?????????????? ??????

? ??? ???

Здесь будут списки умений и тренировочных задач, необходимых для участия в проекте в определенном качестве.

Для участия в определенном качестве требуются умения в некоторых направлениях на начальном уровне ("Где инструкция"), в некоторых на уровне осознания ("Эксперт"). Частая и критичная ошибка в том, что ненужные направления пытаются изучить на уровне осознания, из-за чего на требуемые направления не остается времени. К тому же, отсутствие понимания своей задачи в рамках общего процесса и целей каждого этапа приводит к фрагментарным несвязанным и быстро испаряющимся знаниям.

Некоторые умения естественно будут пересекаться между направлениями. Они будут вынесены в раздел "Общие умения" в рамках главы.

В связи с серьезным развитием технологий, обычно существует несколько (десятков) вариантов решения задачи. Сюда включаются и правила оформления, сохранения, ... Однако в рамках проекта должен выбираться один вариант. Он может быть неоптимальным, длинным, но так было определено. Изменение варианта должно обновлять весь проект, иначе в перспективе будет бардак. Поэтому обычно указывается предпочтительный вариант для конкретного проекта. Если он отсутствует - его нужно определить)

Алгоритм изучения

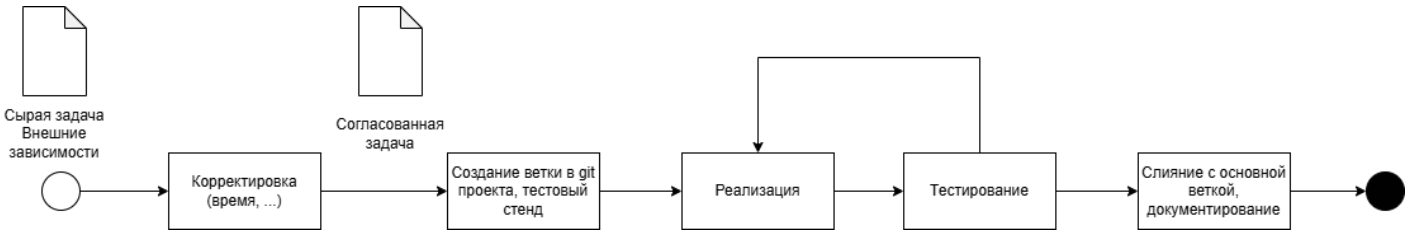
- Сделать тестовый стенд
- Разобраться с бизнес процессами, сделать по инструкции периферийные задачи и запомнить, как искать инструкции
- Убедиться (вспомнить) в наличии необходимых экспертных знаний направления
- В зависимости от желания/необходимости начать изучать в рамках выбранного бизнес процесса непосредственно связанные задачи на экспертном уровне. Цель в том, чтобы в как можно большем количестве бизнес процессов этапы были осознаны. Желательно связанное понимание, то есть два последовательных этапа лучше трех разрозненных.

??? ???????????????

Для разработчиков

????? ??????

Процесс:



Система управления задачами: страница "Текущий статус проекта"

Используемые среды разработки, настройка

Инструкция по работе с git

Инструкция создания тестового стенда

Правила реализации

Правила тестирования

Правила документирования

Правила слияния веток

Структура задач

Направление			Ссылки на описания
Коммерческое			
	Маркетинг		
	Финансы		
	PR		
ИТ			
	ИТ менеджмент		
		Взаимодействие с клиентами	
		Обучение	

Направление			Ссылки на описания
		Глобальные требования	
		Бизнес процессы	
		Требования к ПО	
		Дополнительные сервисы	
	Архитектура		
		Блоки	
		Правила взаимодействия (API)	
	Инфраструктура		
		Требования	
		Правила тестирования	
		Правила мониторинга	
	Фронтенд		
		Дизайн	
		Инструменты	
		Технологии	
	Бэкенд		
		Инструменты	
		Технологии	

??? ???????

Как учиться

???????? ????????

Правила чтения книг

Перед прочтением главы делать быстрый просмотр, концентрируясь на картинках / выделении шрифтом / ... Пара минут.

Записывать вопросы, главные идеи и выводы.

Сосредоточенность и расслабление

Нужно терять концентрацию для очищения. Отключаться от работы полезно во время интеллектуального труда.

