Краткое руководство для Проджект Менеджера в IT

Привет, дорогой друг, читатель и начинающий IT Проджект Менеджер!

Этот документ уже помог немалому количеству PM начать свой путь в IT. Надеюсь, что и для тебя найдется полезная информация здесь. Я бы назвала этот документ – выжимкой всех главных материалов по теме Проектного Менеджмента.

Удачи в изучении новой и интересной специальности PM в IT!

Советы и напутствия:

- Давай четкие ответы, подготовленные заранее. Отвечай без сомнений (даже, если не уверен – старайся придумать вариант ответа и представить свою логику рассуждений)
- Отвечай структурировано и логично (техника STAR (situation, target, action, result) может помочь)
- Подавай себя уверенно, следи за речью: подготовленная, живая (не монотонная). Проявляй заинтересованность в диалоге, не утомляй длинными историями не по теме
- Отвечай на вопрос, который задают. Приводи интересные примеры из практики и сложные кейсы в своей работе. Демонстрирую свой уровень профессионализма
- Отвечать с позитивным настроем,
- Перед интервью - несколько раз просмотри свое резюме, подготовь самопрезентацию, проговори ее до начала интервью 2 раза

Что можно сделать для подготовки перед интервью:

- Чтение статей по типу: «Как пройти собеседование»
- Повторить основные термины по сфере
- Посмотреть видео прохождения интервью – насмотренность на примерах
- Кейсы и их решения
- Практика ответов на вопросы и проработка самопрезентации

Part I. Техническая часть

В чем отличие backend от frontend?
Frontend включает в себя все, что видят пользователи: цвета, шрифты, изображения, графики, таблицы и кнопки. Frontend-разработчики реализуют удобный для пользователей интерфейс с помощью кода. Также они отвечают за адаптивность сайтов и веб-сервисов, которые должны правильно отображаться на экранах всех размеров.
Frontend-язык: только 1 - JavaScript. (HTML - стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере; CSS (каскадные таблица стилей)).
Backend хранит и упорядочивает данные, а также следит за тем, чтобы на стороне клиента все работало хорошо. Backend скрыт от глаз пользователей.
Backend-языки: *Ruby, PHP, Python, Java, NodeJS, C#, Go, VB*
Связь между backend и frontend частями в разработке происходит следующим образом:
1. Frontend отправляет пользовательскую информацию в Backend.
2. В Backend информация обрабатывается.
3. После обработки данные возвращаются и принимают понятный для пользователя внешний вид.
***Очень полезное видео:** https://www.youtube.com/watch?v=XBu54nfzxAQ

Что такое стек?
Это набор инструментов, которые команда применяет при работе в проектах. Он включает в себя языки программирования, фреймворки, системы управления базами данных, компиляторы и другие инструменты

Дженерик (обобщенное программирование) - парадигма программирования, заключающаяся в таком описании данных и алгоритмов, которое можно применять к различным типам данных, не меняя само это описание.

SOAP и REST
SOAP и REST — это два стиля API, которые подходят к вопросу передачи данных с другой точки зрения. SOAP — это стандартизированный протокол, который отправляет сообщения с использованием других протоколов, таких как HTTP и SMTP.
SOAP и REST позволяют создавать собственный API.
API означает интерфейс прикладного программирования. Это позволяет передавать данные из приложения в другие приложения. API получает запросы и отправляет ответы через интернет-протоколы, такие как HTTP, SMTP и другие. Многие популярные веб-сайты предоставляют общедоступные API-интерфейсы для своих пользователей, например, Google Maps имеет общедоступный REST API, который позволяет настраивать Google Maps с вашим собственным контентом. Есть также много API, которые были созданы компаниями для внутреннего использования.
Спецификации SOAP являются официальными веб-стандартами.
В отличие от SOAP, REST — это не протокол, а архитектурный стиль. Архитектура REST устанавливает набор рекомендаций, которым необходимо следовать, если вы хотите предоставить веб-службу RESTful, например, существование без сохранения состояния и использование кодов состояния HTTP.
Поскольку SOAP является официальным протоколом, он поставляется со строгими правилами и расширенными функциями безопасности, такими как встроенная совместимость с ACID и авторизация. Более высокая сложность требует большей

пропускной способности и ресурсов, что может привести к снижению времени загрузки страницы. REST был создан для решения проблем SOAP. Поэтому у него более гибкая архитектура.
Он состоит только из простых рекомендаций и позволяет разработчикам реализовывать рекомендации по-своему. Он допускает различные форматы сообщений, такие как HTML, JSON, XML и простой текст, в то время как SOAP допускает только XML. REST также является более легкой архитектурой, поэтому веб-сервисы RESTful имеют более высокую производительность. Из-за этого он стал невероятно популярным в эпоху мобильных устройств, где даже несколько секунд имеют большое значение (как по времени загрузки страницы, так и по доходам).

Методы HTTP-запроса
Запросы: https://devanych.ru/technologies/metody-http-zaprosa
GET - запрашивает содержимое конкретного ресурса, получает данные и никак не должен изменять эти данные.
OPTIONS - определяет возможности и используемые методы веб-сервера. В ответном сообщении должен быть добавлен заголовок Allow с перечислением всех поддерживаемых сервером методов.
HEAD - похож на GET, но не возвращает тело ответа, а только стартовую строку и заголовки. Используется для получения метаданных, а также проверки и валидации ресурса.
POST - создает новый ресурс из переданных данных в запросе.
PUT - изменяет содержимое запроса по-указанному URI.
PATCH - похож на PUT, но применяется только к фрагменту ресурса.
DELETE - удаляет конкретный ресурс по-указанному URI.
TRACE- возвращает служебную отладочную информацию о том, какие данные добавляют или изменяют прокси-серверы в запросе.
CONNECT - запускает двустороннюю связь с запрошенным ресурсом. Метод обычно используется для открытия прозрачного TCP/IP-туннеля.

- **безопасные** (GET, HEAD, OPTIONS) — не изменяют данные, их можно выполнять в любой последовательности;
- **идемпотентные** (GET, HEAD, PUT, DELETE, OPTIONS, TRACE) — при повторном выполнении результаты будут ожидаемо одинаковыми;
- **не идемпотентные** (POST, PATCH) — при повторном выполнении результаты будут разными, если, например, отправить POST-запрос на создание элемента несколько раз подряд, то он может создать несколько элементов с одинаковыми данными.

JSON & XML
Для передачи информации как в интеграции, так и для сайтов используются определённые форматы данных. JSON и XML используются для получения и отправки данных с веб-сервера.
JSON (JavaScript Object Notation) — простой формат обмена данными, основанный на языке программирования JavaScript. Использует человеко-читаемый текст для передачи объектов данных.
JSON использует читаемый текст для хранения и передачи данных, содержащих массивы и значения пар атрибутов. Текст JSON можно легко преобразовать в объект JavaScript внутри JSON, а затем отправить на сервер. Он основан на JavaScript и эффективно используется с множеством языков программирования.
XML — это обширный язык разметки, созданный для переноса данных. Он определяет некоторый стандартный набор правил для кодирования файлов в читаемом формате. Цель разработки этого XML — сосредоточиться на простоте и удобстве использования в Интернете.

Socket - это действительно программный интерфейс. Это абстрактное понятие, которое, в большинстве случаев, используется для коммуникации программ в сети.
Socket = IP+порт
WebSocket - это протокол обмена данными (как, например, http, ftp, ssl и т.д.). Этот протокол идет поверх (передается посредством) протокола TCP.
При работе по протоколу WebSocket вы будете использовать обычные сокеты для соединения. Так же как и при работе с другими протоколами будут использованы сокеты (и для работы с http, с ftp и др.).
Протокол WebSocket создавался для того, чтобы можно было поддерживать длительные неразрывные соединения между браузером (который является клиентом) и веб-сайтом (который является сервером).
WebSocket — это двунаправленный протокол связи между клиентом (браузером) и сервером, позволяющий обмениваться сообщениями в режиме реального времени
Он устанавливает одно соединение и передает ответ на единственный запрос в тот момент, когда ответ появился — без дополнительных запросов, как у HTTP-протокола. Запросы и ответы приходят без задержек и сетевой нагрузки.
Подходит для маркетплейсов, бирж, игровых приложений, чатов, соцсетей, интернета вещей, push-уведомлений.
AJAX - Ajax использует протокол HTTP и может отправлять запросы с использованием методов POST / GET от клиента к серверу. В Ajax при отправке запроса сервер отправляет ответ для этого запроса, а соединение заканчивается.
AJAX - запрос → ответ. Создает соединение с сервером, посылает заголовки запроса с некоторыми данными, получает ответ с сервера, закрывает соединение. Запросы отправляются часто. Поддерживается во всех основных браузерах
Как плюс: просто реализовать, данные можно сжать.
Как минус: слишком много ненужных запросов. А также большие задержки между созданием и получением данных. Сервер отсылает данные не тогда, когда они появились, а когда прилетит новый запрос.

Long poll - запрос → ожидание → ответ. Создает соединение с сервером, как AJAX, но оставляет соединение открытым некоторое время (не сильно большое). Сервер НЕ реагирует на запрошенную информацию и ждет, пока не появится новой информации. Пока соединение открыто - клиент ждет данные с сервера. Как только что-то новое появилось у сервера - он отсылает ее клиенту. Клиент получив новую информацию или если истекло время ожидания *НЕМЕДЛЕННО* отсылает другой запрос серверу, запуская процесс ожидания на нем снова. Как правило, соединение обычно переустанавливают раз в 20-30 секунд, чтобы избежать возможных проблем, например с HTTP-прокси. Поддерживается во всех основных браузерах.
Как плюс: малое количество запросов (по сравнению с обычным AJAX)
Сервером называют компьютер, который предоставляет какой-то сервис (или несколько сервисов) в частной сети или интернете. Другие устройства — их ещё называют клиентами — могут подключаться к серверу через разные порты, чтобы получить этот сервис. Серверы обычно называют по имени сервиса, который они предоставляют. Например, если сервер принимает запросы через 80-й порт и отдает клиентам веб-страницы, такой сервер обычно называют веб-сервером. Также существуют файловые серверы, почтовые серверы, серверы аутентификации, серверы баз данных, серверы приложений и тому подобные.
Клиент — это устройство, которое подключается к серверам, чтобы пользоваться их сервисами или ресурсами. Клиентом может быть компьютер, веб-сайт, ПО или встроенная система. Например, когда вы заходите на веб-сайт, ваш браузер становится клиентом. По аналогии с именованием серверов, клиенты тоже называются в

зависимости от сервиса, которым они пользуются: бывают email-клиенты, веб-клиенты, клиенты баз данных или клиенты онлайн-чатов.
Монолитная клиент-серверная архитектура не лишена недостатков. Сбои в монолитных системах влияют на всю систему, а сопровождать их довольно сложно. При микро сервисном подходе разработчики разбивают большую монолитную систему на небольшие сервисы — микросервисы. Так называют слабосвязанный изолированный сервис, отвечающий за какой-то процесс.

База данных — это компонент, в котором хранятся сведения различных типов. Есть разные типы баз данных: SQL, облачные, «ключ — значение» (key-value), графовые и иерархические (documented). Серверы баз данных позволяют клиентам взаимодействовать с базами данных по специальным протоколам.

BA

Бизнес-аналитик - собирает и структурирует **требования** к продукту, делая их понятными для всех участников проекта.

!!! В чем разница между системным аналитиком, бизнес аналитиком и Data Analyst?
Системный аналитик работает с требованиями к программному обеспечению (ПО), автоматизированной или информационной системе, тогда как бизнес-аналитик не ограничивается только этими корпоративными активами, а обеспечивает возможность проведения изменений в организации, которые принесут пользу заинтересованным сторонам (стейкхолдерам) через выявление их потребностей и обоснование оптимальных решений.
Если же необходимо проанализировать так называемые «сырые данные» из разных источников и представленные в разных форматах, то **Data Analyst** выполняет целый комплекс специальных операций

- ➔ сбор данных;
- ➔ подготовка данных к анализу (выборка, очистка, сортировка);
- ➔ поиск закономерностей в информационных наборах;
- ➔ визуализация данных для быстрого понимания имеющихся результатов и будущих тенденций;
- ➔ формулирование гипотез по улучшению конкретных бизнес-метрик за счет изменения других показателей.

