# Newbies course content

## Вводный модуль. Описание курса, зачем и кому он нужен.

* описание курса
* назначение курса
* краткое содержание/оглавление курса
* ответы на вопросы

## Модуль №1. Что означает термин «профессионал»?

Основное правило профессионала – слова не должна расходиться с делом. Отсюда важное следствие: если что-то сказал, то за свои слова/обещания надо уметь отвечать (делать в обещанные сроки, делать обещанные «фичи»).

Кого же можно назвать профессионалом? Является ли хороший специалист профессионалом? А наоборот – профессионал – хорошим специалистом? Вот что обычно вкладывается в термин «профессионал» (какие качества):

1. Ответственность. Принципы ответственного поведения:

* не навреди функциональности (все что работало до тебя – должно работать и далее)
* контроль качества должен показать, что качество на должном уровне (прохождение всех тестов)
* уверенность в том, что ваш код работает (тесты)
* не навреди структуре (ухудшение структурированности проекта)

1. Профессиональная этика. Разбираться в как минимум в следующих темах:

* Шаблоны проектирования (Gang of Four)
* Принципы проектирования (SOLID, компонентное проектирование)
* Методологии разработки (поговорим далее)
* Владение инструментарием проектирования (UML, структурные диаграммы, диаграммы состояний/переходов, блок-схемы)

1. Непрерывное обучение и тренировки (далее поговорим о важности)
2. Наставничество (необходимо побывать с разных сторон – быть наставляемым, быть наставником)
3. Знание предметной области и понимание интересов работодателя/заказчика (раскроем тему чуть позже)

## Модуль №2. Общие советы начинающему профессионалу.

Большинство приведенных советов помогут тебе встать на путь профессионального становления. К сожалению, этим правилам редко учат в ВУЗах или на профессиональных курсах.

### Совет №1. Освой сферу бизнеса (предметную область), в которой работаешь. Разберись, как работает бизнес в целом.

Следует разрушить границу между бизнесом и информационными технологиями, смотреть на технологии глазами бизнеса компании. Изучи основные принципы бизнеса, с которым связана твоя деятельность.

Разберись в основах финансирования бизнеса (откуда в твоем бизнесе берутся деньги?), как ты участвуешь в бизнесе своей компании, в бизнесе Заказчика. Ты должен понимать, как твоя работа помогает зарабатывать или экономить деньги, как Заказчику, так и твоей компании. Ты не сможешь помогать бизнесу творчески, если не знаешь, как он устроен.

### Совет №2. Помни, на кого ты работаешь.

У нас есть прекрасный̆ постулат о том, что мы должны согласовывать свою деятельность с целями компании, чтобы гарантированно влиять на конечный̆ результат работы и все такое. Но по правде говоря, многие из нас со своего уровня просто не видят, как этого добиться. За деревьями леса не видать. В хорошо структурированной̆ среде цели руководства являются целями группы. Реши проблему своего начальника, и ты решишь проблему группы. Кроме того, если ваш начальник придерживается аналогичного подхода, то проблемы, которые ты решаешь для него, на самом деле являются проблемами уже его начальства. И так далее, вплоть до самых верхов фирмы — генерального директора, акционеров или даже заказчиков.

Делая свою небольшую часть работы, ты вносишь вклад в достижение целей̆ фирмы. Это позволяет почувствовать свое предназначение. И придает твоей̆ работе смысл. Успех твоего начальника — это и твой успех тоже. Если работа твоего начальника — знать и расставлять приоритеты, а не лично выполнять все задания, то твоя работа состоит именно в выполнении всех заданий. Ты делаешь не работу начальника, а свою работу.

### Совет №3. Будь специалистом.

Выбери свою специализацию и планомерно углубляй свои познания/навыки в ней. Невозможно быть "всеведущим", поэтому выбери одну технологию/язык разработки и сконцентрируй свое внимание и усилия на нем.