Мозг работает в сфокусированном или расслабленном режиме. Оба режима важны для обучения. Для разных задач используются разные участки мозга. Перед стартом обучения нужно сначала полностью сосредоточиться на этом,
чтобы “включить” эти участки мозга и запустить процесс обучения. Расслабленный режим создает связи между информацией. Для эффективного обучения необходимо чередовать режимы. В каждый момент времени можно находиться в одном режиме.

Переход между режимами возможен при прогулке / душе / сне / фокусировке на другой задаче.

Застревание: два варианта - непонимание начальных условий и отсутствие перехода в расслабленный режим. Пауза должна быть 5-10 минут.

Прокрастинация

При напоминании о деле, которое делать неохота, начинает светиться часть мозга, ответственная за боль. Но примерно через 20 минут после начала выполнения этой задачи все прекращается. Техника Pomodoro:

- отключить все отвлекающие элементы (телефон, музыка, коллеги, ...)
- установить таймер на 25 минут
- Максимально сфокусироваться на задаче
- Через 25 минут наградить себя (посмотреть видео / послушать музыку / ... 5-10 минут

Фокусировка именно на награде после завершения периода. Не на факте выполнения всей работы. Затем можно увеличивать время, но обязательно награждать себя после завершения.

Факт переключения задачи (многозадачность) требует энергии. Задачи, которые мы делали часто, выполняются проще, на автомате.

Отвлекающий фактор	Решение отвлекающего фактора
Звонок телефона	Убирать телефон на время Pomodoro

Активное запоминание

Запись ключевых идей. Повтор ключевых идей через страницу - две чтения.

Три ключа эффективного чтения:

- Предварительный просмотр
- Аккуратное чтение
- Активное запоминание (повтор ключевых идей). Также необходимо учиться в различных местах для лучшего усвоения материала.

Метафоры

Очень помогают при освоении и запоминании нового.

? ??????????????

1. Лучшие инженеры всегда стремятся решить проблемы пользователей

Очень легко влюбиться в технологию и начать искать, где бы её применить. Но разработчики, создающие наибольшую ценность, действуют наоборот — они глубоко озадачиваются проблемами пользователей и генерируют свои решения из этого состояния.

Проявляется эта озадаченность в виде выделения времени для обработки тикетов, общения с пользователями, рассмотрения их проблем и ломания головы до тех пор, пока не удастся докопаться до сути. Когда разработчик по-настоящему вникает в проблему, он часто выясняет, что элегантное решение проще, чем предполагалось.

Те же, кто сразу переходят к поспешному решению, часто в стремлении его оправдать всё слишком усложняют.

2. Доказать свою правоту не главное, главное — найти общий консенсус

Вы можете победить во всех технических спорах и при этом потерять проект. Я видел, как прекрасные инженеры всякий раз молча копят недовольство, считая себя самыми умными из участвующих в дискуссии. Позднее это выливается в «загадочные сложности при реализации проекта» и «странное сопротивление».

Но в таких случаях мастерство проявляется не в том, чтобы всегда оказываться правым. Оно заключается в способности принять адекватное участие в дискуссии для согласования общего видения задачи, в проявлении терпимости ко мнению других и более скептичном отношении к собственной непогрешимости.

Важно уметь выражать своё уверенное мнение, но при этом быть готовым его изменить — и не потому, что тебе недостаёт напористости, а потому что решения, принимаемые в условиях неопределённости, не должны быть привязаны к отдельной личности.

3. Будьте решительны, не тяните с релизом — можно исправить неудачную страницу, но нельзя исправить пустую

Квест по достижению совершенства сильно затягивает. Я наблюдал, как разработчики неделями спорят о том, какой будет идеальная архитектура для того, что они никогда не создавали. Идеальное решение редко рождается в результате одних только размышлений — чаще это происходит в результате контакта с реальностью. И здесь вполне способен помочь ИИ.

Сначала просто сделайте что-то, потом доведите до ума, а затем уже улучшайте. Не бойтесь выкатить пользователям грязный прототип. Пусть первая версия проектной документации будет неразборчивой. Поставьте MVP, за который будет немного стыдно. За неделю

реальной обратной связи вы узнаете больше, чем за месяцы теоретических споров.

Динамика вносит ясность. Аналитический паралич не создаёт ничего.

4. Стремитесь к понятности, замудрённость ведёт к издержкам

Инстинкт писать мудрёный код присущ почти всем разработчикам. Как будто это должно подтверждать их компетентность.