Иногда аналитик больших данных **(Big Data)** также занимается разработкой и тестированием моделей машинного обучения **(Machine Learning**). Однако, в большинстве случаев, Machine Learning является областью ответственности другого Big Data специалиста – исследователя или ученого по данным **(Data Scientist'a)**

Тест-кейс — это такое описание проверки работы системы, которое может выполнить **любой** человек из команды, будь то тестировщик, разработчик, аналитик или даже бизнес-заказчик.
Тестировщик создает тест-кейс, чтобы проверить, работает ли определенная фича должным образом, и чтобы подтвердить, что функционал, UI/UX и другие параметры системы удовлетворяют всем соответствующим стандартам, руководствам и требованиям клиентов

- ***Тест-кейсы*** – документация по исполнению тестирования:
  - ожидаемый результат,
  - шаги, которые должны привести к ФР (инструкция исполнения).
- Тест-кейс – аналог спека для программиста
- Хранится в CVS

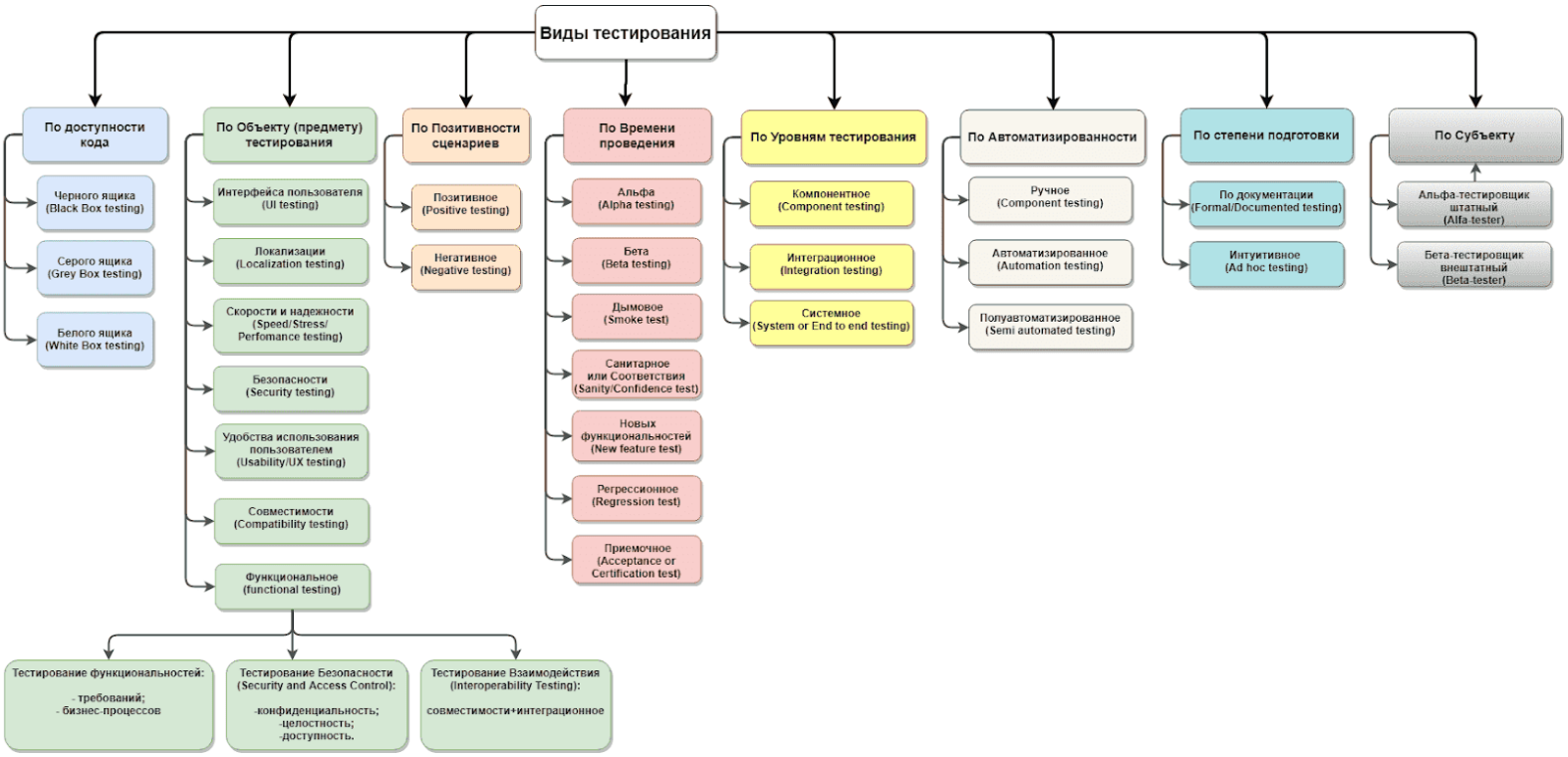
Идея: Покупка в интернет магазине
Шаги:
1. Открыть страницу 2. Войти в аккаунт “a” с паролем “1” 3. Выбрать товар “b” 4. Поместить в корзину 5. Оплатить товар кредитной картой “123”
ОР:
1. Создан заказ, содержащий товар “b” 2. Должна быть произведена оплата с карты “123”

В чем связь бизнес аналитика и QA?
Бизнес аналитик подготавливает требования, которые должны быть у системы, программы, проекта. QA, в свою очередь, проверяет, что работа функционала соответствует условиям, которые написаны в требования.

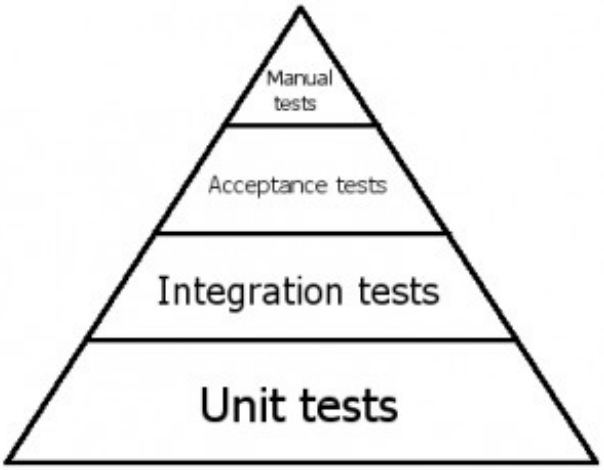
QA (Quality Assurance)

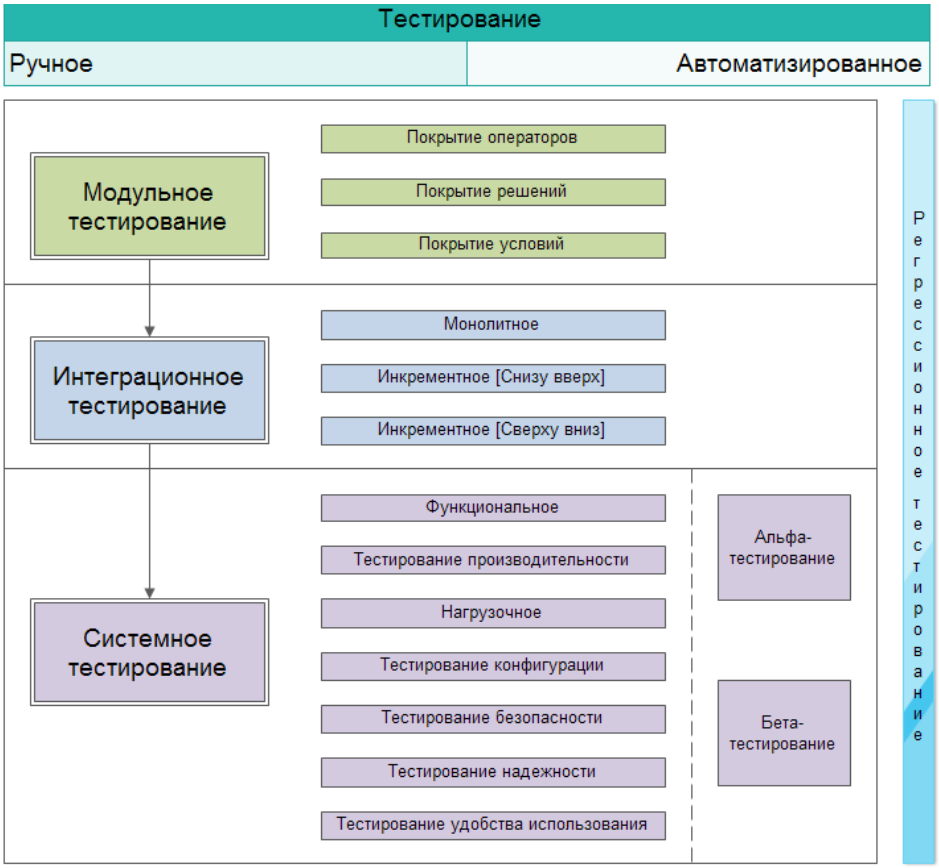
Выделяют 4 основных уровня тестирования:

- Компонентное/модульное тестирование (Component/Unit Testing)
- Интеграционное тестирование (Integration Testing)
- Системное тестирование (System Testing)
- Приемочное тестирование (Acceptance Testing)



Пирамида Тестирования





Дымовые тесты (Smoke Tests): выполняются каждый раз, когда мы получаем новый билд (версию), проекта (системы) на тестирование, при этом считая её относительно нестабильной. Нам нужно убедиться что критически важные функции AUT (Application Under Test) работают согласно ожиданиям. Идея данного вида тестирования заключается в том, чтобы выявить серьезные проблемы как можно раньше, и отклонить этот билд (вернуть на доработку) на раннем этапе тестирования, чтобы не углубляться в долгие и сложные тесты, не затрачивая тем самым время на заведомо бракованное ПО.

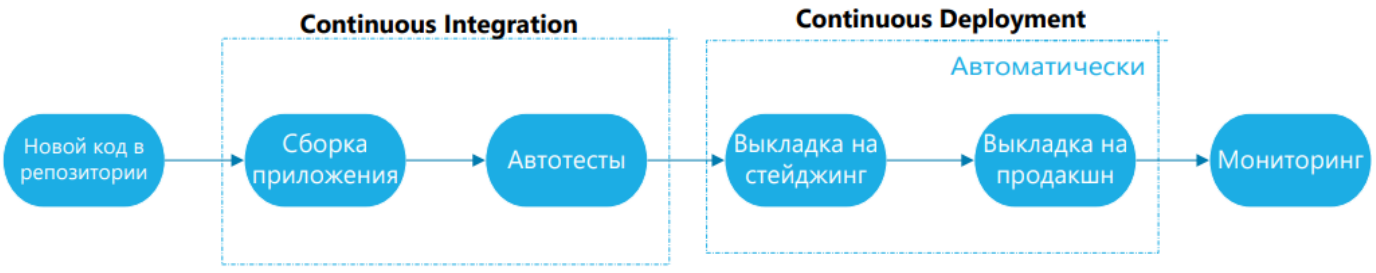
Санитарное тестирование: используется каждый раз, когда мы получаем относительно стабильный билд ПО, чтобы определить работоспособность в деталях. Иными словами, здесь проходит валидация того, что важные части функциональности системы работают согласно требованиям на низком уровне.

Ре-тест: проводится в случае, если фича/функциональность уже имела дефекты, и эти дефекты были недавно исправлены
Регрессионные тесты: собственно то, что занимает львиную долю времени и для чего существует автоматизация тестирования. Проводится регрессионное тестирование AUT тогда, когда нужно убедиться что новые (добавленные) функции приложения / исправленные дефекты не оказали влияния на текущую, уже существующую функциональность, работавшую (и протестированную) ранее.
Юнит-тесты - это тесты белого ящика, которые проверяют отдельные части кода, такие как функции, методы и классы. Они должны быть написаны на том же языке, что и тестируемый продукт и храниться в том же репозитории. Они часто прогоняются как часть сборки, чтобы сразу же увидеть успешно ли завершается тест или нет.
Интеграционные тесты - это тесты черного ящика, которые проверяют, что интеграционные точки между компонентами продукта работают корректно. Тестируемый продукт должен быть в активной фазе и развернут на тестовом окружении. Тесты сервиса часто являются именно тестами интеграционного уровня.
End-to-end тесты - это тесты черного ящика, которые проверяют пути выполнения системы. Их можно рассматривать как многошаговые интеграционные тесты. Тесты для веб-интерфейса часто являются именно end-to-end тестами.

DEVOPS

DevOps - это подход, методология и даже культура и философия процесса разработки, при котором программисты, тестировщики и системные администраторы могут работать над продуктом быстрее и эффективнее.
Подход помогает снизить ошибки при передаче проекта от разработчиков к тестировщикам и сисадминам и наладить между ними взаимодействие.
В основе лежит идея, что разработка, тестирование и эксплуатация цифровых продуктов — это единый, бесшовный и циклический процесс.

Это набор практик, нацеленных на активное взаимодействие и интеграцию специалистов по разработке и эксплуатации.
Нацелена на то, чтобы помогать организациям быстрее создавать и обновлять программные продукты (популяризация термина DevOps в 2009 г.).



Ссылка на презентацию:
http://samokhvalov.info/blog/pictures/DevOps%20%D0%BE%D1%82%20%D0%90%20%D0%B4%D0%BE%20%D0%AF.pdf

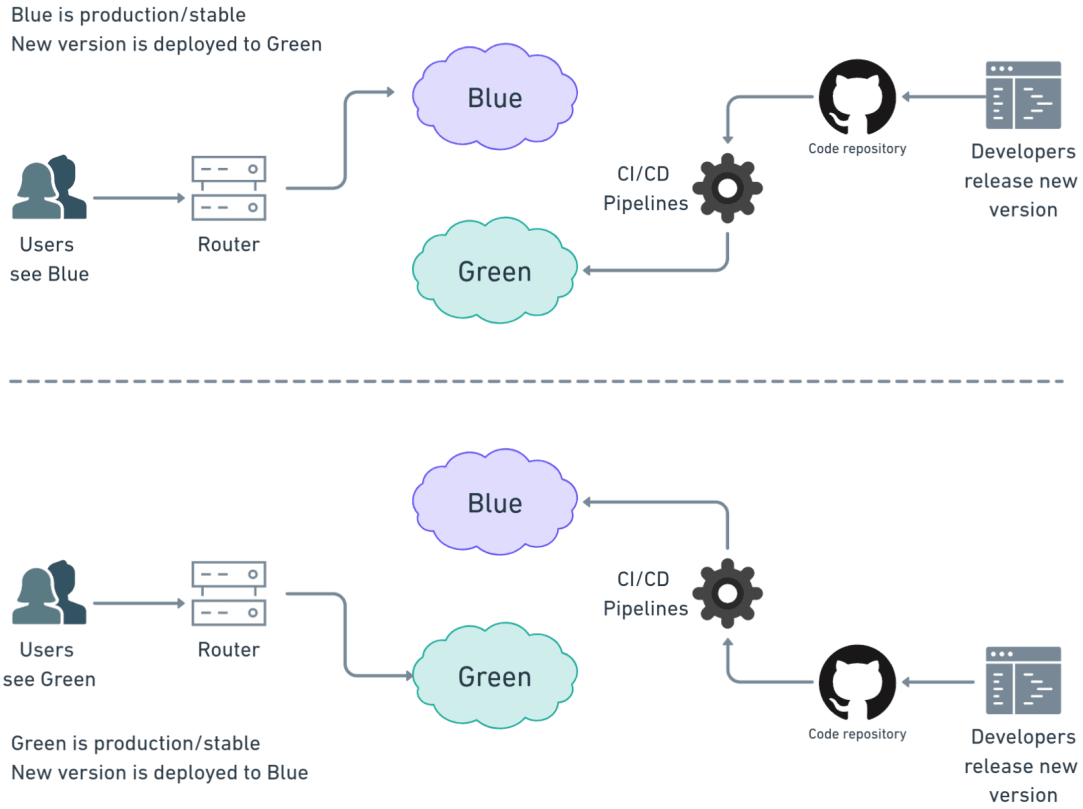
chmod (от англ. change mode) — команда для изменения прав доступа к файлам и каталогам, используемая в Unix-подобных операционных системах
Архитектура веб-сервиса, которая позволит переключаться между версиями по распространенным схемам развертывания:

- с помощью сине-зеленого развертывания (blue-green deployment) и
- канареечного развертывания (canary deployment).

