Что я понял на первой работе программистом / Мои советы Junior-разработчикам

**8 мин**

**38K**

[Python\*](https://habr.com/ru/hubs/python/)[Программирование\*](https://habr.com/ru/hubs/programming/)[Django\*](https://habr.com/ru/hubs/django/)[Карьера в IT-индустрии](https://habr.com/ru/hubs/career/)

Мнение

Всем привет! Меня зовут Максим. Я backend-разработчик, который не так давно устроился на свою первую работу. В этой статье хочу поделиться своими наблюдениями/советами/рекомендациями для начинающих программистов. Ведь, как известно, если опыт не превращён в текст, он даже не становится прошлым.

Материал будет полезен тем, кто ищет первую работу или не так давно её нашёл. Примеры будут из области Python Backend, но наблюдения универсальны и спокойно перекладываются на другую область. Поехали!

Погружайся в инструменты

Судя по тому, что я читаю в Интернете, начинающие программисты делают акцент на **изучение** языка, фреймворков, best practice и т.д., а инструменты отодвигают на второй план. Очевидно, что человек, будучи джуном, уже в курсе как настроить IDE, что такое Docker, Git и прочее. Однако, когда разрабатываешь что-то большее, чем собственные pet-проекты, следует копать глубже и предметно погружаться в вопрос.

Если раньше можно было спокойно использовать Dockerfile c образом alpine и было ОК, то теперь надо искать [эффективные и надёжные альтернативы](https://pythonspeed.com/articles/alpine-docker-python/) (например, python-slim-buster).

Создавать отельный файл docker-compose.yml для локальной разработки тоже необязательно, ведь существует docker-compose.override.yml!

Вряд ли в домашних проектах, кто-то кроме тебя добавлял изменения в main-ветку, пока работаешь в feature-ветке. А в коммерческой разработке это норма. Придётся резко узнать о существовании rebase. Подобных моментов куча.

Теперь ты пишешь код не один - значит пришло время полноценно научиться пользоваться pre-commit, black, isort, mypy и другими инструментами, использование которых принято в компании. Чтобы войти в курс дела рекомендую посмотреть ролик:

Также приведу другие ресурсы, которые взял на вооружение:

* **Git**
  1. [***Курс работы с Git и GitLab***](https://www.youtube.com/playlist?list=PLbf8m52BvqlFlblJqQKPuEU26pwgqe7zK)
  2. [**GIT - Полный Курс Git и GitHub Для Начинающих [4 ЧАСА]**](https://www.youtube.com/watch?v=O00FTZDxD0o)
  3. [**Официальная документация Git**](https://git-scm.com/book/en/v2)
* **Docker**
  1. [**Docker с 0 до 100%. Всё, что нужно знать**](https://www.youtube.com/watch?v=O8N1lvkIjig)
  2. Книги - [Docker на практике](https://www.labirint.ru/books/716923/), [Docker: Up & Running](https://www.oreilly.com/library/view/docker-up/9781492036722/), [Использование Docker](https://www.labirint.ru/books/566829/)
* **Nginx** - **[Nginx Mastery](https://www.youtube.com/playlist?list=PLOLrQ9Pn6cawvMA5JjhzoQrnKbYGYQqx1)**
* **Ansible** - книга [Бас Мейер, Лорин Хохштейн и Рене Мозер Запускаем Ansible. Простой способ автоматизации управления конфигурациями и развертыванием приложений.](https://www.labirint.ru/books/965627/)

Всё это не мишура, которая отвлекает от написания кода. Чем лучше знаешь инструменты, тем быстрее пишешь код и приносишь пользу бизнесу.

Коммуникация и soft skills

Я считаю, для джуна чрезвычайно важны soft skills. Минимальный базовый набор hard skills он получил, а софты дадут возможность быстро прокачиваться за счёт взаимодействия с более опытными коллегами. Очевидно, что обмен реальным опытом даёт мгновенный рост (это я почувствовал на себе уже в первую неделю работы).

Не надо бояться спрашивать. Как говорится - не стыдно не знать, стыдно не хотеть знать. Но спрашивать надо тоже правильно. Нет смысла доставать коллег банальными вопросами из разряда: “А как сделать сортировку в админке?”. Спрашивать нужно то, с чем действительно не можешь разобраться сам. Решения **нет** на форумах, YouTube, статьях в Интернете. Перепробовал кучу всего, результата нет, а время идёт. Старайся по максимуму закрывать вопросы самостоятельно, но если не получается - это не смертельно. Все с чего-то начинали.

Кроме того, важно уточнять у менеджера, если не до конца понял задачу. Лучше “показаться тупым” и уточнить, чем за день до сдачи проекта переделывать фичу, которая работает не так, потому что ты что-то не правильно понял.

Кроме того, не стоит бояться просить ревью у старших коллег. Команда также заинтересована в твоём росте. Ведь твоё профессиональное совершенствование напрямую связано с решением вопросов бизнеса.