### Совет №4. Будь универсалом.

Универсалы встречаются редко, поэтому ценятся высоко. Ваши навыки не должны ограничиваться одной технологической платформой. Необходимо регулярно выделять время на изучение чего-то нового.

Казалось бы, что это противоречит предыдущему пункту, но нет - речь идет о том, что необходимо иметь широкий кругозор в технологиях и знать инструментарий - чем о бОльшем числе инструментов ты знаешь, тем лучше ты будешь понимать какой из них более применим в конкретной ситуации, какой следует изучить глубже, чем знаешь сейчас. Как и в предыдущем пункте - постоянно практикуйся/инвестируй. Также, аналогия со спортом.

### Совет №5. Постоянно практикуйся, инвестируй время в свое ремесло.

Тренируйся на пределе своих возможностей. Аналогия со спортом. Code Kata упражнения.

### Совет №6. Будь "самым худшим специалистом" в комнате. Найди наставника.

Всегда ищи у кого и чему ты можешь научиться. В зависимости от другого человека нет ничего страшного. Главное - убедиться, что это правильный человек. Без примера для подражания сложно найти стимул к совершенствованию. Наставник поможет выбрать верные направления для обучения, из огромного множества вариантов. Еще наставник служит доверенным лицом, которое наблюдает и оценивает принимаемые тобой решения и твои успехи.

В идеале было бы здорово найти человека, который будет активно тебя направлять, но это далеко не всегда реализуемо на практике. Ты можешь стать наставником самому себе :)

### Совет №7. Помогай учиться коллегам. Стань наставником.

Всегда делись полученными знаниями, помни - обучение один из самых лучших способов самому лучше разобраться в предмете. Чтобы понять, на самом ли деле ты хорошо разбираешься в теме, попробуй объяснить эту тему кому-то еще.

Не следует также недооценивать тот факт, что помогать людям *приятно*. Если мысленно вы можете завладеть вниманием публики, имеет смысл воспользоваться этой способностью в альтруистических целях. В неопределенности современной экономической ситуации реальная *помощь* другим — это работа, с которой тебя не могут уво­лить. И платят за нее валютой, которая не обесценивается с инфляцией. Как правило - наставники не попадают под сокращение :)

### Совет №8. Полюби или меняй.

Если ты хочешь достичь высот в своей работе, ты должен быть влюблен в нее. Если же тебе все равно, это непременно станет заметно. Работай потому, что не можешь не работать. Найди работу/задачи/проект/область деятельности, которые тебе по-настоящему интересны.

Mood monitoring. Проведи следующий эксперимент: начиная со следующего понедельника на протяжении двух недель веди журнал. Каждый будний день, проснувшись, отмечай уровень энтузиазма по десятибалльной шкале: 1 означает, что ты предпочел бы заболеть, лишь бы не идти на работу, а 10 — что высокая температура не может удержать тебя в постели, так тебя захватила идея по поводу решения очередной задачи. Через две недели внимательно посмотри на результаты. Есть ли на графике пики? Какие тенденции он показывает? Он прижат вниз или стремится ввысь? Какую бы ты поставил себе оценку, если бы это был школьный тест? На протяжении следующих двух недель каждое утро думай над тем, каким образом ты собираешься сегодня работать на 10. Планируй сегодняшнее рабочее расписание так, чтобы на следующий день ты не мог дождаться момента, когда приступишь к работе. Каждый день фиксируй вчерашний уровень энтузиазма.

Если после двух недель ничего коренным образом не поменяется, имеет смысл задуматься о переменах – поменять подход к работе или сменить задачи/проект/область деятельности. Ты не представляешь, насколько огромен наш маленький IT мир ☺ Спроси совета у коллег или у своего наставника, побеседуй с руководителем, спроси людей чье мнение ты ценишь и уважаешь.