Обе схемы используют пару бэкендов: «синий» и «зеленый».
Сначала на одном из бэкендов (например, на «синем») размещается стабильная версия, доступная пользователям, а другой («зеленый») используется для тестирования следующей версии. Когда тестирование окончено, бэкенды меняются ролями:

- **При сине-зеленом развертывании** весь пользовательский трафик одномоментно перераспределяется с одного бэкенда на другой. Сине-зеленое развертывание (blue-green deployment) — это метод внесения изменений в веб-сервер, приложение или сервер базы данных, путем замены чередующихся промышленных и промежуточных серверов.
- **При канареечном развертывании** переключение происходит постепенно, начиная с части пользователей.

После этого основным становится «зеленый» бэкенд, а на «синем» бэкенде можно тестировать следующую версию сервиса. Также, пока на «синем» бэкенде остается предыдущая версия, на нее можно откатить сервис, снова поменяв бэкенды ролями.



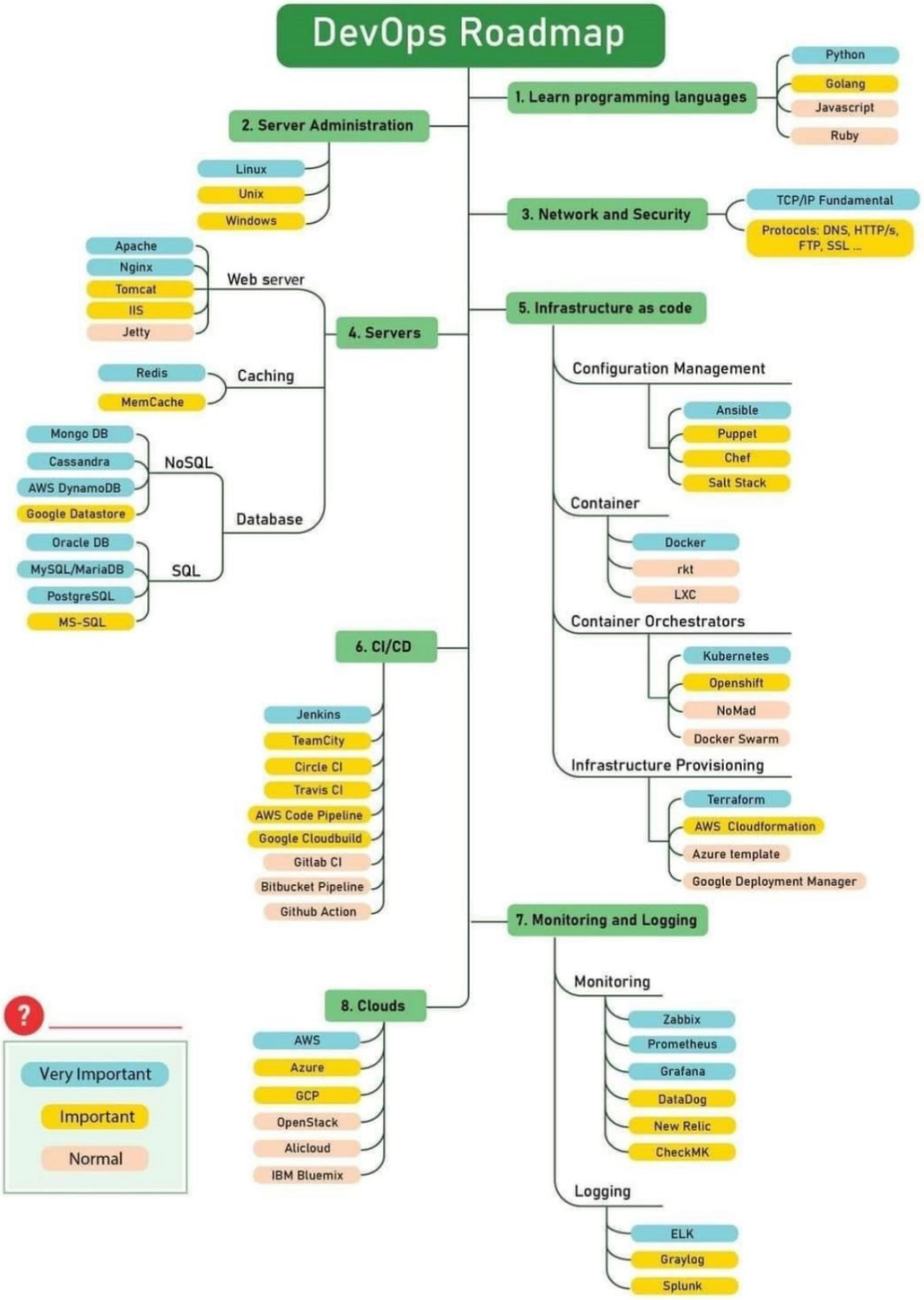
Канареечный релиз (canary release) — это метод снижения риска внедрения новой версии в эксплуатацию путем предоставления изменения небольшому подмножеству пользователей с нарастанием до состояния, когда это изменение становится доступным для всех

Балансировщик нагрузки (Load Balancer) — сервис, помогающий серверам эффективно перемещать данные, оптимизирующий использование ресурсов доставки приложений и предотвращающий перегрузки. Он управляет потоком информации между локальным или облачным хранилищем и конечным устройством — ПК, ноутбук, планшет или смартфон.

С помощью инструмента Docker девопс организует упаковку кода в контейнеры. Примерно как обычные грузовые контейнеры, которые благодаря стандартным габаритам укладываются, как конструктор, на палубе любого сухогруза. Только внутри виртуальных контейнеров — код, который «едет» на сервер, чтобы там исполняться. Девопс настраивает систему управления контейнерами — Swarm. Она следит за их состоянием, совместной работой, распределяет нагрузку.

Контейнеризация помогает:

- увеличить скорость выкладки кода на сервер, так как контейнер при запуске уже готов к работе
- быстро вернуться к предыдущей версии, если в новой обнаружились проблемы: закачиваем старые контейнеры — и снова все работает
- В итоге программисты сосредоточены на своей главной задаче: пишут код. А девопс помогает пройти весь остальной путь: собирает проверенные коды в контейнер, отправляет его на сервер и добавляет к уже работающей программе



Окружения среды разработки:

- **Local -** компьютер разработчика
- **Development -** сервер разработки выступающий как песочница, где разработчик может выполнить unit-тестирование
- **Integration -** основа для построения CI, или для тестирования сайд-эффектов разработчиком

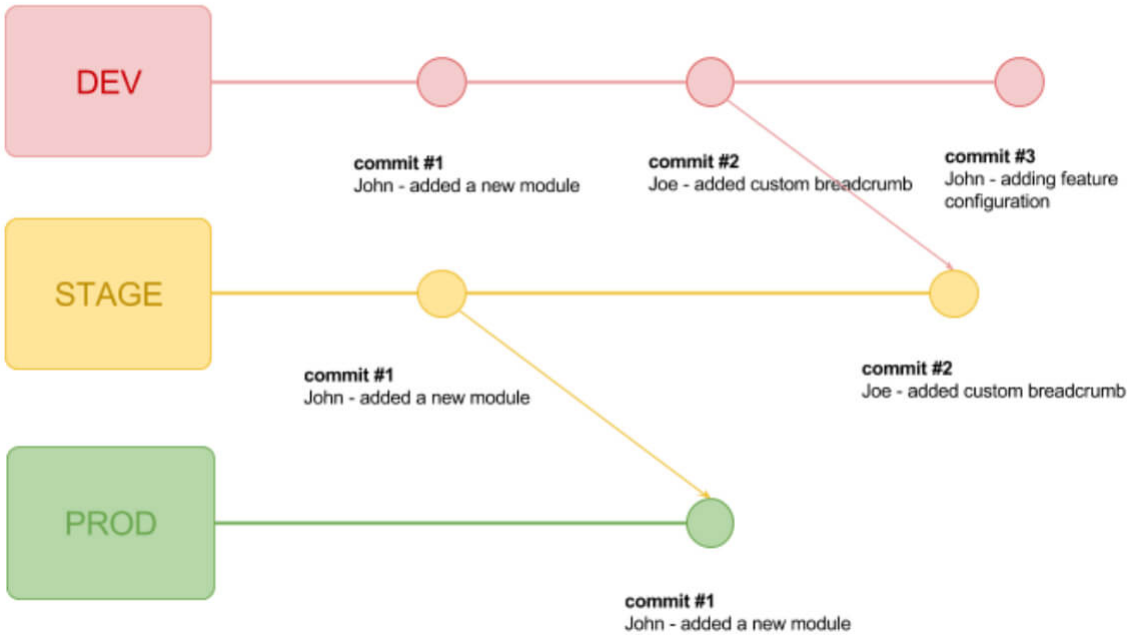
- **Testing/Test/Internal Acceptance -** окружение в котором выполняется тестирование интерфейса. Команда контроля качества проверяет что новый код не будет иметь влияния на существующую функциональность системы после деплоя нового кода в тестовое окружение.
- **Staging/Stage/Model/Pre-Production/Demo -** зеркало прод окружения
- **Production/Live -** серверы конечных пользователей/клиентов

Мерж реквест / Пул реквест - мерж-реквест или пулл-реквест создаётся в системе управления git-репозиториями. Это запрос на мерж одной ветки в другую, подобно задаче, назначаемый на какого-либо исполнителя. GitHub и Bitbucket используют термин «пулл-реквест», потому что первое необходимое действие — сделать пулл предлагаемой ветки.

Деплой - это развертывание/помещение исполняемого кода на сервер, где он будет работать.

Deploy - это целый процесс действий, которые делают программный продукт готовым к использованию:

- ➔ выпуск;
- ➔ установка;
- ➔ активация;
- ➔ адаптация;
- ➔ обновление;
- ➔ исправление ошибок и другие.



Docker — больше, чем сами контейнеры. Это обширный набор инструментов для разработчиков, позволяющий создавать, публиковать, запускать и управлять контейнерными приложениями. Это платформа, которая позволяет упаковать в контейнер приложение со всем окружением и зависимостями, а затем доставить и запустить его в целевой системе.

Создание образов - docker Build создает базовый образ контейнера, который включает все необходимое для запуска приложения — его код, двоичные файлы, сценарии, зависимости, конфигурацию, переменные среды и другое. Для определения и запуска много контейнерных приложений применяется инструмент docker compose.

Docker daemon - сервис, через который осуществляется взаимодействие с контейнерами
Docker client - интерфейс командной строки управления Docker daemon
Docker image - файл (образ), из которого разворачиваются контейнеры
Docker container - развернутое из образа приложение
Docker registry - репо с докер образами
Docker file - инструкция для сборки образа
Kubernetes (K8s) — портативная платформа оркестрации контейнеров с открытым исходным кодом. Этот инструмент используется для автоматизации развертывания контейнерных приложений на разных хостах, а также планового масштабирования и управления ими.

Для чего нужен Kubernetes

- Управление контейнерами на нескольких хостах одновременно.
- Оптимизация ресурсов используемого оборудования.
- Автоматическое развертывание и обновления приложений.
- Подключение и добавление хранилищ для запуска приложений с отслеживанием состояния.
- Масштабирование контейнерных приложений и их ресурсов на лету.
- Декларативное управление службами, что гарантирует полный контроль над развернутыми приложениями.
- Автоматический контроль работоспособность и восстановление приложений с помощью функций автозапуска, автозамены, авторепликации и автомасштабирования.

Виртуальная машина — это полноценная операционная система внутри другой ОС, с собственным ядром и другими изолированными ресурсами. Контейнер — не готовый «компьютер», а лишь изолированный механизм для запуска одного приложения.

В отличие от аппаратной виртуализации, контейнеризация обеспечивает разделение ресурсов не на аппаратном уровне, а на базе ядра операционной системы. Контейнеры более легковесны, менее требовательны и полностью зависимы от «материнской» ОС, чем VM.

Контейнер приложения — экземпляр исполняемого программного обеспечения (ПО), который объединяет двоичный код приложения вместе со всеми связанными файлами конфигурации, библиотеками, зависимостями и средой выполнения.
Смысл и главное преимущество технологии в том, что контейнер абстрагирует приложение от операционной системы хоста, то есть остается автономным, благодаря чему становится легко переносимым — способным работать на любой платформе
Зачем нужна контейнеризация

- Контейнеры решают критически важную проблему переносимости кода. Они сводят на нет возможные противоречия между собственной локальной средой разработки и производственной средой приложения.
- Упаковка в контейнеры позволяя отделить код от базовой инфраструктуры, в которой он работает. В производственной среде этот контейнер можно запускать на любом компьютере, на котором есть платформа контейнеризации. В продакшене код будет работать также хорошо, как и на машине программиста.

Преимущества контейнеризации

- Контейнеризация стала ключевым техническим компонентом для обеспечения непрерывного развертывания и уменьшения жизненного цикла инновации продукта. Благодаря тому, что предприятия разбивают монолитную архитектуру

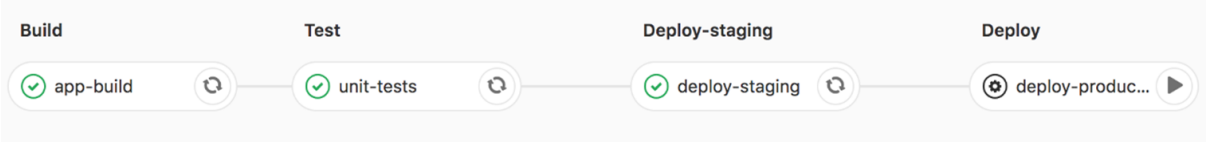
своих производств на гибкие контейнерные микросервисы, время вывода на рынок начинает отсчитываться не в месяцах, а в днях.

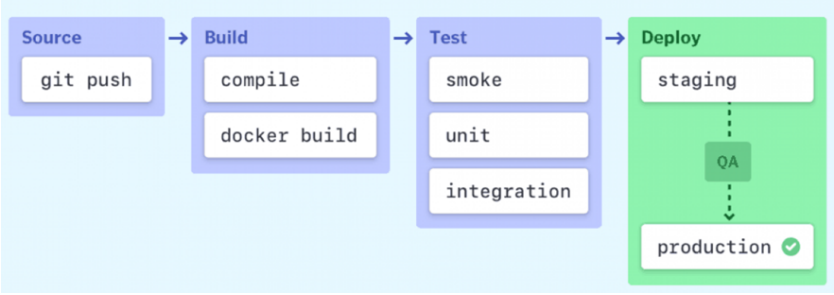
- Упрощение конфигурирования приложений. Стандартный контейнер Docker универсален. Упакованное в него приложение может работать без дополнительных настроек где угодно — на персональном компьютере (ПК, Mac, Linux), в облаке, на локальных серверах и даже на пограничных устройствах.
- Контейнерная технология крайне эффективна как средство повышения эффективности разработки. Небольшие группы могут разрабатывать и упаковывать свои приложения на локальных устройствах (например, ноутбуках), а затем развертывать его в практически любой тестовой или производственной среде. Время и усилия, сэкономленные при тестировании и развертывании, кардинально меняют правила игры во всем процессе автоматизации производства.
- Контейнеры легко воспроизвести, поэтому их удобно использовать для автоматического масштабирования приложений.

