самостоятельно. Вы должны быть максимально ознакомлены со спецификой той области, для которой пишете продукт. Составляйте списки вопросов, терроризируйте заказчиков или менеджеров – их должно тошнить от вашей въедливости. Люди не всегда хорошо структурируют свои знания: забывают о важном, придают вес незначительному и т. д. Ваша задача на первом этапе – выделить главные и проблемные направления будущего продукта, область его применения и основные функции.

Не пугайтесь, когда вы будете сталкиваться с требованиями, конфликтующими между собой или не поддающимися описанию кодом. Вы в любом случае столкнулись бы с ними, но позже, когда время для анализа было бы уже упущено. Сбор требований должен формировать для вас картину будущего продукта, указывать на его сильные и слабые стороны, потенциальные проблемы и конкурентные преимущества. Будьте заранее готовы к тому, что, когда работа над проектом будет уже в самом разгаре, часть требований окажется неактуальной. Это распространенный сценарий, он не должен деморализовать вас. Любой продукт – живой, он будет развиваться и видоизменяться. Ваша задача – определить его возможности и рамки, в которых он будет существовать.

Ваши заказчики или эксперты в предметной области наверняка проанализировали рынок на предмет аналогичных продуктов. Но вам тоже будет нужно провести свое исследование, чтобы представлять, как работают существующие аналоги, насколько они удобны и каким требованиям соответствуют. В большинстве случаев вам будет доступна только пользовательская часть системы, поэтому постарайтесь извлечь из нее максимум сведений, обращая особое внимание на те моменты в ее работе, которые пересекаются с требованиями для вашей системы.

Отдельным важным шагом будет выбор технологического стека для вашего проекта: языка программирования, инструментов, компонентов и библиотек. Для многих разработчиков ощущения на этом этапе будут похожи на восторг ребенка в магазине игрушек. Получите от этого процесса максимум удовольствия, но не позволяйте эйфории взять верх – набирая

технологии в свой мешок с подарками, нужно исходить только из целесообразности их использования. Здесь как никогда важно дать приоритет проекту и его требованиям. Он будет зависеть от вас, от того, что вы выберете, – не подведите его, находите решения, которые помогут проекту стать успешным.

Ограничьте время анализа по конкретным вопросам, иначе вы рискуете столкнуться с аналитическим шоком и неспособностью принять решение. Если чувствуете, что анализ какого-либо вопроса занимает больше времени, чем должен, перейдите к следующему пункту. Если все равно сомневаетесь при каждом принятии решения, постарайтесь понять, говорит ли это ваша тревожность или есть объективные сомнения в правильности выбора. Тревожность необходимо просто принять: вы были бы отвратительным специалистом, если бы не чувствовали ответственности за свои действия. Объективные же сомнения можно отбросить, досконально проанализировав вопрос и выбрав максимально приемлемый вариант. Возможно, он не будет идеальным, но выбор уже сделан и вы больше не рефлексируете над ним.

Определите сроки работы с заказчиками или менеджерами, разбив весь объем на этапы (milestones), однако всегда помните о коварности установления строгих дат. Это крайне важно для новых проектов, где требования будут постоянно меняться и усложняться. Не пытайтесь никого впечатлить скоростью, на кону ваше время и хорошее самочувствие. Выделяйте для анализа и работы столько времени, сколько вам требуется. Я не привожу здесь подробного описания методологий разработки и способов планирования работы над проектом (их слишком много, а ваше время для меня ценно). Универсальный совет: уважайте свое время и не позволяйте дедлайнам сказываться на качестве работы.

Если вы чувствуете, что начали прокрастинировать, это абсолютно нормально и ожидаемо. Совет здесь будет только один: начинайте работать. Через силу, через усталость и нежелание, но начинайте работать по чуть-чуть, выполняя какие-то легкие, но необходимые задачи для нового проекта. Это может быть подготовка инфраструктуры приложения, организация загрузчиков или билд-системы. Черт, да вы можете потратить время на проработку формата вывода в лог, главное – начать делать хоть что-то, и вы обязательно втянетесь в работу в процессе.

Постарайтесь получить максимум удовольствия от нового проекта, пусть это будет захватывающе и интересно, пусть это станет испытанием. Новый проект даст вам новый опыт и знания, вы сможете создать что-то прекрасное из ничего – дайте себе шанс насладиться этим.

Тезисы

- Не паникуйте, получайте удовольствие от процесса.
- Досконально изучите предметную область.
- Узнайте максимум об аналогах и конкурентах.
- Не позволяйте эйфории от выбора технологий затмить требования проекта.
- Не позволяйте себе впадать в аналитический шок.
- Устанавливайте свои сроки и учитывайте приоритеты.

Задание

Если у вас есть время, представьте себе проект, который хотели бы реализовать. Определите его предметную область, его функции, сильные и слабые стороны. Составьте список формальных и неформальных требований к проекту. Определите технологический стек, который стали бы использовать, учитывая требования. Проведите анализ потенциальных проблем, с которыми можете столкнуться. Разбейте проект на этапы, определив ориентировочные сроки их завершения.