### Совет №9. Учись ловить рыбу. Будь проактивным. Будь катализатором изменений.

Не жди, пока тебе расскажут, спрашивай сам! Анализируй чужой код из открытых источников, код приложений и библиотек, которые используешь в своей работе. Чужой код не только позволяет найти ответы на конкретные вопросы, но и служит своего рода увеличительным стеклом для рассмотрения собственного стиля и способностей. Доступное количество таких программ столь велико, что вряд ли их все можно прочитать. Разумеется, попадаются среди этого изобилия и плохие проекты, тем не менее нам доступно и довольно много *великолепных* примеров. Существует открытый код, реализующий практически любую решаемую программно задачу и почти на всех доступных языках программирования. Критически анализируя этот код, ты постепенно начнешь вырабатывать собственный вкус, как это бывает в музыке, живописи и литературе. Различные стили и приемы тебя позабавят, удивят, рассердят и (надеюсь) *стимулируют* к работе. Ищущий найдет тут всё — от трюков, повышающих продуктивность работы, до парадигм проектирования, полностью меняющих подход к целому классу проблем. Как и в искусстве, изучая особенности чужих работ и учась на них, ты выработаешь свой ни на кого не похожий стиль разработки ПО. Изучай паттерны и анти-паттерны разработки ПО.

Проактивность помогает и в работе с заказчиком – смотри **Модуль №X**.

### Совет №10. Подход к работе.

Разработка ПО - это не некая вещь, а процесс создания некой вещи. При написании кода важно сосредоточиться не только на разрабатываемом продукте, но и на самом процессе разработки. Отвлекаясь от процесса, мы рискуем опоздать со сроками выполнения задания, получить некачественный продукт или не получить вообще ничего. Ни один из этих вариантов заказчика не обрадует. К счастью, процесс создания хорошего программного обеспечения (и в целом продукции) давно как следует обдуман. И изрядная часть этой информации легла в основу целой группы методик.

Как программист, тестировщик или дизайнер ПО ты можешь считать, что сам по себе процесс разработки не входит в сферу твоей ответственности. Возможно, это правильно, так как ты всего лишь наемный работник. Но, к сожалению, обычно эта ответственность повисает в воздухе. И даже если ее на кого-то возлагают, она передается «группе организации процесса» или другому отдельному подразделению. Но дело в том, что для успешного внедрения метода разработки программного обеспечения его должны принять те, кто будет им пользоваться, — такие, как ты.

Найти человека, который сможет писать программы, куда проще, чем человека, умеющего организовать процесс написания программ. Поэтому имеющиеся в твоем арсенале сведения о принципах организации процесса разработки ПО будут тебе только в плюс.

**Более подробно практики и методологии разработки рассмотрим в Модуле №X и последующих.**

**Совет №11. Автоматизируй свою работу и повседневные задачи.**

Выбери повторяющуюся задачу, с которой̆ тебе часто приходится сталкиваться, и напиши для нее генератор кода. Начни с простого. О возможности многократного использования пока можно не беспокоиться. Просто сделай̆ так, чтобы генератор экономил твое время. Подумай̆, как поднять уровень абстракции генерируемого кода.

**Более подробно автоматизацию повседневных задач и всего процесса разработки рассмотрим в Модуле № Х.**

### Совет №12. Камешек в ведре воды.

Что произойдёт, если ты встанешь и выйдешь из офиса, чтобы никогда туда не возвращаться? Я знаю многих программистов, которые утешаются, представляя подобную сцену. Ты просто встаешь, идешь в кабинет начальника и кладешь ему на стол заявление об уходе. Они еще поймут, какого ценного работника потеряли! Подобные мечты неплохо помогают пережить по-настоящему неудачные дни, но предаваться им постоянно — не самая лучшая позиция.

Начнем с того, что на самом деле это неправда. Люди увольняются с работы каждый̆ день. Многих увольняют. Многие увольняются сами. Некоторые даже реализуют твою мечту и демонстративно уходят без уведомления. Но покидаемые ими фирмы крайне редко ощущают последствия их ухода. В большинстве случаев, даже когда речь идет о ключевых позициях, эффект оказывается на удивление слабым. Для фирмы твое присутствие на работе подобно камешку в ведре воды.