Что такое inode в Linux?
Индексный дескриптор в файловой системе, который хранит метаданные о файлах, за исключением имени файла. В дескрипторе хранится: длина файла в байтах, id устройства с файлом, id пользователя владельца файла, id группы файла, режим файла (права доступа), timestamp последнего изменения файла, счётчик хардлинков, указатели на блоки файла.
Что такое POSIX?
Набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов. Создан для совместимости различных Unix-подобных дистрибутивов.

CI/DC

Как выполняется настройка CI/CD, процесс: https://losst.ru/nastrojka-gitlab-ci-cd
Continuous Integration, Continuous Delivery — непрерывная интеграция и доставка — это технология автоматизации тестирования и доставки новых модулей разрабатываемого проекта заинтересованным сторонам (разработчики, аналитики, инженеры качества, конечные пользователи).
Pipeline - организованный процесс миграции кода из репозитория до конечного сервера через ci/cd.
build-test-staging-prod





Этапы CI/CD:

Написание кода. Каждый из разработчиков пишет код своего модуля, проводит ручное тестирование, а затем соединяет результат работы с текущей версией проекта в основной ветке. Для контроля версий используется система Git, либо аналогичные решения. Когда участники команды опубликуют код своих модулей в основной ветке, начнется следующий этап.
Сборка. Система контроля версий запускает автоматическую сборку и тестирование проекта. Триггеры для начала сборки настраиваются командой индивидуально — фиксация изменений в основной ветке проекта, сборка по расписанию, по запросу и т.д. Для автоматизации сборки используется Jenkins, либо аналогичный продукт.
Ручное тестирование. Когда CI система успешно проверила работоспособность тестовой версии, то код отправляется тестировщикам для ручного обследования. При этом тестовая сборка получает номер кандидата для дальнейшего релиза продукта (например, v.1.0.0-1).
Релиз. По итогам ручного тестирования сборка получает исправления, а итоговый номер версии кандидата повышается (например, после первого исправления версия v.1.0.0-1 становится v.1.0.0-2).
Развертывание. На этом этапе рабочая версия продукта для клиентов автоматически публикуется на production серверах разработчика. После этого клиент может взаимодействовать с программой и ознакомиться с ее функционалом
Поддержка и мониторинг. Конечные пользователи начинают работать с продуктом. Команда разработки поддерживает его и анализирует пользовательский опыт.

Инструменты для CI/CD
GitLab. Среда позволяет управлять репозиториями проекта, документировать функциональность и результаты доработок и тестов, а также отслеживать ошибки и работать с CI/CD pipeline.
Docker. Система автоматического развертывания проектов. Поддерживает контейнеризацию и позволяет упаковать проект вместе со всем окружением и зависимостями в контейнер, который можно перенести на Linux-систему.
Circle-CI. Продукт также использует бесшовную интеграцию с GitHub. Предлагается веб-интерфейс для отслеживания версий сборок и ведения задач. Также решение поддерживает отправку оповещений по почте, slack.
Jenkins. Довольно популярный инструмент в DevOps среде. Заслужил свою репутацию благодаря работе с различными плагинами, позволяющими гибко настраивать CI/CD процессы под требования разработки конкретного продукта. Jenkins система с открытым исходным кодом, то есть продукт доступен для просмотра, изучения и изменения, создан на Java. Дженкинс позволяет автоматизировать часть процесса разработки программного обеспечения, без участия человека. Данная система предназначена для обеспечения процесса непрерывной интеграции программного обеспечения. Jenkins является бесплатным инструментом, обладающим огромными возможностями в виде тысяч плагинов, которые постоянно добавляются и обновляются.
Преимущества Jenkins:
-режим работы сразу в двух и более средах;
-повышенная надежность развертываемого программного обеспечения;
-уменьшение ошибок, связанных с человеческим фактором;
-уменьшение затрат на персонал.
-упрощение рабочего процесса (нет необходимости нанимать дорогостоящую команду опытных специалистов)

Непрерывная интеграция (Continuous Integration, CI) это процесс разработки программного обеспечения, смысл которого заключается в **постоянном соединении**

рабочих копий в общую линию разработки, и выполнении постоянных автоматизированных сборок проекта для быстрого выявления возможных ошибок и решения интеграционных проблем.

GIT

Что такое система контроля версий?
Это сервис или программа, в которой хранятся данные о продукте, чтобы в любой момент можно было вернуться к любому этапу его создания. Самый известный сервис для этого — Git.
Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создает ссылку на предыдущую версию идентичного файла, который уже сохранен. Git представляет свои данные как, скажем, **поток снимков.**

*** **Код-ревью** — это процесс проверки кода, который позволяет: выявить ➔ ошибки, пропуски, уязвимости и стилистические недочеты (с точки зрения проекта или принятых в команде правил).

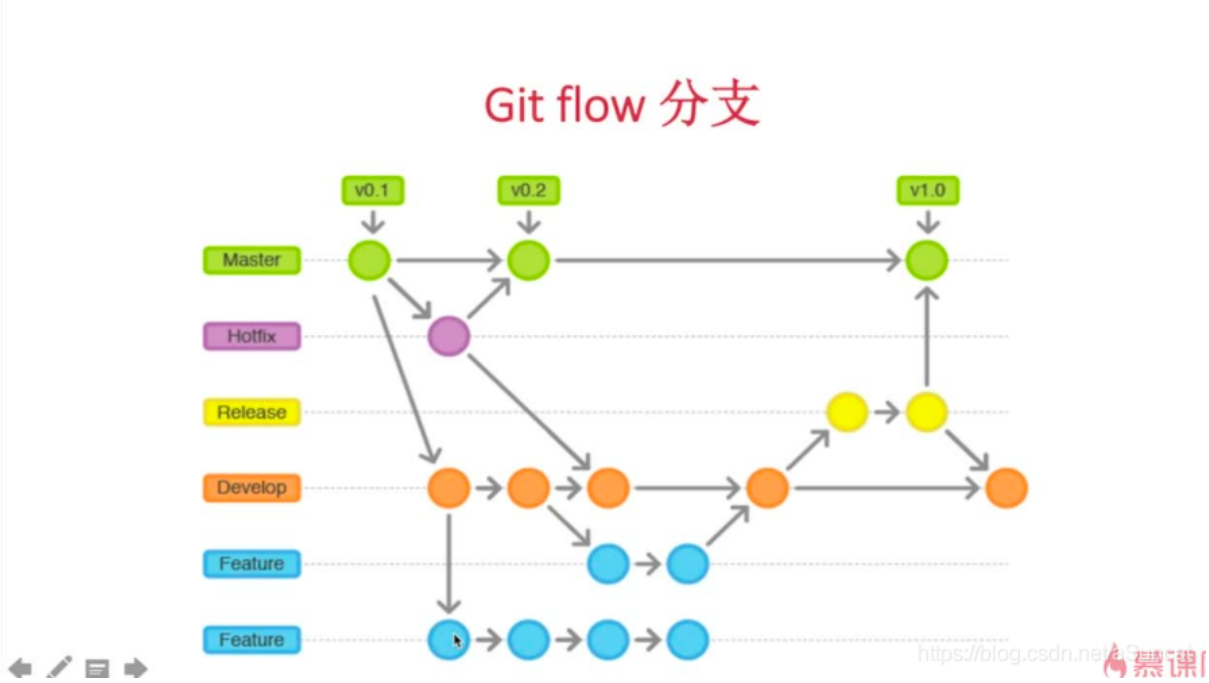
Основные команды Git:
- ➔ push
- ➔ pull
- ➔ merge
- ➔ rebase
- ➔ fetch

Git pull отличается от git fetch
git pull — это шоткод для последовательности двух команд: git fetch (получение изменений с сервера) и git merge (сливание в локальную копию).
При использовании fetch, git собирает все коммиты из целевой ветки, которых нет в текущей ветке, и сохраняет их в локальном репозитории. Однако он не сливает их в текущую ветку. Это особенно полезно, если вам нужно постоянно обновлять свой репозиторий, но вы работаете над функциональностью, неправильная реализация которой может негативно сказаться на проекте в целом. Чтобы слить коммиты в основную ветвь, нужно использовать merge.

Ссылка на README GIT, консольные команды:
https://github.com/cyberspacedk/Git-commands

Git Flow - альтернативная модель ветвления Git, в которой используются функциональные ветки и несколько основных веток. Эта модель была впервые опубликована и популяризована Винсентом Дриссеном. В Git-flow используется больше веток, каждая из которых существует дольше, а коммиты обычно крупнее.
В соответствии с этой моделью разработчики создают функциональную ветку и откладывают ее слияние с главной магистральной веткой до завершения работы над функцией. Такие долгосрочные функциональные ветки требуют тесного взаимодействия разработчиков при слиянии и создают повышенный риск отклонения от магистральной ветки. В них также могут присутствовать конфликтующие обновления.
Последовательность действий при работе по модели Gitflow:
- Из ветки main создается ветки develop, release
- Из ветки develop создаются ветки feature

- Когда работа над ветками feature и release завершается, они сливается в ветку develop
- Если в ветке main обнаруживается проблема, из main создается ветка **hotfix**
- Когда работа над веткой hotfix завершается, она сливается с ветками develop и main



Hosting

Виртуальный хостинг (Shared) - сервер, на котором расположено множество сайтов. Они используют одинаковое программное обеспечение и имеют равные возможности. На одном сервере может находиться около тысячи сайтов. Как правило, на таком хостинге располагают небольшие веб-ресурсы, не требующие больших мощностей и дискового пространства. Благодаря своей низкой стоимости и отсутствию необходимости в администрировании сервера, виртуальный хостинг — самый распространенный выбор среди пользователей.

Виртуальный выделенные сервер (VPS-сервер)

VPS отличается от виртуального хостинга тем, что владелец такого сервера может устанавливать и настраивать любое программное обеспечение. По сути, управление VPS-сервером не отличается от управления физическим сервером. Сайту на VPS выделены определенные ресурсы в соответствии с тарифом.

Выделенные физический сервер (Dedicated)

Пользователю дается отдельный сервер в дата-центре под самостоятельное управление с возможностью устанавливать любую операционную систему, менять программное обеспечение под свои нужды. Такой хостинг используется для масштабных ресурсоемких проектов с высокими требованиями и хорошо подойдет для крупных интернет-магазинов с высокой посещаемостью или, например, онлайн-игр.

Облачный хостинг (Cloud hosting)

Облачный хостинг — объединённая система серверов, на которых располагаются клиентские сайты. Таким образом, выделяемые для клиента мощности не

ограничиваются одним сервером, а распределяются сразу между несколькими серверами. Это обеспечивает бесперебойную работу ресурса вне зависимости от выхода из строя какого-то одного сервера. Кроме повышенной производительности, облачный хостинг привлекает клиентов и другими преимуществами. Во-первых, пользователь платит только за те ресурсы, которые использовал — цена услуги зависит от объемов потребления мощностей. Во-вторых, на облачном хостинге при изменении нагрузки выделенные ресурсы увеличиваются или уменьшаются автоматически.
Colocation — размещение оборудования клиента в дата-центре провайдера
Colocation буквально переводится как «соразмещение», что точно описывает этот вид хостинг-услуг. Ваш сервер располагается на технологической площадке провайдера (дата-центре), который обеспечивает его высокоскоростным интернет-каналом и заботится о прочих необходимых условиях содержания.

Порты

В компьютерах есть два вида портов — физические и программные (или сетевые):
Физический порт — это разъем на компьютере, с помощью которого можно подключить флешку, сетевой кабель, принтер, наушники и так далее. Физический порт обменивается электричеством с устройством, который вставлен в этот порт.
Сетевой порт — то число, которое записывается в заголовках протоколов транспортного уровня сетевой модели OSI. Оно используется для определения программы или процесса-получателя пакета в пределах одного IP-адреса.Порты позволяют определить сетевые приложения, работающие на компьютере, множество которых может быть запущено одновременно. Основными сетевыми программами могут быть:

- ➔ **WEB-сервер** – предоставляет трансляцию данных с веб-сайтов;
- ➔ **Сервер почты** – позволяет обмениваться электронными письмами;
- ➔ **FTP-сервер** – обеспечивает передачу информации.

Порт отображается в виде **16 битного числа от 1 до 65535**, которое доступно приложениям для того, чтобы обмениваться трафиком.
Для чего нужны порты?
Основным предназначением сетевых портов является прием и передача данных определенного вида, а также устранение ошибки неоднозначности при попытке установить связь с хостом по IP-адресу. Для обеспечения трансляции данных с веб-сервера необходимо указать IP адрес хоста и номер порта, определяющий программу веб-сервера.
Физические порты при подключении определяют, к какой программе относятся данные, которые получает сетевая плата. Например, для использования порта одновременно сетью и интернетом пользуются разные программы:

- ➔ ОС запрашивает обновления безопасности на сервере;
- ➔ браузер загружает страницу в соцсети;
- ➔ различные мессенджеры отправляют и принимают сообщения;
- ➔ фоном пользуетесь видеозвонком, который тоже отправляет видеоданные;
- ➔ различные облачные диски синхронизируют данные.