Но процесс разработки ПО включает в себя элемент времени и участие множества программистов. В таком контексте ясность перестаёт быть личным предпочтением и становится средством снижения операционных рисков.

Ваш код является стратегической памяткой для незнакомых людей, которые будут отлаживать его в 2 часа ночи во время сбоя. Так что при его написании акцент нужно делать на понятности, а не на видимой элегантности. Большинство уважаемых мной старших инженеров научились разменивать замудрённость кода на его ясность.

5. Инновации — это займ, который вы выплачиваете в виде сбоев, найма новых сотрудников и когнитивной нагрузки

Рассматривайте свои технические выборы как организацию, имеющую небольшой бюджет «токенов инноваций». Тратьте по одному каждый раз, когда внедряете что-то нестандартное. Много таких расходов вы себе позволить не можете.

И здесь главный смысл не в том, чтобы отказаться от инноваций, а в том, чтобы «внедрять их только тогда, когда это даёт некую уникальную отдачу». В остальных случаях нужно следовать скучными, избитыми путями, потому что их опасности уже известны.

Лучшим инструментом для реализации какой-то отдельной задачи часто будет «наименее проблемный из тех, которые подходят для многих задач», так как управляться с целым зоопарком экзотики будет намного сложнее.

6. Вас продвигает не код, а люди

В начале карьеры я считал, что отлично проделанная работа должна говорить сама за себя. Это было ошибкой. Код лежит себе в репозитории и помалкивает, а вот ваш менеджер либо упоминает вас на встрече, либо нет, равно как и коллега или просто знакомый, который может предложить вашу кандидатуру на какой-то перспективный проект.

В больших организациях решения принимаются на встречах, куда вас не приглашают, на основе отчётов, которые вы не писали, занятыми людьми, у которых есть 5 минут и 12 приоритетов. Если никто не сделает акцент на вашей роли, когда вас нет на встрече, то ваша роль становится, по сути, посредственной.

И речь не обязательно о собственном росте, а о том, чтобы сделать цепочку ценностей в организации прозрачной для всех, включая вас самих.

7. Лучший код — это тот, который не пришлось писать

В культуре разработки мы чествуем созидание. Никого не продвигают за удаление кода, хотя нередко это несёт для системы больше пользы, чем его добавление. Каждая строка кода, которую вы не напишете — это строка, которую вам не придётся отлаживать, обслуживать или объяснять.

Прежде, чем создавать что-либо, ответьте на вопрос: «А что произойдёт, если этого просто… не делать?» Иногда ответом будет «ничего плохого», и именно это станет вашим решением.

И проблема не в том, что инженеры не могут писать код или используют для этого ИИ. Дело в том, что мы настолько в этом хороши, что порой забываем подумать, когда те или иные вещи вообще нужны.

8. В больших масштабах даже у ваших багов есть пользователи

Что бы вы изначально своим пользователям ни обещали, при их достаточном числе любое наблюдаемое поведение становится зависимостью. Кто-то скрейпит ваш API, встраивает в свои рабочие процессы нештатные фичи ПО и кэширует баги.

Из всего этого следует важный для карьеры вывод: нельзя рассматривать обеспечение совместимости как «обслуживание», а добавление новых функций как «реальную работу». Совместимость — это тоже продукт.

Проектируйте окончание поддержки тех или иных функций в виде миграции с учётом необходимого времени, инструментов и интересов пользователей. По факту значительная часть процесса «проектирования API» состоит как раз из отключения устаревших функций API.

9. «Медлительность» большинства команд зачастую объясняется их рассогласованностью

Когда проект дрейфует, мы инстинктивно начинаем винить исполнителей — люди работают недостаточно усердно, технология выбрана не та, разработчиков недостаточно. Но обычно всё это не является реальной проблемой.

В крупных компаниях команды выступают единицами внутренней «валюты», но по мере роста их числа затраты на координацию растут в геометрической прогрессии. И в основном медлительность объясняется проблемами согласованности — люди создают не то, что надо или без учёта необходимой совместимости.

Старшие разработчики тратят больше времени на прояснение общего направления, интерфейсов и приоритетов, чем на «ускорение написания кода», потому что именно эти аспекты оказываются узким звеном.

