от джуна до сеньора

как стать </>

вострebованным

разработчиком

Владимир Швец

Владимир Швец

От джуна до сеньора. Как стать востребованным разработчиком

Текст предоставлен правообладателем
http://www.litres.ru/pages/biblio_book/?art=68616021
От джуна до сеньора: Как стать востребованным разработчиком / Владимир Швец: Альпина Паблишер; Москва; 2023
ISBN 9785961484250

Аннотация

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.

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*

но.

Копирование, воспроизведение и иное использование электронной книги, ее частей, фрагментов и элементов, выходящее за пределы частного использования в личных (некоммерческих) целях, без согласия правообладателя является незаконным и влечет уголовную, административную и гражданскую ответственность.

* * *

Владимир Швец

ОТ ДЖУНА ДО СЕНЬОРА

Как стать востребованным разработчиком

альпина
ПАБЛИШЕР

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 произво-

дите. 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 кон-

сультантами – 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 из-

менением 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.

Конец ознакомительного фрагмента.