Подумай̆ вот о чем: если завтра ты уволишься, это произведет (в среднем) примерно такой же эффект, как увольнение любого из твоих коллег. Чем более незаменимым ты себя считаешь, тем менее незаменимым ты становишься (и тем меньше людей̆ хотят с тобой̆ работать). Чувство незаменимости является плохим симптомом, особенно у разработчика программного обеспечения. Заменить нельзя только того, кто справляется со своей̆ работой̆ особым, недоступным другим способом. Хотя мы все хотели бы претендовать на гениальность, крайне немногие разработчики настолько уникальны, что их и в самом деле нельзя заменить. Я слышал, как программисты полушутя говорили, что создать «гарантированное рабочее место» можно, просто написав трудный̆ в сопровождении код. Мне доводилось даже встречать тех, кто предпринимал подобные попытки. И каждый̆ раз эти люди становились мишенями для других.

### Совет №13. Умей говорить «нет».

Самый̆ быстрый̆ путь к невыполнению обязательств — непосильные обязательства. Несмотря на очевидность этого утверждения, мы берем их на себя каждый̆ день. Попав в затруднительное положение, мы не хотим разочаровывать начальство и соглашаемся сделать нереальную работу за нереально короткий̆ срок.

Говоря «да», чтобы избежать разочарования, мы попросту врем. Однако это не значит, что если не можешь сказать «да», надо сказать «нет». Предложи альтернативное решение проблемы, которое может быть компромиссом или план к достижению компромисса. «Нет» сказанное напрямую и безальтернативно – не самый лучший способ для коммуникации.

### Совет №14. Умей говорить «да».

Говорить «да» тоже требует умения и искусства.

Сказать. Ответственно отнестись. Сделать. Обещание состоит из трех частей̆.

1. Вы говорите, что вы это сделаете.
2. Вы ответственно относитесь к своим словам.
3. Вы выполняете обещанное.

Тщательно выбирайте формулировки, которые вы используете в своих обещаниях, потому что по словам часто можно судить о дальнейшем ходе событий. Если вам не удается подобрать нужные слова, скорее всего, вы недостаточно ответственно относитесь к сказанному или не верите в его выполнимость. Несколько примеров слов и выражений, которые являются характерными признаками пустых обещаний.

* **«Нужно/должен»**: «Нам нужно это сделать поскорее», «Мне бы нужно сбросить лишние килограммы», «Кто-то должен об этом позаботиться».
* **«Надеюсь/хорошо бы»**: «Надеюсь, это будет сделано к завтрашнему дню», «Надеюсь, мы еще встретимся и поговорим на эту тему», «Хорошо бы выкроить время для этого», «Хорошо бы, чтобы компьютер работал побыстрее».
* **«Давайте»**: «Давайте потом встретимся», «Давайте доделаем эту штуку».

Если вы начнете обращать внимание на эти слова, то увидите, как часто они звучат вокруг вас — и даже в том, что вы говорите другим. Вы увидите, как мы стараемся избежать ответственности за сказанное. И это недопустимо, если вы или кто-то другой̆ полагается на эти обещания в своей̆ работе. Впрочем, первый̆ шаг уже сделан — вы начинаете узнавать признаки безответственного отношения к обещаниям в окружающих и в себе.

Серьезное обещание утверждает факт того, что ВЫ что-то сделаете, с указанием четкого момента завершения.

### Совет №15. Учись работать в коллективе.

Это означает, что вы выполняете свои функции настолько хорошо, насколько возможно, и помогаете своим коллегам, если они окажутся в беде. Человек, умеющий̆ работать в коллективе, часто общается с другими, обращает внимание на своих коллег и добросовестно исполняет свои обязанности.