Ссылка на статью:
https://vc.ru/u/1193668-fenixhost/441676-chto-takoe-porty-kakie-byvayut-i-zachem-oni-nuzhny

Серверы

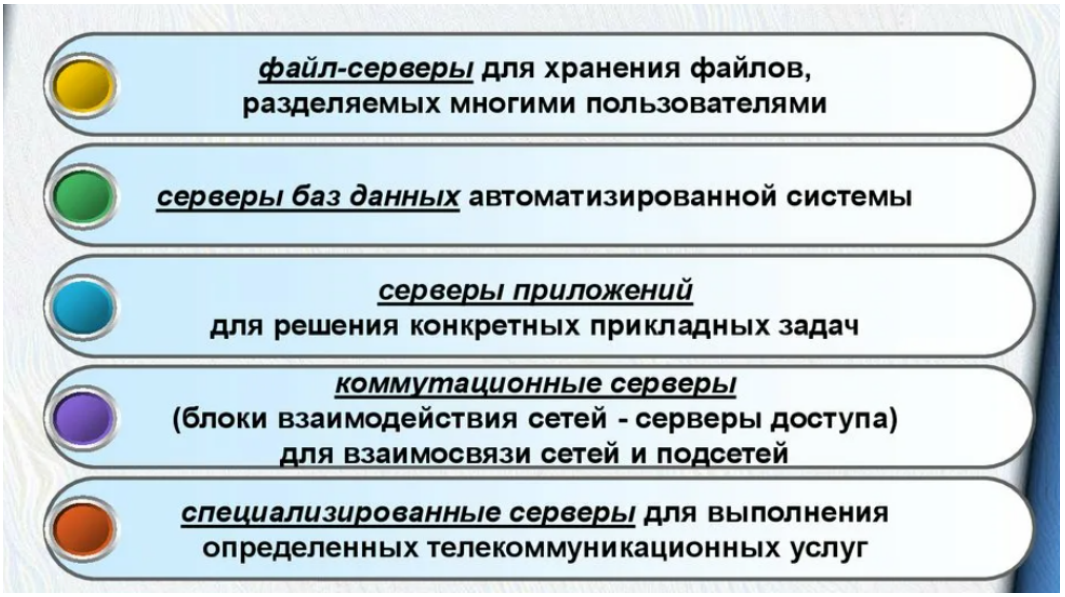
Для чего нужны серверы?
Основное направление — поддержка интернет-ресурсов. Количество задач, возлагаемых на сервер велико. Вот несколько сценариев, в которых необходимы подобные устройства:

- хостинг сайтов;
- разработка веб-приложений;
- платформа для приема и отправки электронных писем;
- хранение файлов;
- создание общего рабочего пространства для сотрудников одной фирмы;
- создание шлюзов (proxy или VPN), заменяющих информацию о подключившемся компьютере на другую;
- добыча криптовалюты

Подбор серверных систем происходит по сформулированным запросам заказчиков. Благодаря широкому ассортименту не просто с ходу подобрать подходящее устройство, чтобы оно удовлетворяло пожеланиям клиента и оперативно справлялось с поставленными задачами, поэтому **ориентируются на:**

- мощность;
- габариты;
- надежность;
- управляемость;
- масштабируемость;
- бюджет;
- готовность к работе

- ***Файл-серверы (file servers)*** – компьютеры + ПО, предоставляющие доступ к подмножеству своих файловых систем, расположенных на дисках, другим компьютерам локальной сети (LAN). *Пример: SAMBA* (*SMB* – от *Server Message Block*) – серверное ПО для ОС типа UNIX (Linux, FreeBSD, Solaris , etc.), обеспечивающее доступ с Windows-компьютеров LAN к файловым системам UNIX-машины. Samba также реализована для платформы Macintosh/MacOS
- ***Серверы приложений (application servers)*** – компьютеры + ПО, обеспечивающие вычислительные ресурсы для (удаленного) исполнения определенных классов (больших) приложений с других компьютеров LAN. *Примеры:* WebSphere (IBM), WebLogic (BEA) – наилучшие из известных application-серверов для приложений J2EE
- ***Серверы баз данных (database servers)*** – компьютеры + ПО (Microsoft SQL Server, Oracle, etc.), обеспечивающие доступ другим компьютерам сети к базам данных, расположенным на этих компьютерах
- ***Web-серверы (Web servers)*** – компьютеры + ПО, обеспечивающие доступ через WWW к Web-страницам, расположенным на этих серверах. *Примеры: Apache; Microsoft.NET Web Servers; Java Web Servers*
- ***Proxy-серверы*** – компьютеры + ПО, обеспечивающие более эффективное выполнение обращений к Интернету, фильтрацию трафика, защиту от атак
- ***Email-серверы*** – компьютеры + ПО, обеспечивающие отправку, получение и "раскладку" электронной почты для некоторой локальной сети. Могут обеспечивать также *криптование* почты (email encryption)
- ***(Server) back-end*** – группа (pool) связанных в LAN компьютеров (вместо одного сервера), обеспечивающая серверные функции



Типы серверов

- **Сервер приложений (Application Server)**
  - Динамичный контент
  - Обычно включает в себя веб сервер (для возможности обработки запроса по HTTP)
  - Есть возможность строить сложную инфраструктуру приложений для формирования динамических веб страниц
  - Дополнительные возможности (отказоустойчивость, кластеризация и пр.)
  - Middleware - связь между клиентом и данными, централизованный доступ к другим частям архитектуры
  - Имеет веб интерфейс для администрирования (независимо от операционной системы)

- В обоих случаях серверов – это программа, написанная на каком-либо языке программирования (C, Java)
- Часто под сервером подразумевается физическая машина

Сервер базы данных

С данным типом устройств есть возможность обрабатывать данные, которые хранятся совместно, при этом по правилам структурированные и организованные. Самые популярные **инструменты** для управления базами данных:

- MySql;
- MS SQL Server;
- Apache;
- Oracle.

Эти средства нужны, если бизнес процессы организации требуют отдельный вычислительный ресурс, когда подготавливают и обрабатывают данные. Серверная система будет с определенными параметрами, которые зависят от некоторых нюансов:

- количество пользователей;
- масштаб базы данных;
- характер обращений;
- динамика запросов.

Узел должен быть надежным и отказоустойчивым, чтобы доступность данных была на необходимом уровне.

Брандмауэры, файрволлы

Защитные системы, блокирующие отрицательное воздействие из интернета. Исходящие данные проходят без проблем. А обратная связь организована сложнее: полный анализ поступающего информационного потока. Сервер определяет опасные, вредоносные данные и извлекает их из общей информационной массы. Сегодня такие экраны отлично выполняют свои функции, защищают от атак, вирусов, которые так и норовят проникнуть из интернета и украсть информацию или все сломать.

Базы данных

Ссылка на полезный ресурс:
https://telegra.ph/Pochemu-SQL--ehto-must-have-dlya-menedzhera-v-IT-11-02

Система управления базами данных (СУБД) является субъектом управления и программой, а **база данных** объектом управления и собственно данными, которыми управляет СУБД.
******Кратко:***

- **БД** - это данные хранимые в структурированном виде. Обычно в виде набора файлов.
- **СУБД** - это программа или библиотека, которая умеет с этими файлами работать.

База данных (БД) — это специальным образом организованная совокупность данных о некоторой предметной области, хранящаяся во внешней памяти компьютера.

Система управления базой данных (СУБД) — это программные средства, которые позволяют выполнять все необходимые операции с базой данных.

БД + СУБД = **информационная система**

Примеры NoSQL: Mongo, Redis

Postman

Postman — это HTTP-клиент для тестирования API. HTTP-клиенты тестируют отправку запросов с клиента на сервер и получение ответа от сервера. API (Application Programming Interface) — это интерфейс для обмена данными с сервера между двумя приложениями или компонентами ПО. Тестировщикам Postman помогает в проектировании дизайна API и создании mock-серверов (имитаторов работы приложения).

Swagger

Swagger – это язык описания интерфейса для описания RESTful API, выраженных с помощью JSON. Swagger используется вместе с набором программных средств с открытым исходным кодом для проектирования, создания, документирования и использования веб-служб RESTful.

Part II. Project Management. Methodologies, Time Management, Risk management, People Management

PMBoK 2021г.:

- ➔ 5 этапов: ИНИЦИАЦИЯ - ПЛАНИРОВАНИЕ - ИСПОЛНЕНИЕ - КОНТРОЛЬ ЗАВЕРШЕНИЕ
- ➔ 8 доменов
- ➔ 10 областей знаний
- ➔ 12 принципов

-8 доменов:

Заинтересованные стороны. Была область знаний, а стал домен. Ничего, по сути, не изменилось.

Команда. Раньше PMBOK Guide не различал людей и асфальтоукладчики. Теперь авторы увидели различие и выделили команду в отдельный домен. Это выделение напрашивалось давно уже. Так что здесь явное улучшение.

Неопределенность. Была область «управление рисками», а стал домен неопределенность. Яйца в профиль.

Поставка результатов. Были две области знаний «Содержание» и «Качество», зачем-то разделенные. Отчасти логика была, т.к. тема качества развивается в отдельных

стандартах. Но с точки зрения практики отделение было нелогичным. Сейчас все стало на свои места – есть один домен, в котором нужно думать о продуктах и их качестве.
Жизненный цикл. Раньше эта тема отделялась от областей в красивую логику. Теперь это стало одним из доменов. Мне кажется, это менее удобным. Возможно, просто нужно привыкнуть. Радует, что тема жизненного цикла проработана глубже. Хотя ребята сами запутались и инкрементный подход проиллюстрировали неверно.
Планирование. Раньше был управленческий цикл из 5 шагов. Конечно, инициация и завершения были довольно куцыми, но вместе с ними последовательность была целостной, законченной. Теперь остались только три самых наполненных действиями шага – планирование, выполнение и мониторинг.
Выполнение. Был шаг управленческого цикла, теперь отдельный домен
Измерение прогресса. Был шаг управленческого цикла «мониторинг и контроль», теперь отдельный домен. Справедливости ради, стало больше четкости, т.к. речь идет только про метрики слежения за прогрессом.
PMBOK, как описано в Руководстве, распознает 49 процессов, которые делятся на пять основных групп процессов и десять областей знаний, типичных для большинства проектов

Области знаний	Группы процессов управления проектом				
	Группа процессов инициации	Группа процессов планирования	Группа процессов исполнения	Группа процессов мониторинга и контроля	Группа процессов закрытия
4. Управление интеграцией проекта	4.1 Разработка устава проекта	4.2 Разработка плана управления проектом	4.3 Руководство и управление работами проекта	4.4 Мониторинг и контроль работ проекта 4.5 Интегрированный контроль изменений	4.6 Закрытие проекта или фазы
5. Управление содержанием проекта		5.1 Планирование управления содержанием 5.2 Сбор требований 5.3 Определение содержания 5.4 Создание ИСР		5.5 Подтверждение содержания 5.6 Контроль содержания	
6. Управление сроками проекта		6.1 Планирование управления расписанием 6.2 Определение операций 6.3 Определение последовательности операций 6.4 Оценка ресурсов операций 6.5 Оценка длительности операций 6.6 Разработка расписания		6.7 Контроль расписания	
7. Управление стоимостью проекта		7.1 Планирование управления стоимостью 7.2 Оценка стоимости 7.3 Определение бюджета		7.4 Контроль стоимости	
8. Управление качеством проекта		8.1 Планирование управления качеством	8.2 Обеспечение качества	8.3 Контроль качества	
9. Управление человеческими ресурсами проекта		9.1 Планирование управления человеческими ресурсами	9.2 Набор команды проекта 9.3 Развитие команды проекта 9.4 Управление командой проекта		
10. Управление коммуникациями проекта		10.1 Планирование управления коммуникациями	10.2 Управление коммуникациями	10.3 Контроль коммуникаций	
11. Управление рисками проекта		11.1 Планирование управления рисками 11.2 Идентификация рисков 11.3 Качественный анализ рисков 11.4 Количественный анализ рисков 11.5 Планирование реагирования на риски		11.6 Контроль рисков	
12. Управление закупками проекта		12.1 Планирование управления закупками	12.2 Проведение закупок	12.3 Контроль закупок	12.4 Закрытие закупок
13. Управление заинтересованными сторонами проекта	13.1 Определение заинтересованных сторон	13.2 Планирование управления заинтересованными сторонами	13.3 Управление вовлечением заинтересованных сторон	13.4 Контроль вовлечения заинтересованных сторон	

Что такое скрам?

Фреймворк для разработки проектов, который помогает командам правильно приоритизировать задачи и работу над продуктом. Его основа — итеративная разработка и получение регулярной обратной связи от заказчиков и пользователей. В 2002 году был создан Scrum Alliance во главе с Кеном Швабером и Джефом Сазерлендом. А в 2009 году Кен Швабер официально вышел из Scrum Alliance и основал собственную компанию – Scrum.org. Важным шагом на пути к популяризации Scrum стал выпуск Скрам Гайда в 2010 году, который с момента публикации прошел несколько редакций – 2011, 2013 и 2016 год.

Чем Scrum отличается от Waterfall? Какие ещё есть методологии и фреймворки, как их использовать?

По сути это Спиральный и Каскадный метод разработки.
Scrum относится к спиральному методу разработки, постоянно увеличивая ценность продукта из итерации в итерацию (спринт). Инкрементный подход, опираемся на полученные знания для планирования следующей итерации. Ценностью данного подхода является постоянное улучшение продукта, в условиях минимальной формализации и отсутствием фиксированных сроков.
Waterfall (Предиктивный ЖЦ) относится к каскадным методам разработки, иначе говоря продукт распиливается на определённые вехи, каждый кусок оценивается по объему работ и фиксируется сроками и критериями приемки, переход между вехами возможен только в случае полного достижения критериев предыдущей. На практике возврат к предыдущей вехе приминяется, при этом происходит переоценка проекта и отодвигание сроков (увеличение затрат).

В каких случаях проекта выбирать скрам, а в каких – ватерфол?
В первую очередь, нужно оценить проект по следующим параметрам:

- Бюджет
- Сроки
- Объем и сложность
- Ожидания заказчика
- Тип проекта и индустрия
- Команда
- Организация
- Заинтересованные стороны
- Инструменты

Почему скрам НЕ работает на гос проектах?

- Scrum совершенно не подходит в случаях, если высока цена ошибки. То есть разработка отказоустойчивого железа или ПО не должна подразумевать, что его используют на самых критических важных объектах. Иначе говоря, Scrum не подходит для постройки ядерного реактора, дата-центра или самолета.
- Обязательно должна быть обратная связь, а заказчик обязан вовлекаться в процесс. Иначе говоря, заказчик должен быть заинтересованным, реагировать на результат или же принимать продукт. Это подходит для программ, сайтов.
- При низкой квалификации команды, нехватке бюджета, ресурсов и так далее, система тоже не будет функционировать. Ведь Scrum-методология заточена на максимальную эффективность за короткое время, а не долговременную разработку.
- Ограничения по части специализации используемых методов. Это следствие пункта 1: если нет возможности экспериментировать и проверять решения, то система будет неэффективной.
- Ограничение по части общения между членами команды, бюрократизация процесса. Здесь все просто — система просто погрязнет в бумагах, согласованиях и так далее. То есть исчезнет сам смысл Scrum-методики.
- Большое число исключений — следствие пунктов 2, 4 и 6. Если цена ошибки высока, а денег нет, при этом все этапы нужно согласовывать, тогда в методологии нет смысла.