10. Акцентируйтесь на том, что можете контролировать, игнорируя то, что не можете

В больших компаниях огромное число переменных находятся вне зоны вашего контроля. Это и организационные изменения, и решения руководства, и колебания рынка, и переориентация продукта. Уделение всему этому лишнего внимания только порождает лишнюю тревожность по тем вопросам, на которые вы повлиять не в силах.

Инженеры, которые смотрят на вещи здраво, акцентируют внимание на своей сфере контроля. Вы не можете повлиять на то, произойдёт ли реорганизация, зато можете изменить свою реакцию и качество своей работы, повлияв на получаемый опыт. Оказываясь в условиях неопределённости, разбивайте задачи на части и определяйте, какие конкретно действия для их решения вам доступны.

И это будет признаком не пассивного принятия, а стратегического фокуса. Энергию, потраченную на то, что вы не в силах изменить, можно потратить на то, что вам подвластно.

11. Абстракции не убирают сложность, а лишь сдвигают её до момента, когда на дежурстве окажетесь вы

Создавая любую абстракцию, вы делаете ставку на то, что вам не придётся понимать происходящее под её навесом. Иногда эта ставка себя оправдывает. Но рано или поздно что-то да ускользает, и когда это происходит, вам нужно знать, с чем конкретно вы имеете дело.

Старшие инженеры продолжают осваивать «низкоуровневые» детали, даже когда это требует всё больше усилий. И дело не в ностальгии, а в осознании того, что в один прекрасный момент абстракция рухнет, и вы окажетесь один на один с системой в 3 ночи. Используйте свой стек. Но держите в уме рабочую модель потенциальных внутренних режимов отказа.

12. Самый быстрый способ освоить что-либо — это начать учить других

Письменное изложение информации помогает лучше её понять. Когда я объясняю некий принцип другим — будь то в документации, беседе, код-ревью или даже в чате с ИИ — то нахожу пробелы в собственном понимании. Сам факт попытки прояснить что-либо другим делает этот момент понятнее для меня самого.

Это не означает, что для освоения профессии хирурга вам нужно начать обучать ей других. Но в сфере разработки ПО эта идея чаще всего работает.

И дело не в том, чтобы щедро делиться знаниями. Фактически это просто эгоистичный лайфхак по самообучению. Если вы считаете, что разбираетесь в каком-то вопросе, попробуйте объяснить его простым языком. Моменты, на которых вы будете подвисать, укажут вам на пробелы в этом понимании. Обучение — это своего рода отладка собственных ментальных моделей.

13. Работа, которая делает возможной другую работу — бесценна и незаметна

Связующие элементы работы — документация, онбординг, координация команд, оптимизация процессов — жизненно необходима. Но если делать её бессознательно, она может затормозить ваш вектор технического развития и привести к выгоранию. И ловушка в том, что мы подходим к ней как к взаимопомощи, а не как к чётко очерченной задаче с видимыми результатами.

Ставьте для подобных процессов временные рамки. Преобразуйте их в артефакты: в документы, шаблоны и автоматические механизмы. И пусть они ведут к очевидным результатам, а не просто отражают личное качество.

Бесценность и незаметность — это опасная комбинация для вашей карьеры.

14. Если вы побеждаете в каждом споре, это наверняка ведёт к накоплению молчаливого несогласия оппонентов

Я научился допускать ошибочность в собственных убеждениях. Когда я «выигрываю» спор слишком легко, то обычно это говорит о том, что что-то не так. Люди перестают с вами спорить не потому, что вы их убедили, а потому что они устали пытаться, и это несогласие они в итоге выразят в работе, а не на встречах.

Для достижения реального согласия по вопросу требуется больше времени. Вам нужно вникнуть в другие точки зрения и понять их, собрать обратную связь и иногда открыто изменить своё мнение.

Краткосрочное чувство правоты куда менее ценно, чем долгосрочная работа над проектом с людьми, разделяющими общее видение.

15. Когда метрика становится целью, она перестаёт быть метрикой

Любая метрика, которую вы продемонстрируете руководству, в конечном итоге станет объектом манипуляций. И дело не в злом умысле, а в том, что люди по своей природе стремятся подстраиваться под измеряемые показатели.

Если вы отслеживаете эффективность работы по количеству строк кода, то будете получать их всё больше. Если эффективность отслеживается по скорости, команды начнут завышать предполагаемые сроки работы.