История из жизни

«Экзамен для меня всегда праздник, профессор». И это действительно так. Новый проект для меня всегда праздник. Мне немного страшно, но ощущение предвкушения, чувство нового начала куда сильнее. Я знаю, что первое время будет очень сложно. Время принятия самых важных решений, время формирования архитектуры, которая должна пройти проверку годами. Для меня начать новый проект – это все равно что, как в детстве, стоять перед нераспакованным новогодним подарком, испытывая огромную радость и предвкушая что-то очень интересное…

И напоследок…

Вот мы и подошли к концу моей (хотя теперь уже нашей) книги. Мне немного грустно, что она закончилась, но радостно, что вы ее прочитали. В эту книгу я вложил очень много себя, своих историй, опыта и переживаний. Спасибо, что разделили ее со мной.

Распоряжайтесь этой книгой как вам удобно. Может, вы прочитаете ее один раз и этого будет достаточно, но дайте ей шанс раскрыть себя. В ней много разделов, которые окажутся полезнее уже тогда, когда вы наберетесь опыта и будете лучше понимать мир IT.

А еще я хотел сказать, что верю в вас. Верю в ваше природное любопытство, в ваши интерес и любознательность. У вас все получится. Вам никогда больше не будет скучно, в IT вы всегда найдете что-то новое, что захватит вас, накроет с головой, станет важной частью вашей жизни.

Я очень жду ваших собственных проектов и надеюсь, что моя книга поможет вам их реализовать!

Встретимся в IT!

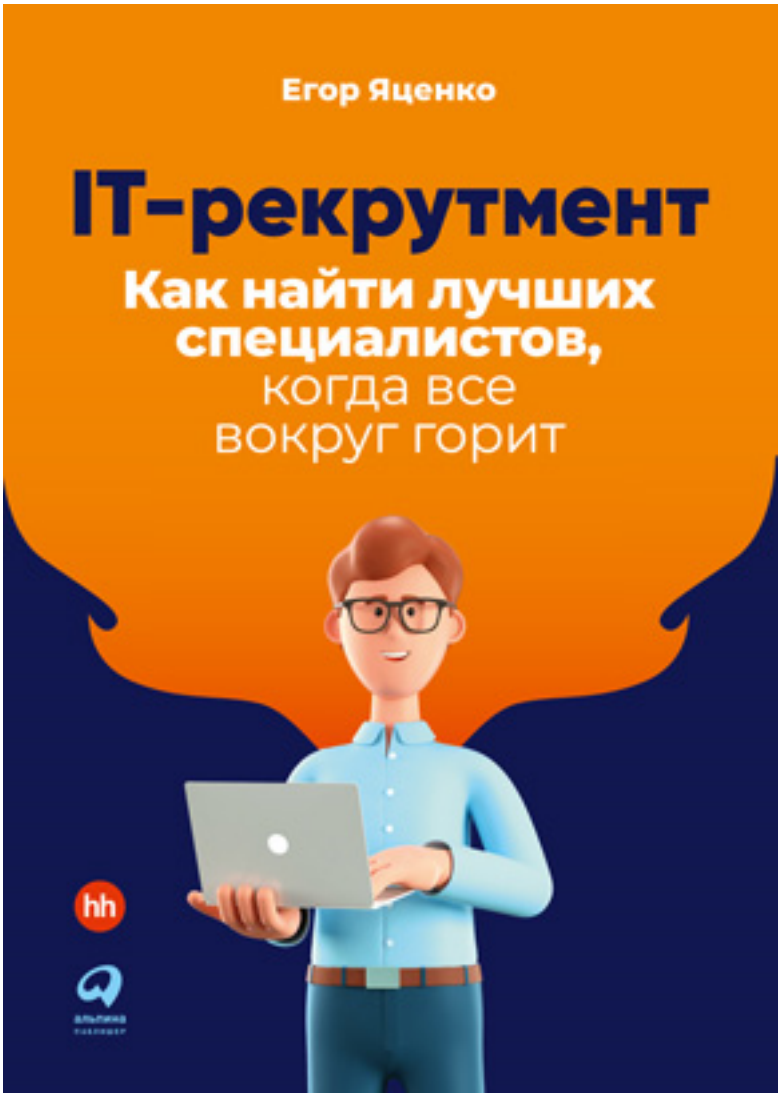
Рекомендуем книги по теме

Масштабированный скрам: Как организовать гибкую разработку в крупной компании

Крэг Ларман, Бас Водде

Мама, я тимлид! Практические советы по руководству IT-командой
Марина Перескокова

Этой кнопке нужен текст: О UX-писательстве коротко и понятно
Кирилл Егерев



IT-рекрутмент: Как найти лучших специалистов, когда все вокруг горит
Егор Яценко