Поэтому Scrum **не подходит для государственных проектов и заказов**, строительства, военных задач (кроме упомянутого в начале статьи тактического планирования и нанесения удара).

Что такое Agile? Какие у него принципы?
Это гибкий подход к управлению проектами. Работа разделена на множество итераций, и на каждом этапе могут вноситься изменения.
Принципы Agile:
4 важных принципа: https://agilemanifesto.org/
1. Люди и взаимодействие важнее процессов и инструментов.
2. Работающий продукт важнее исчерпывающей документации.
3. Сотрудничество с заказчиком важнее согласования условий контракта.
4. Готовность к изменениям важнее следования первоначальному плану.

Что такое Scrum и какие есть там роли?
Scrum — это фреймворк гибкой разработки ПО, основанный на принципах Agile.
Scrum подходит для небольших команд!
Кроме разработчиков в scrum-команду входят: Product Owner: он общается с заказчиком и управляет бэклогом.
Scrum-мастер: он следит за соблюдением принципов Scrum и убирает препятствия для эффективной работы.
В проекте также участвуют пользователи, стейкхолдеры, и клиент, если это заказная разработка.

Что такое спринт? Из чего он состоит?
Это интервал в неделю или две, за который команда выполняет запланированный объем работы. В каждом из них есть фазы дизайна, разработки, тестирования и демо фичи заказчику. В начале каждого спринта разработчики планируют задачи — новые, либо доработку старых.
Каждый день проходят 15-минутные собрания — члены команды обсуждают, что удалось сделать вчера, что нужно сделать сегодня и что может этому препятствовать. Спринт завершается ретроспективой — оценкой эффективности команды. Также в конце заказчики дают обратную связь по продукту, и если есть доработки, команда добавляет их в следующий спринт.

Что такое груминг бэклога?
Это встреча команды, на которой обсуждаются и оцениваются задачи на ближайшие несколько спринтов. Как правило, на 2-3 спринта вперед.

Что такое рабочий флоу (Work flow)?
Это последовательность рабочих операций. Включает в себя общие задачи команды, шаги по их реализации, коммуникацию со стейкхолдерами, инструменты, постановку целей и результаты работы.

Что такое kick-off?
Kick-off — это первая встреча тех, кто работает над проектом.
На внутреннем kick-off участники команды знакомятся с проектом и между собой. Менеджер должен рассказать, как будет строиться работа и распределяться обязанности.
Внешний kick-off — это первая встреча с клиентом. Он проходит после встречи с командой. На нем присутствуют менеджер по продажам, проджект с командой и заказчик. На нем заказчик знакомится с командой и узнает, как будет строиться работа. Также на внешнем kick-off оговаривают, насколько клиент планирует контролировать проект.

Основные модели разработки ПО
- Code and fix — модель кодирования и устранения ошибок;
- Waterfall Model — каскадная модель, или «водопад»;

- V-model — V-образная модель, разработка через тестирование;
- Incremental Model — инкрементная модель;
- Iterative Model — итеративная (или итерационная) модель;
- Spiral Model — спиральная модель;
- Chaos model — модель хаоса;
- Prototype Model — прототипная модель

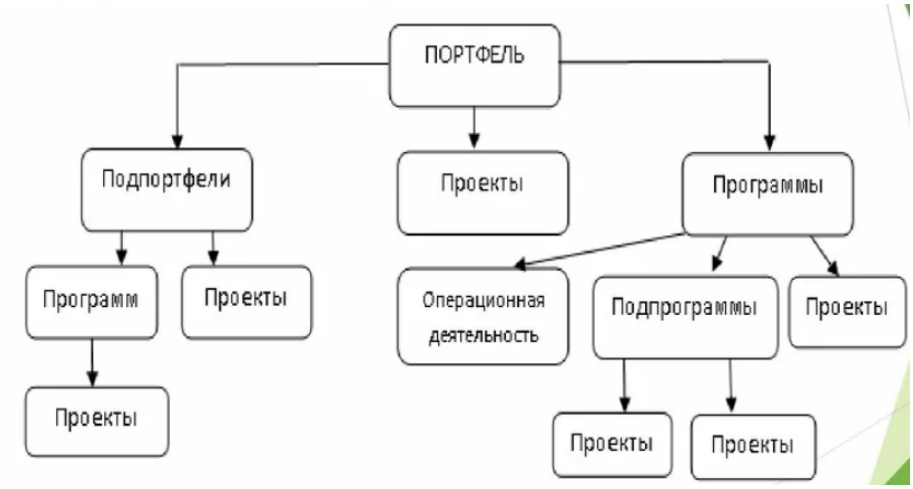
Agile:

- экстремальное программирование (Extreme Programming, XP);
- бережливая разработка программного обеспечения (Lean);
- фреймворк для управления проектами Scrum;
- разработку, управляемую функциональностью (Feature-driven development, FDD);
- разработку через тестирование (Test-driven development, TDD);
- методологию «чистой комнаты» (Cleanroom Software Engineering);
- итеративно-инкрементальный метод разработки (OpenUP);
- методологию разработки Microsoft Solutions Framework (MSF);
- метод разработки динамических систем (Dynamic Systems Development Method, DSDM);
- метод управления разработкой Kanban

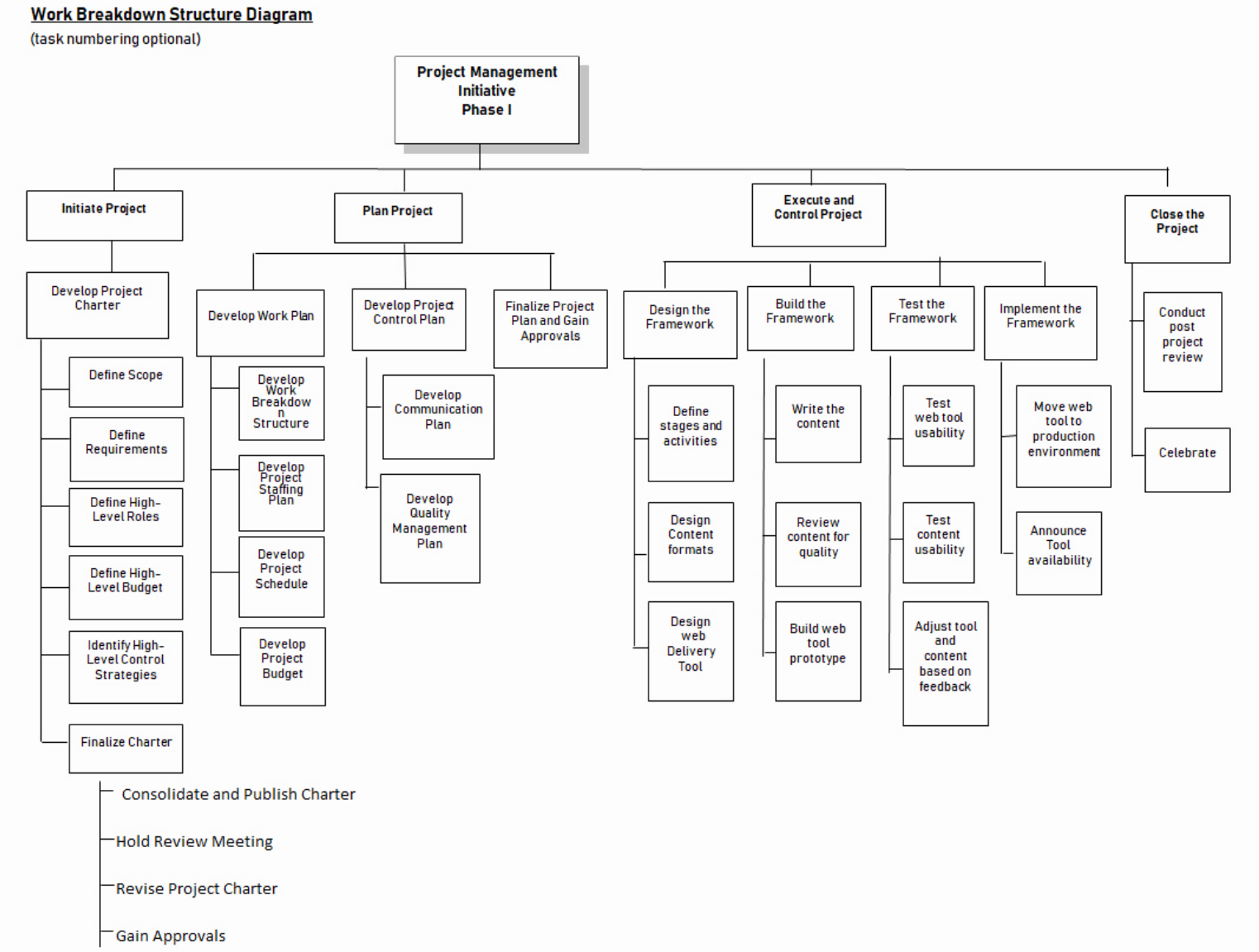
Ссылки:

- https://gb.ru/posts/methodologies
- https://www.osp.ru/cio/2017/08/13053143
- **https://www.teamwork.com/project-management-guide/project-management-methodologies/**

Структура: Портфель, программа, подпрограмма, проекты



Пример WBS:



Пакет работ

Описание пакета работ является логическим продолжением структуры разбивки работ. В структуре разбивки работ (WBS) определяются и отображаются код WBS, название, даты, ответственность и ход выполнения пакетов работ.

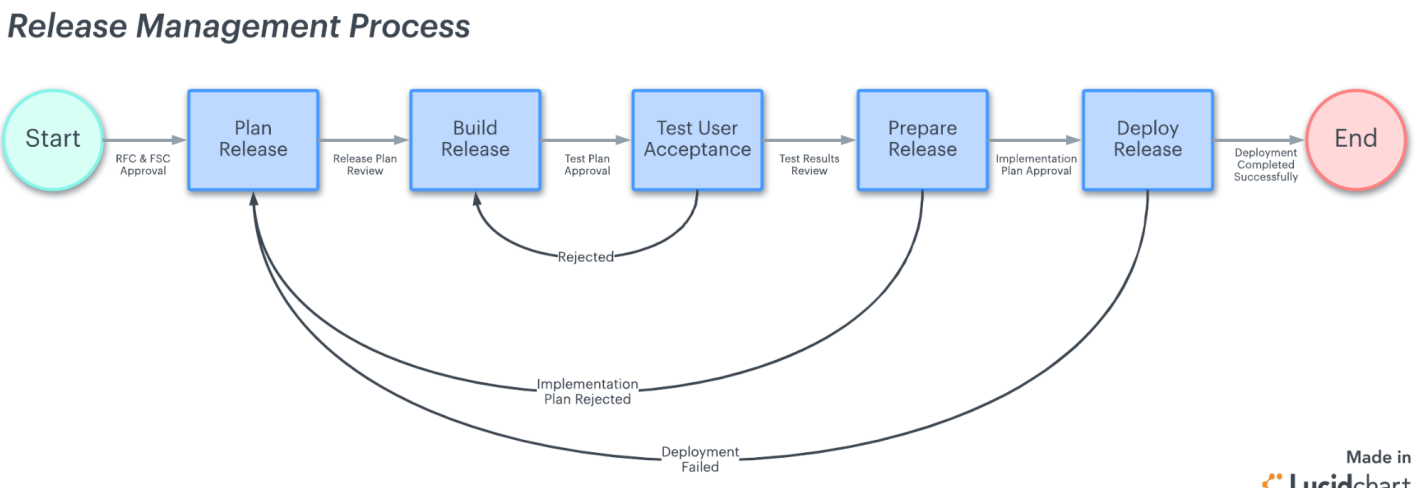
Задача описания пакета работ - точно определить, какие меры должны быть приняты в ходе обработки пакета работ и каким должен быть конечный результат пакета работ.

Любая важная информация может быть определена как содержание описания пакета работ, например, оценка рисков, персонал и бюджетные ресурсы, материалы, оборудование, инфраструктура. Фактическое содержание описания пакета работ должно быть переопределено от проекта к проекту и зависит от некоторых существенных характеристик **(размер, сложность, состав команды, опыт команды).**

Пример описания пакета работ - подробный вариант

WORK PACKAGE DESCRIPTION		
WBS-Code:	Name of Work-Package:	Responsible:
Goals:	• ... • ... • ...!	
Content:	• ... • ... •...	
Results:	• ... • ... • ...	
Resources:	• ... • ...	
Start date:		End date:

Процесс управления релизами



Чем Задача отличается от Фичи?

Фича — это дополнительная функция или особенность продукта. Различают несколько видов:

- "базовая фича", которая является неотъемлемой частью продукта — например, возможность ответить на сообщение в мессенджере или поставить лайк в инстаграме;
- "киллер-фича" — это то чем мы значительно отличаемся от конкурентов;
- "вау-фича" - то, чего наш клиент не ждет, но такая функция может его покорить и сделать нашим клиентов навсегда.

Фича описывает сам функционал и как с ним будет взаимодействовать пользователь, какую ценность он получит.

Задача - про исполнение. Вы решили сделать определенную фичу, и в свою очередь вы уже описываете требования к реализации этой фичи для исполнителя, т.е исполнителю вы уже ставите задачу.