*Если хочешь идти быстро иди один, если хочешь идти далеко иди с кем‑то*. Это характеризует и современную разработку. Разработчики люди, а люди — существа социальные. И с этим ничего не сделаешь. [[**Senior Softskills Engineer]**](https://habr.com/ru/articles/738450/)

Не изобретай то, что уже сделали в команде раньше

Когда приходишь в компанию, скорее всего, у неё уже есть наработки. Например, при запуске нового проекта не пытайся сочинить структуру самостоятельно, основываясь на своём, напомню, “игрушечном опыте”. Скорее всего в команде есть boilerplate-шаблон, которого следует придерживаться. Узнай про это заранее (опа, soft skills in action).

Это же касается code style и других guidelines. Тот же Black Formatter настаивается по-разному: одинарные или двойные кавычки, длина строки 79 или 88 символов и т.д. Это уже давно продумано и согласовано, твоя задача узнать “правила игры” и придерживаться их. Круто, если ты можешь привнести что-то новое. Например, если нет общего стандарта документирования кода попробуй его продумать и предложить для внедрения.

Узнай как конкретно у вас принято именовать коммиты/ветки при работе с системой контроля версий. Мы в команде используем **[Conventional Commits 1.0.0](https://www.conventionalcommits.org/en/v1.0.0/).** Однако не факт, что у вас будет также.

Аналогично с моментами из кодовой базы. Например, нет смысла изобретать собственную библиотеку для SEO, если её уже сделали раньше. Не факт, что сделаешь лучше или она будет корректно работать, а вот время определённо потратишь. Возьми готовое решение и переиспользуй для своего проекта. Если в последствии доделаешь или расширишь – будет круто. У компании появится внутренний инструмент (а может и open source), который можно тянуть от проекта к проекту. Однако стоит придерживаться одного правила:

*"Привносить что-то новое можно и нужно, но только если это не вредит выполнению текущих задач" (с) Егор Романов, Frontend-разработчик в ProninTeam.*

Стоит быть аккуратным со сторонними библиотеками. Обязательно посмотри поддерживают ли её авторы до сих пор, закрываются ли issues, пользуются ли ей другие люди. Возвращаясь к вопросу soft skills, спроси у коллег, они подскажут проверенный вариант, если сам не уверен.

"*Мне только спросить*" - ваша коронная фраза на ближайшее время.

Делай рефакторинг, если позволяют сроки

Безусловно надо стремиться к тому, чтобы сразу писать качественный чистый код. Однако это не всегда получается (особенно если речь идёт о джунах). Поэтому имеет смысл посмотреть код проекта и подумать, что и как можно улучшить. Например, подсмотреть более эффективный алгоритм, найти хорошую библиотеку или внедрить паттерны проектирования. Так можно убить сразу двух зайцев - улучшить качество кодовой базы проекта и прокачаться как разработчику. Потом можно ревью провести и ещё улучшить, но..

*"Коммерческая разработка - это про поиск рабочих решений за оптимальное время" (с) Александр Кондратьев, Backend-разработчик в ProninTeam.*

"Причёсывать" код необходимо, если тебе позволяют сроки. Да, паттерны это круто. В дальнейшем будет проще расширять код, поддерживать и т.д. Но если ты из-за неопытности долго провозился с внедрением очередной Стратегии и не успел доделать к дедлайну - бизнес спасибо не скажет. Программирование - это не про абстрактные фабрики, микросервисы и Docker. Программирование - это про решение задач бизнеса. Бизнесу нужно, чтобы приложение решало его проблемы (оформляло заказы, присылало уведомления, обрабатывало данные). И если код решает эти проблемы **эффективно** через набор if’ов, а не pattern matching - бизнес не пострадает.

В идеале планировать процесс разработки, чтобы оставалось хотя бы какое-то время на улучшение. Если удалось выкроить время на рефакторинг, имеет смысл сделать код лучше - внедрить паттерны и приемы. Так ты приобретаешь опыт и в следующий раз, скорее всего, будешь писать код сразу применяя наработки. Значит дополнительное время уже не нужно или пойдёт на усовершенствование других моментов.

Рост - процесс итеративный. Нельзя просто так взять и стать middl’ом. К этому надо прийти, маневрируя между сроками, тасками и саморазвитием.

**База про паттерны и DDD**

Очень доступно про паттерны (примеры на С++): <http://cpp-reference.ru/patterns/creational-patterns/factory-method/>

Паттерны с примера на Python: <https://github.com/pkolt/design_patterns>

Про DDD c примерами на Python:  
<https://habr.com/ru/articles/559560/>  
<https://habr.com/ru/companies/oleg-bunin/articles/488010/>

Щупай смежные области

Изучать разное - это залог профессионального роста и возможность не пресытиться одноплановой работой.

Например, если ты backend’ер можно на базовом уровне разобраться в HTML/CSS/JS. Верстка - смежная область, а значит будет легче понимать, что вообще происходит и станет проще коммуницировать с frontend’ерами в команде (а это ускоряет процесс разработки).