Как в этих случаях действуют опытные разработчики? Каждую запрошенную метрику они дополняют парной, которая её компенсирует. Одна для скорости — другая для оценки качества или рисков. При этом акцент также делается на интерпретации общих трендов, а не промежуточных результатов. Задача — получить чёткую картину, а не установить слежку.

16. Признать, что вы чего-то не знаете, безопаснее, чем притвориться, что знаете

Когда старший разработчик говорит «я не знаю», он не показывает слабость, а разряжает обстановку. Если лидер выражает неуверенность, это означает, что и другие участники совещания могут сделать то же самое. В организациях с альтернативной культурой, напротив, все делают вид, что всё понимают. В итоге реальные проблемы остаются незамеченными до наступления критических моментов.

Я видел команды, в которых ведущий руководитель никогда не признавал сомнений, и также видел печальные последствия этого поведения. В таких условиях не задаются нужные вопросы, высказанные предположения не подвергаются критике, и джуниоры молчат, потому что верят, что остальные всё понимают.

Выражайте интерес к обсуждению, и вы получите команду, которая будет по-настоящему учиться.

17. Ваши социальные связи переживут любую работу

В начале своей карьеры я фокусировался только на работе, пренебрегая социальными связями. Оглядываясь назад, я понимаю, что это было ошибкой. Мои коллеги, которые вкладывались в общение с другими людьми — как внутри компании, так и за её дверями — в итоге пожинали плоды десятилетиями.

Они раньше узнавали о возникающих возможностях, быстрее наводили мосты, получали рекомендации на перспективные места и основывали предприятия с людьми, с которыми выстроили доверительные отношения.

Ваша текущая работа рано или поздно закончится, а социальные связи останутся навсегда. Подходите к их построению с интересом и искренностью, а не из меркантильных соображений.

Когда придёт время двигаться дальше, именно связи с людьми зачастую открывают самые перспективные двери.

18. Чаще всего прирост производительности возникает, благодаря исключению лишней работы, а не добавлению новой

Когда системы замедляются, мы интуитивно стремимся что-то добавить: кэширование слоёв, параллельную обработку, более умные алгоритмы. Иногда это правильное решение. Но лично я чаще видел прирост производительности после ответа на вопрос «А что из текущих вычислений нам не нужно?»

Удаление лишней работы почти всегда несёт больше пользы, чем её ускорение. Самый быстрый код — это тот, который никогда не выполняется.

Прежде чем что-то оптимизировать, задайте вопрос, насколько вообще этот компонент необходим.

19. Процесс существует для уменьшения неопределённости, а не для отчётности на бумаге

Хорошие процессы упрощают координацию и уменьшают последствия сбоев. Плохие же просто создают бюрократический театр, никак не помогая найти крайних в случае ошибок.

Если вы не можете объяснить, каким образом некий процесс уменьшает риски или повышает ясность, то наверняка это просто лишние издержки.

И если люди тратят больше времени на документирование своей работы, чем её реальное выполнение, то здесь явно что-то не так.

20. В конечном итоге время дороже денег, так что действуйте соответственно

В начале своей карьеры вы ставите во главу угла деньги, а не время — и это нормально. Но в определённый момент члены этой пропорции меняются местами. Вы начинаете понимать, что время — это не возобновляемый ресурс.

Я наблюдал, как старшие инженеры выгорают в погоне за очередным повышением, усердно оптимизируя программы ради небольшого прироста к зарплате. Некоторым из них это удавалось. Но большинство потом задавались вопросом, насколько оно того стоило.

И секрет не в том, чтобы «усердно трудиться», а в том, чтобы «осознанно делать выбор в пользу тех или иных ценностей».

21. Коротких лазеек нет, но есть эффект накопления

Опыт возникает как результат осознанной практики, когда ты слегка выходишь за рамки своих текущих навыков, потом рефлексируешь… и делаешь так раз за разом. Годами. У него нет какой-то концентрированной формы.

Но в обучении есть одна приятная особенность — опыт растёт, когда мы создаём некие новые решения, а не просто генерируем новую информацию. Пишите не из формальных обязательств, а с упором на ясность. Создавайте примитивы, которые можно будет использовать повторно где-то ещё. Собирайте весь печальный опыт в полезные руководства.

Разработчик, который воспринимает свою карьеру как вложение в будущее, а не лотерейный билет, добивается гораздо большего.