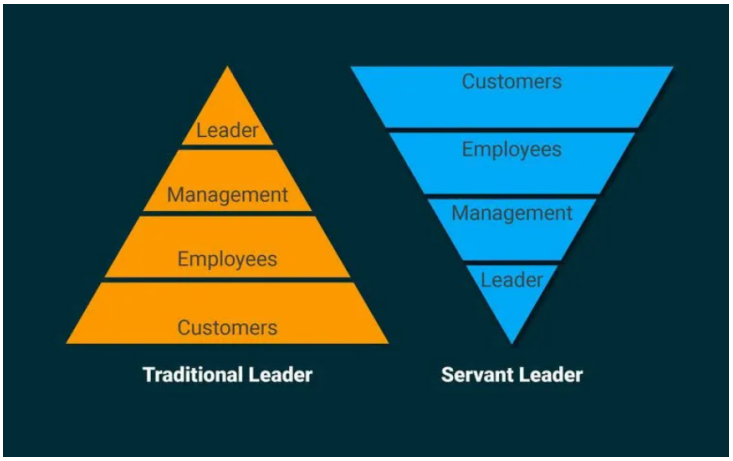
Стили лидерства

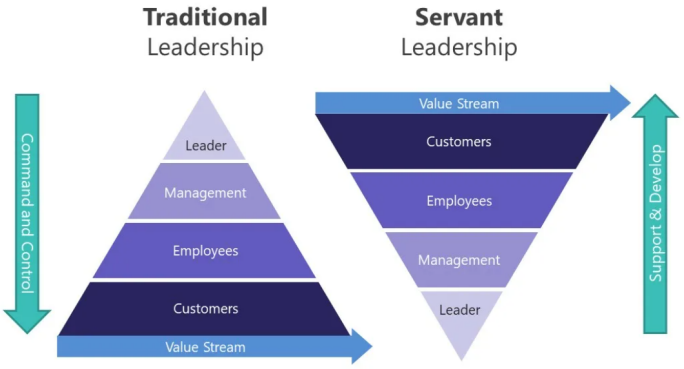
Стиль руководства — манера поведения руководителя по отношению к подчиненным, чтобы оказать на них влияние и побудить к достижению целей организации.

Авторитарный (директивный)	**Либеральный** (попустительский)	**Демократический** (разрешительный)

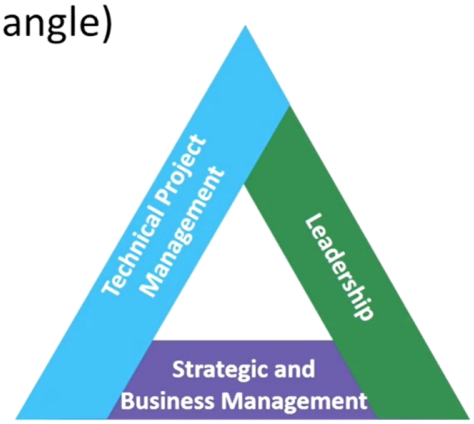
Выбор стиля управления зависит от личных качеств руководителя и подчиненных, а также от конкретной ситуации.

Traditional / Servant Leader





Треугольник талантов менеджера (PM talent triangle)



Стили руководства

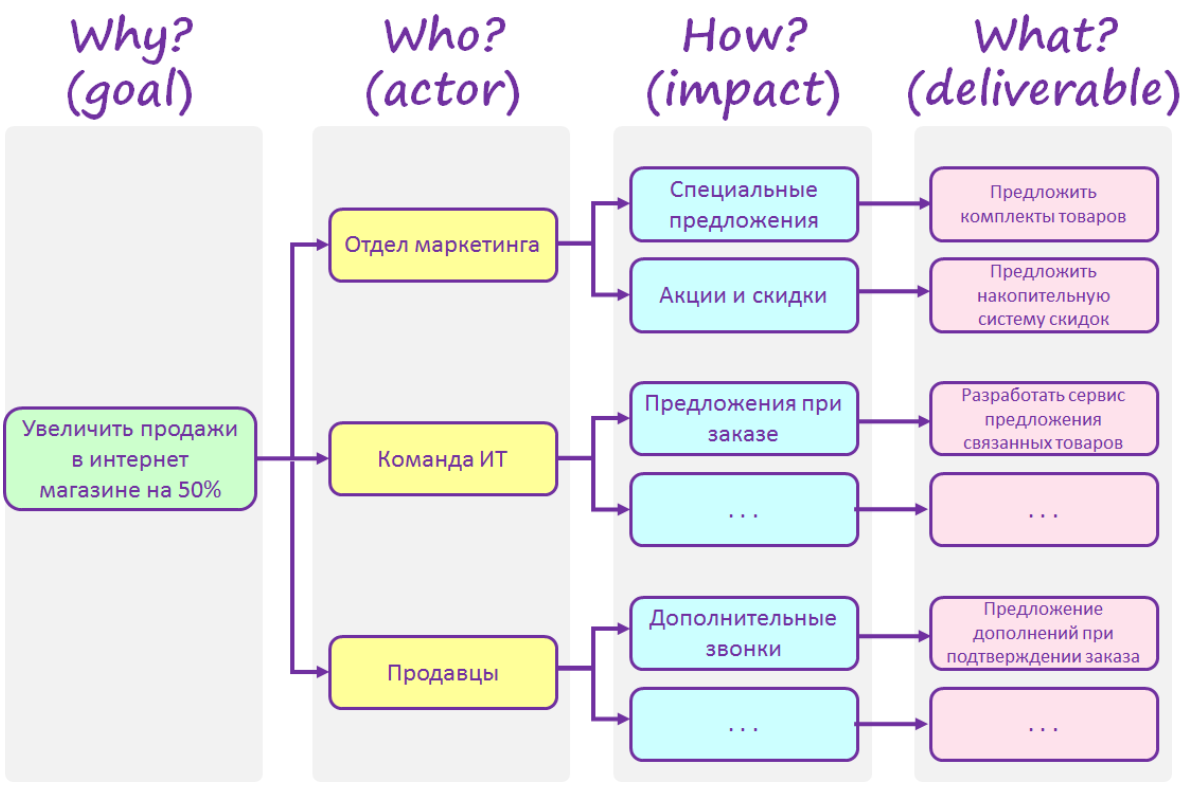
S1	S2	S3	S4
Говорить Указывать Направлять	"Продавать" Объяснять Прояснять Убеждать	Поощрять Участвовать Сотрудничать	Делегировать Наблюдать Отслеживать Завершать

Типология D.I.S.C

Impact Mapping

Это инструмент для декомпозиции цели развития продукта через призму влияния на ее достижение клиентов, инвесторов и других заинтересованных лиц.

- Для формирования бэклога
- Для определения рисков
- Для выявления конфликтов интересов заинтересованных лиц
- Для генерации гипотез по развитию продукта



WSJF — это система оценки задач по формуле: **Cost of Delay / Job size**. Cost of Delay считает деньги, которые бизнес потеряет, если задача не будет выполнена вовремя. Приоритезация задач, когда вокруг все “сложно”.

Формулы для расчета показателей эффективности проекта в тройственном ограничении

EMV

Для расчета риска, EMV (expected monetary value) = P*I
P - probability (вероятность возникновения риска);
I - impact (влияние риска)

EAD

Оценка по трем точкам, EAD (estimated activity duration) = **(P+4M+O)/6**
SD= (P-O)/6
Range= EAD +/-SD*N

Оценка прогресса по методу EVM

- ➔ PV (по плану)
- ➔ EV (по факту)
- ➔ По деньгам (AC)

Отслеживание сроков:
SPI (сроки, индекс, %) = EV/PV
SPI (скроки, отклонения) = EV - PV
Отслеживание:
CPI (бюджет, индекс, %) = EV/AC
CPI (бюджет, отклонения) = EV - AC

Каналы связи
Сколько каналов связи коммуникации у вас есть в команде:
N*(N-1)/2

Юнит-экономика и метрики
https://skillbox.ru/media/management/razbor-metodiki-yunitekonomika/

Как управлять релизом, как выбрать наиболее приоритетрные фичи:
https://habr.com/ru/company/hygger/blog/422131/

- RICE
- ICE

Project Idea	Impact	Confidence	Ease	ICE Score
Billing system	7	1	5	35
Community tab	7	2	8	112
Update receipt design	5	5	3	75

Ссылка для расчетов: https://hygger.io/

Part III. Кейсы и ситуативные вопросы

57 questions for PM with answers

https://www.greycampus.com/blog/project-management/top-thirty-project-management-interview-questions-and-answers

51 questions for PM with answers

ttps://www.simplilearn.com/project-management-interview-questions-and-answers-article

Cases

https://mconsultingprep.com/wp-content/uploads/2021/01/BCG-Online-Case-Example-Official.pdf
Case Study: https://career.guru99.com/

Вопросы

- Расскажи про самый сложный проект?
- Ваши сильные/слабые стороны, как специалиста. Что вам помогает в работе? Чему еще планируете научиться? Где есть пробелы?

- Есть ли какой-то план развития на ближайшие 3-5 лет? К чему стремитесь в карьерном плане?
- Расскажи про самую большую команду?
- Какими проектами ты управлял? Что делал в этих проектах?
- Приходилось ли управлять другими менеджерами?
- Какие курсы и тренинги проходили? Какие книги по управлению читали?
- Опишите свой обычный рабочий день последние полгода.
- Как вы улучшили процесс управления проектами на предыдущем рабочем месте?
- Когда в последний раз вы делегировали обязанности не должным образом и каковы были последствия?

Как можно ускорить проект?

Ответы Junior:

- Отказаться от бесполезных задач
- Делать наиболее важные задачи в начале проекта
- Пересмотреть подход к решению проблемы в проекте
- Исключить задержки в коммуникациях
- Увеличить объем ресурсов за счет сверхурочной оплачиваемой работы

***** Ответ на Middle/Senior будет подкреплен конкретикой, примерами и дополнительными деталями!**

Как мотивировать свою команду разработки на проекте?

- Интересные задачи, понятный и долгосрочный проект;
- Комфортное рабочее место, высокотехнологичный инструментарий;
- Компетентное руководство;
- Корпоративное обучение, перенятие опыта, участие в конференциях, перспективы для развития;
- Комфортная психологическая атмосфера, моральное удовлетворение от результатов работы;
- Участие в принятии решений, признание личного вклада;
- Соцпакет (оплачиваемые отпуска, больничные, ДМС);
- Бесплатные обеды;
- Бесплатный спортзал, интересные выездные корпоративы;
- Возможность удаленной работы, гибкий график.
- Прозрачный и понятный для всех подход,
- Минимум бюрократии
- Минимальная субъективность в оценках
- Открытость, честность и результативность

Что делать, если вы не успеваете к срокам?

- ➔ Можно попробовать сдвинуть сроки
- ➔ Выкинуть часть функционала по согласованию с заказчиками
- ➔ Привлечь людей с других проектов или фрилансеров (спорный кейс - исполнителей можно дополнять только в том случае, если он успеет быстро погрузиться и действительно принести вэлью проекту и помочь)

Дальше, обычно, ситуация конкретизируется, появляются дополнительные ограничения: «срок сдвигать нельзя», «есть очень важные фичи, а есть не очень» и т.п. Параллельно интересуется «как вы будете выкидывать фичи?», «будете предупреждать сразу или тянуть до последнего?» и т.п.

Что будете делать, если не успеваете протестировать?

-Надо посадить программистов тестировать - это подход из методологии Kanban.
-Садиться и самому тестировать функционал.
-Поговорить с заказчиком, выбрать самые важные, которые можно успеть.
-Подключить людей с другого проекта.
Заказчик хочет получить релиз через месяц, а разработчики оценили работу в 6 месяцев.
Надо посмотреть, что запланировали разработчики. Возможно, они хотят сделать сложную архитектуру, которая в данном случае абсолютно не нужна, а время съест много.

Почему уходите из прошлой компании?
Есть стандартный вариант ответа «На этом месте работы я достиг своего потолка, дальнейшего развития не предвидится, а я хочу двигаться дальше». Не стоит говорить плохо о старом месте работы.

Чем хотите заниматься? Какая работа вам интересна?
Надо придумать ответ заранее, желательно с учетом специфики вакансии. Например, мне нравится делать востребованные продукты и релизить их в срок.

Человек хочет увольняться, но не хочет передавать дела. Как его стимулировать?
Дополнительное условие – материальная стимуляция вам недоступна, то есть вы не можете выписать ему премию или оставить в подарок корпоративную страховку на год. Надо пообещать увольняющемуся хорошую рекомендацию для его будущего работодателя.

Сегодня вечером вы уезжаете в отпуск, а тут все сломалось.
Например: «Когда я куда-либо уезжаю, я всегда оставляю вместо себя замещающего и эту проблему будет решать он».
Какое соотношение тестировщиков и программистов вам кажется оптимальным? Это зависит от проекта, но обычно достаточно 2 разработчика на 1 тестировщика, либо в случае потребности в очень высоком качестве 1 к 1.
Что такое реестр рисков?
Это документ, содержащий все выявленные риски проекта, включая список обязательных и потенциальных действий.
Что такое юзабилити?
Это оценка удобства взаимодействие пользователя с интерфейсом. Например, насколько комфортно работать с сайтом или приложением: все ли понятно, легко ли найти нужные данные и ориентироваться в структуре.