* Не надо пытаться
* Не надо проявлять пассивную агрессивность

### Совет №16. Привыкай к странностям заказчика.

Очень часто тебе будут встречаться моменты, которые будут вызывать недоумение, злость, непонимание и похожие чувства. Просто прими это – большинство таких вещей ты изменить не в силах и придется с ними/в таких условиях работать:

* Заказчик меньше беспокоится о проекте, чем вы сами
* Заказчик не владеет всей информацией
* Заказчик может прийти с изменениями в последний момент
* Что-то другое ☺

### Совет №17. Совещания – важная часть работы.

Привыкай к тому, что частью твоей работы станут совещания – не простые обсуждения техники внутри команды, а совещания с заказчиком, многие из которых покажутся тебе тратой времени.

О важности совещаний…

## Модуль №3. Обзор методологий и подходов к разработке ПО.

### Waterfall

* обзор процесса
* применимость в разных ситуациях
* плюсы и минусы

### Agile

* agile manifest
* обзор процесса
* применимость в разных ситуациях
* плюсы и минусы

### Scrum

* обзор процесса
* применимость в разных ситуациях
* плюсы и минусы

## Модуль №4. Agile (описание, принципы, техники).

* TDD
* BDD
* XP
* Pair programming
* Agile in real world

## Модуль №5. Scrum (процессы, события, артефакты).

* Scrum 3-3-5 model (3 roles, 3 artifacts, 5 events)
* Scrum interactions: PO-SM-Dev Team
* Scrum implementation in real world

## Модуль №6. Do's and don'ts в корпоративной среде. Коммуникация с заказчиком.

* **Do!** Правило «60 минут». Если столкнулся с проблемой и ее не получается решить в течение 60 минут – необходимо сообщать команде о проблеме (райзить ее).
* **Do!** Этот совет тесно связан с предыдущим. Если видишь и озвучиваешь проблему, всегда предлагай варианты решения и свои соображения. Просто указать на проблему недостаточно. Если не знаешь никаких вариантов – представь команде то, что уже «накопал» по данной проблеме, это существенно поможет команде.
* **Do!** Необходимо всегда пользоваться системой трекинга задач/багов. Если выполняется любая работа – она должна быть зафиксирована в этой системе. Нет записи – нет задачи!
* **Do!** Всегда имей все записи митингов («минутки», MoM) и переговоров, рассылай их после совещания заказчику (конечно, согласовав внутри команды). Также необходимо протоколировать все технические решения (да, придется писать документацию ☺), для этого подходит любая онлайн интерактивная система типа вики. Незаписанные решения – по факту не существуют (существуют только в вашей голове).
* **Do!** Всегда необходимо помнить, что вы говорите от имени всей команды/компании. Обдумывайте что и как вы скажете заказчику, какой имидж своей команды и компании вы создаете и поддерживаете.
* **Do!** Мы уже об этом говорили – но повторить не повредит ☺. В общении с заказчиком будь проактивен, выясняй подробности и детали задачи, требования, бизнес-кейс, который автоматизирует/решает задача и т.п. Вряд ли ты придешь в проект где все ясно и очень детально расписано, многие вещи придется выяснять самому. Часто в проекте есть аналитик, который существенно поможет с этим, но многие проекты обходятся без него – эту работу придется делать команде и тебе тоже.
* **Do!** Всегда выясняй какие процессы использует заказчик и следуй им. Если тебе они не нравятся или ты хочешь что-то изменить, для начала узнай/изучи уже существующие процессы и лишь затем предлагай обоснованные изменения/улучшения. Критиковать сходу то, что не использовал или не понимаешь – по крайней мере глупо.
* **Do!** Не бойся ошибиться или показать, что ты чего-то не знаешь. Помни – не ошибается только тот, кто ничего не делает. Команда всегда тебя поддержит и поможет вырасти и научиться. Если это не так – возможно это уже проблема, требующая решения?
* **Do!** Помни, не существует окончательных решений! То, что считалось «запечатленным в камне», может легко стать «песочным замком» по требованию бизнеса.
* **Don't!** Никогда не замалчивай проблемы, это ведет к наслоению проблем и к печальным последствиям для всей команды.
* **Don’t!** Не увлекайся, знай меру во всем. Улучшать техническое решение или рефакторить код можно до бесконечности, всегда необходим разумный компромисс между качеством и требованиями к срокам и ресурсам. Всегда помни об этом.
* **Don’t!** Никогда не разделяй в проекте на «мое» и «не мое». Весь код принадлежит команде, нет «чьих-то» частей в коде. Также, как и нет «моих» и «не моих» задач – команда вместе и сообща добивается успеха. Если разработчику необходимо написать тест или протестировать что-то – это надо сделать, и наоборот. Все задачи команда делает сообща!
* **Don’t!** Никогда не давай необдуманных ответов без обсуждения с командой. Всегда лучше (и профессиональнее!) взять небольшой таймаут и вернуться к заказчику с обдуманным ответом.
* **Don’t!** Никогда напрямую не критикуй решения/инфраструктуру/продукт заказчика или его контрагентов. Для начала – это невежливо ☺ Всегда лучше сначала выяснить все аспекты и историю того, что вам не нравится – продукта/системы/процесса. Поспрашивай у пользователей, нравится ли им это решение. И предложи свой продуманный вариант, будь готов ответить на вопросы о стоимости (ресурсы), времени реализации, переходе на твое решение и т.п. Все вышесказанное также относится к инструментам разработки, используемым фреймворкам и библиотекам.
* **Don’t!** Don't guess – just ask! Правило простое, но многие им пренебрегают. Ложные предположения всегда (!) ведут к неверным техническим решениям. Поэтому всегда спрашивайте, если что-то непонятно.