Если frontend’ер, то почему бы не научиться писать Dockerfile для Nest, чтобы самому было проще работать локально.

Работа становится эффективнее, задачи бизнеса закрываются, а ты и твоя ценность, как специалиста, растёт.

Лично я в процессе работы стал робко посматривать на серверные моменты (настройка Nginx, Caddy) и CI/CD-пайплайны.

Сделай план развития

В процессе работы обнаружишь пробелы в знаниях. Составь список того, что стоит изучить/прокачать. В идеале разделить этот список на две части:

* пробелы, которые нужно закрыть в кратчайшие сроки (то, что будет нужно в работе уже в ближайшее время)
* долгосрочная перспектива (то, что в идеале нужно освоить, но в ближайшую неделю-месяц, скорее всего не понадобится).

Лично у меня получился такой список:

* **Краткосрочные**
  1. Админ-панель Django (кастомные виджеты, list\_select\_related, classes, django-nested-admin и т.д.)
  2. Миграции
  3. Permissions DRF
* **Долгосрочные**
  1. Nginx/Caddy
  2. Docker (на продвинутом уровне)
  3. Git (на продвинутом уровне)
  4. Ansible

Развиваться желательно не в рабочее время (см. ущерб текущим задачам). Лично я предпочитаю прокачиваться размеренно в выходные или перед работой. Например, если рабочий день начинается в 8:00 - не грех встать в 7:00 и поучиться. Развитие начинающего - дело рук самого начинающего.

Не пытайся прыгнуть выше головы

Ты теперь часть команды, часть корабля. Не нужно бежать вперёд остальных и пытаться закрыть проект за два дня без еды, сна и отдыха. Разработка - это не спринт, а марафон. Да, на энтузиазме можно один раз сделать всё и сразу. Ну, два раза, три… Но в определённый момент либо не выдержит здоровье, либо наступит то самое хайповое выгорание. От того, что ты сделал проект за три дня вместо пяти запланированных компании лучше не станет. А вот потеря эффективности и трудоспособности точно негативно отразится на процессе разработки в дальнейшем.

В компании есть менеджеры - они выстраивают процесс работы, коммуникацию с заказчиком, пытаются равномерно и по способностям распределить задачи. Ты в руках профессионалов, доверься им. От тебя требуется только качественно и в срок выполнять требуемые задачи.

Это не значит, что не стоит быть активным и нужно пассивно сидеть. Я лично старюсь предлагать какие-то решения и придерживаться принципа давай чуть больше, чем от тебя требуется. Но под “чуть больше” не подразумевается работа в надрыв и ущерб самому себе.

Канбан-доски, кстати, очень помогают организовать работу. Не игнорируй их, если они есть на проекте. Если их нет, веди сам. Например, зарегистрируйся в YouGile и записывай всё в “Приватных задачах”. Также можешь туда складывать персональный бэклог (то, что было бы неплохо сделать, подпилить или доработать).

Персональный бэклог

Отдыхай

Последний в списке, но не последний по важности, пункт - это отдых.

Разработка - это интенсивная интеллектуальная сфера труда. Чтобы ментальная и физическая составляющие не пострадали следует **не пренебрегать отдыхом, едой и сном.**

Важно делать перерывы и в процессе работы. За это тебе скажет спасибо и продуктивность, и здоровье (спина, шея).

Напоминаю, компании не нужно чтобы ты умер или заболел. Ей нужно чтобы ты качественно делал свою работу, желательно на всём жизненном цикле компании (а не один месяц).

Когда ты сидишь над задачей 10 часов и она не получается, зачастую проблема не в том, что задача сложна или ты тупой, а в том, что нужно просто остудить мозг…сделать перерыв хотя бы 15 минут…(с) Егор Романов, Frontend-разработчик в ProninTeam в [интервью](https://www.youtube.com/watch?v=zD6AyVfT3Gc) каналу PyLounge.

Бытовой отдых нужен обязательно, даже если тебя мучает совесть.

Можно отдыхать каждый день понемногу или полностью разгружать выходные. Лично мне не нравится “тюленить” полные выходные, я предпочитаю отдыхать равномерно в течение недели, например, не делать ничего *«полезного»* после 20:00. [[**Как всё успеть? | Мой тайм-менеджмент**](https://habr.com/ru/articles/709462/)]

В идеале отдых должен быть в отрыве от компьютера, телевизора, телефона. Лично мне помогает разгрузиться поход в спортзал или просто прогулка по улице.

Заключение

Ко всем этим мыслям я пришёл опытным путём и надеюсь, что эти рекомендации облегчат процесс интеграции при устройстве на первую работу и помогут вам достичь успеха на нелёгком пути Junior-разработчика. Если есть что добавить, милости прошу в комментарии. Читателям и мне будет интересно узнать альтернативное или дополняющее мнение.