## Модуль №7. Методы оценки задач.

Прежде всего – что такое оценка? Можно найти много определений:

* приблизительный прогноз или вычисление
* предварительный расчет ресурсов, необходимых для реализации какой-либо функции / исправления ошибки / реализации проекта в целом
* суждение, основанное на впечатлениях или личное мнение

Все эти определения в целом верны, но в конечном итоге в реальной жизни оценка всегда увязывается с деловыми целями, обязательствами и контролем.

### Основные принципы

1. **Не путай оценки, цели и обязательства.** Цель – это формулировка деловой задачи, обязательство – это простое обещание предоставить некоторую функциональность на согласованном уровне качества (которое как мы помним надо еще и выполнить).

Пример цели: версия 2.5 должна быть готова к продакшену через 2 недели.

Пример обязательства: мы обещали заказчику закончить все в этом месяце.

1. **Если тебя просят оценить что-либо, важно понять или определить (или просто спросить), что именно нужно спрашивающему – оценка или план достижения цели.**

Пример: первая версия для пресейла должна быть готова через 1 месяц.

Это пример того, что необходима не оценка («это займет минимум 2 месяца!»), а именно план – как достигнуть поставленной цели («чтобы сделать это за 1 месяц надо пожертвовать функциями 2, 5, 8 и снизить % покрытия кода тестами»).

1. **Если ты столкнулся с точечной оценкой, постарайся понять, действительно ли это оценка или в реальности это просто цель.**

прпкар

1. Ваоао
2. Урв
3. Вор
4. Ваоао
5. Ао
6. Вао

Краткий обзор простых и эффективных методов оценки задач.

* Оценки и обязательства
* PERT
* Дельфийский метод/аффинная оценка
* Быстрое голосование (Fist of five)
* Planning poker
* Закон больших чисел

## Модуль №8. Обзор CI/CD практик.

TBD – наверное необходим дев опс.

## Вопросы/ответы

Сессия вопросов-ответов от слушателей, если таковые будут.

## Использованная/рекомендуемая литература

1. Чед Фаулер. Программист-фанатик.
2. Майкл Нейгард. Release It!
3. Эндрю Хант, Дэвид Томас. Программист-прагматик.
4. Роберт Мартин. Идеальный программист.
5. Том ДеМарко – все книги.
6. Тимоти Листер – все книги.
7. Кент Бек – все книги.
8. Мартин Фаулер – все книги.
9. Джоэл Спольски – все книги, блог.