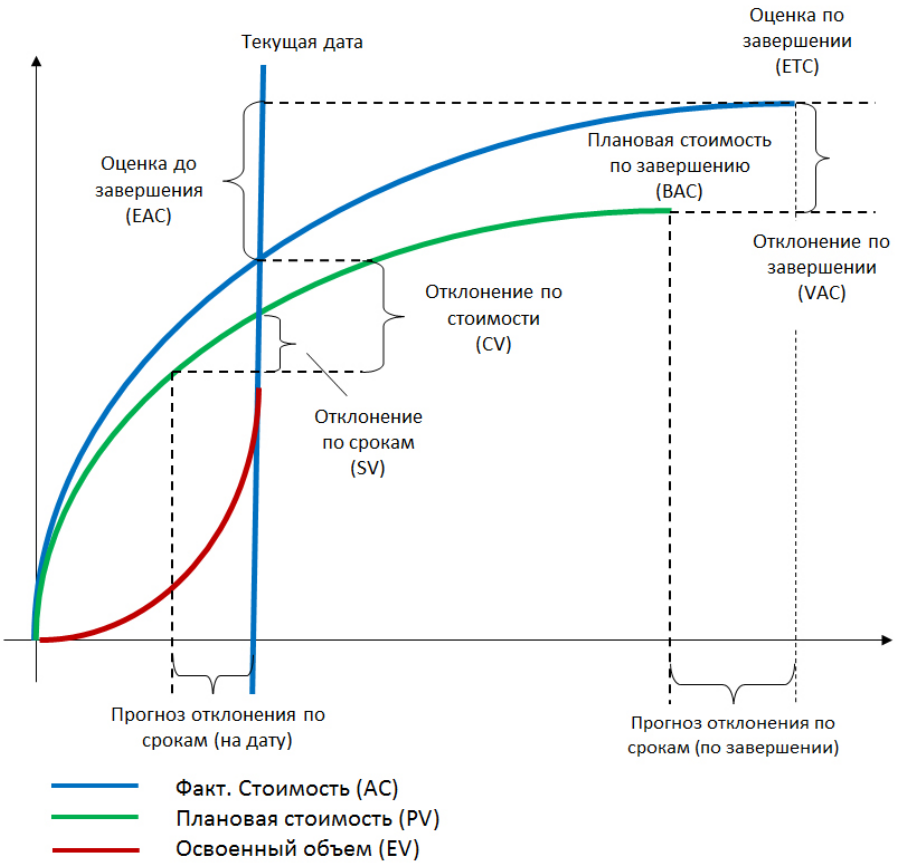
Планирование:
- Как ты планируешь проект?
- Тебе нужно запланировать проект у которого есть жесткий дедлайн и скоуп, который обязательно нужно успеть. Как ты будешь планировать?
- Как ты построишь процесс оценки бэклога?
- Как ты конвертируешь оценку работ в план?
- Как вычислишь длительность проекта?
- Какие фазы/виды работ ты включишь в свой проект? (не только разработка тестирование)

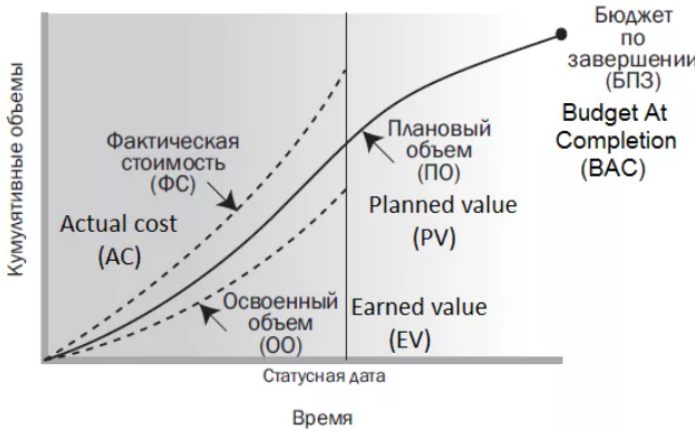
Как ты будешь готовиться к релизу? (приемка, план развертывания, стабилизация)

- Планирование. Весь путь продукта начинается с его планирования и стратегии, а планирование релиза позволяет рассчитывать и определять количество спринтов или итераций. На этом этапе происходит начальная приоритезация, предварительная оценка затрат и анализ взаимозависимостей. К планированию привлекаются аналитики и заказчик. Планирование включает ожидания серьезных изменений в продукте, которые могут быть отражены в дорожной карте
- Согласование. На этом этапе получается подтверждение ресурсов от исполнителей и заказчиков, происходит оценка бюджета релиза. Содержание релиза финально согласуется со всеми заинтересованными сторонами.
- Документирование. Этот процесс закрепления и систематизации всех процедур, где отражены, например, последние данные о новых фичах.
- Коммуникация и поддержка. Для каждого менеджера продукта очень важно не только успешно выпустить продукт или обновление, но и обеспечить налаженное взаимодействие с командой поддержки.
- Статусы готовности. Статус релиза отражается в актуальном состоянии вашего плана. Актуализация статуса помогает снизить риски и обеспечить связь с заинтересованными сторонами.
- Тестирование/Корректировки. В процессе управления релизом не обойтись без регулярных проверок и корректировок, которые помогают достигать порядка и достижения финальной цели.
- Развертывание. На этом этапе изменения передаются в эксплуатацию. Выходит новая версия продукта или сам конечный продукт

Проект уже идет, как ты поймешь успеваешь к релизу или нет? Какие метрики?
Метрики проекта по методике Google:
Happiness (Счастье)
— пользовательское удовлетворение;
— ощущение, что продуктом легко пользоваться;
— net promoter score.
Engagement (Вовлечение)
— количество визитов пользователя в неделю;
— количество фото, загружаемых юзером в день;
— количество лайков и шэров.
Adoption (Принятие)
— обновления до новой версии;
— созданные пользователем подписки;
— покупки сделанные новыми пользователями в приложении.
Retention (Возвращаемость)
— количество пользователей, остающихся активными с течением времени
— churn
— повторные покупки
Task Success (Успех ключевых задач)
— успешные поиски;
— время загрузки фотографии;
— полностью заполненный пользователем профиль

!!! Метод освоенного объема (Earned Value Technique, Earned Value Management) — Система методик, объединенных под общим названием, использующихся для измерения и контроля эффективности выполнения проектов. Метод основан на использовании ряда числовых показателей, рассчитываемых по ходу проекта.
Презентация: https://ppt-online.org/387238





- ПС – плановая стоимость: стоимость запланированных работ (BCWS);
- ОО – освоенный объём: плановая стоимость выполненных работ (BCWP);
- ФС – фактическая стоимость: реальная стоимость выполненных работ (ACWP).
- CV – отклонение по стоимости;
- SV – отклонение по срокам;
- BAC – плановые затраты на проект;
- EAC – оценка реальных затрат на проект.

Количественные метрики релиза:
Чтобы понять, как у вас дела, достаточно мерить скорость **vs** качество вашей работы

- Скорость команды или составная метрика: пользовательские истории, сделанные в течение этой итерации/не завершенные, но запланированные/перенесенные в следующую итерацию.
- Количество дефектов, найденных за итерацию/vs пробравшихся в итоге на продакшн
- Распределение дефектов по типам/компонентам и источникам

Качественные метрики релиза:

- Отношение количества сделанных пользовательский историй к общему числу историй в проекте
- Отклонение от запланированной даты релиза
- Технический долг

Проектные метрики:

- срок реализации (допустимые отклонения от базового плана)
- выполнение бюджета (допустимые отклонения)
- плановый и фактический срок реализации
- причины сдвигов по времени.

Важно: если проект велся корректно, то любое смещение срока реализации должно было фиксироваться запросом на изменение. Подобные решения PM не принимает — он может только все подготовить, предложить опции дальнейших действий, но последнее слово остается за спонсором.

- отчет по расходам (сколько планировали и сколько в итоге потратили)
- риски (нужно указать, какие контрмеры были эффективны, а какие — нет)

Планирование:

- Как ты будешь измерять прогресс, если не весь бэклог декомпозирован?
- Какие показатели ты включишь в отчет? А на фазе приемки?
- Клиент часто уточняет скоуп проекта, что ты будешь делать с этим?
- Как ты построишь коммуникацию с клиентом при утверждении новых элементов скоупа проекта?
- Как отличить новый скоуп от простой детализации требований?
- Что делать если изменений слишком много?

Финансы:

- Что такое бюджет проекта?
- Как рассчитать прогноз выручки? (что такое эффективная реализация персонала?)

- От чего зависят затраты проекта? (какие затраты включаются в овертаймы, а какие нет?)
- Как повысить прибыльность проекта?
- Какие риски с деливери несут манипуляции с прибыльностью?
- Какие пункты контракта влияют на прибыльность?

Кризисы:

- Вы не успели к дедлайну, ваши действия?
- Серьезные дефекты на продакшене, твои действия?
- Эскалация по любому другому вопросу, что будешь делать?

Управление ПМом:

- Как оценить качество работы ПМа?
- Клиент или команда эскалируют на подчиненного ПМа, твои действия?
- Как будешь собеседовать ПМа?
- Приходилось ли набирать людей?
- Как проводили собеседования?
- Как отбирали кандидатов?
- Приходилось ли увольнять?
- Как происходил пересмотр зарплаты?

Что такое юзер стори?
Это короткое описание фичи в формате ответа на вопрос: «Кто, что и зачем делает?». Это объяснение фичи с точки зрения конечного пользователя.
Например, для Google Maps это будет так: «Как пользователь, я хочу ввести город, улицу и номер дома в строку поиска, чтобы найти расположение нужного дома на карте»

Что такое критерии приемки проекта?
Это условия, при которых юзер стори можно считать готовой и полностью работающей. Они нужны, чтобы команда сделали именно то, что просил заказчик.
****Для Google Maps критерии будут выглядеть так:** На странице есть строка и кнопка поиска. Поиск выполняется, если пользователь вводит город, улицу и номер дома. Когда поиск выполнен, на карте появляется точка, показывающая расположение адреса. Это похоже на описание, только оно включает любые детали юзер стори. А без критериев фича не заработает, или заработает, но не выполнит сценарий юзер стори.

Top 7 Super Fast Cases Before the Interview

Кейс 1
Ваш проект должен получить достаточно большие инвестиции, а главный разработчик Антон сообщает, что намерен покинуть проект и компанию. При разговоре с ним вы узнаете, что у разработчика три проблемы:
Устал от однообразных задач.
Не устраивает ЗП. Конфликт с коллегой.
Ваши действия, чтобы убедить Антона остаться?
Кейс 2
Вы закончили эстимейт проекта, осталось согласовать его с клиентом. Однако фичу, которую вы оценили в 40 часов работы, клиент требует сделать за 25. Заказчик

аргументирует это тем, что его знакомому выполнили похожую таску за такой же срок (25 часов).

Как ускорить разработку?

Кейс 3

Вы недавно начали новый проект. Команда хорошо справляется, однако у разработчика-джуна Александра есть проблема. Ему постоянно не хватает времени. Из-за чего страдает общая производительность.

Что вы будете делать?

Кейс 4

Вы работаете в компании около 6 мес. На данный момент у вас три проекта по разработке различных приложений и четвертый – поддержка готового продукта. В определенный момент вы отдаете себе отчет, что не успеваете, из-за чего страдает качество разработки.

Что вы будете делать, чтобы улучшить ситуацию?

Кейс 5

У вас таска: провести код-ревью базы для клиента. Однако, в вашем распоряжении всего один джун. Остальные команды загружены.

Ваши действия?

Кейс 6

Вы сопровождали проект три месяца и через два дня у вас первый релиз. Трудно избавиться от волнения: в продукте много фич и возможно какая-то из них сломается.

Как будете готовиться к релизу?

Кейс 7

Вы начали новый проект. К этому моменту с командой было оговорено, что каждый день сотрудники должны трекать время в Jira и подробно описывать, на что они его потратили.

Через неделю вы замечаете, что разработчик Юлия не делает этого. Когда вы подняли этот вопрос на дейли, она ответила, что не видит потребности в трекинге времени.

Ваши действия?

Part IV. Понятийный уровень. Основы программирования

Логика и алгоритмы
Типы данных и их виды (строка, целое число, булевое значение)
Основные структуры данных
Функции, операции
Условия (else, if, elif, else if, switch)
Циклы (while, for, for in, break)
События
Прокси сервер, reverse прокси
API
HTTP запрос
Виды протоколов, уровни
Конфиги, логи
Сниппет
Сокеты, web socket

Архитектура Web приложений:
Понятие URL и документа
Виды документов: HTML, CSS, JS, JSON
Абсолютные и относительные URL
Правила разрешения URL-ов
Гиперссылки
Клиент-серверная архитектура

WEB-клиенты:
Консольные утилиты. Telnet
Библиотеки в ЯП. libcurl, urllib
Браузер
Фичи браузера: куки, сессии, Referer
Основной сценарий работы
Классические web приложения: SPA, MPA, PWA

Язык разметки HTML:
Основы разметки
DOCTYPE - сокращенно DTD, представляет собой набор правил, используемых для объявления документов определенного типа.
Блочные и строчные теги
Таблицы и списки
Гиперссылки и формы

CSS (язык описания стилей):
Синтаксис
Селекторы
Псевдоклассы и псевдоэлементы
Приоритеты стилей и каскадирование
Основные стили
Позиционирование

Box-model - это алгоритм расчёта размеров каждого отдельного элемента на странице, которым браузеры пользуются при отрисовке.

Сетевые протоколы
DNS
Домены и зоны, делегирование
Рекурсивные запросы
TCP
Понятие порта
TCP handshake
TCP клиент и сервер
TLS

Протокол HTTP
Назначение и ключевые особенности
Синтаксис запроса и ответа
Методы запросов
HTTP заголовки
Коды ответа

Web-сервера
Файлы и процессы сервера
Внутренняя архитектура сервера
Примеры конфигурации
Понятие location
Методы обработки сетевых соединений

Архитектура frontend - backend
Задачи frontend сервера
Reverse proxy
Проксирование запросов
Application сервера
Протоколы CGI, FastCGI, WSGI

MVC фреймворки
Компоненты MVC
Конфигурация проектов
Маршрутизация URL
Обратная маршрутизация URL

Работа с СУБД
Реляционная модель данных
Проектирование баз данных

Обработка форм
GET и POST формы
Общий сценарий обработки
Перенаправления в HTTP
Описание форм в Django
CSRF

Сессии и Авторизация

Basic HTTP Authorization
Механизм Cookie
Установка и получение cookie в HTTP
Авторизация с использованием cookie

Part V. Общие темы для “погуглить” самостоятельно

Компиляторы, интерпретаторы, ассемблеры
Программы и языки программирования
Основы алгоритмов для руководителя IT-проекта

Основы сетей и их протоколов
Клиент-серверная архитектура
Браузер и другие виды клиентов
Общие принципы построения архитектуры проектов

Этапы создания приложения
Типы задач разработчиков
Алгоритм оценки приложения
Декомпозиция

Начало проекта
Планирование проекта
Управление требованиями к ПО
Команда, проект, распределение ролей в команде
Бюджет проекта
Управление рисками
Коммуникация на проекте
Управление качеством
Основы управления процессом тестирования на проекте
Реализация проекта. Мониторинг и контроль. Проектная отчетность и создание документации
Проект, его структура и необходимые документы
Продукт vs Проект
Устав, роли и разделение труда
Содержание проекта
Методы сбора требований
Как и когда собирать требования проекта
Категории требований
Требования к требованиям
Утверждение требований
Иерархическая структура работ
Что такое ИСР (WBS)
Форматы ИСР
Пакеты работ
Стандарты создания ИСР
Как и когда создавать ИСР
Оценка проекта

Процесс оценки
Методы оценки проекта
Виды оценок
Управление качеством проекта
Планирование управления качеством
Обеспечение качества
Контроль качества
Сроки и расписание проекта
Глоссарий расписания проекта
Диаграмма Ганта
Метод критического пути
Риски
Идентификация рисков
Качественный анализ рисков
Количественный анализ рисков
Планирование реагирования на риски
Контроль рисков
Стоимость и бюджет проекта
Оценка стоимости
Определение бюджета
Контроль стоимости
Управление изменениями
Прогнозирование изменений
Методологии управления изменениями
Оформление change requests
Работа с возражениями
Адаптация в условиях VUCA (Volatility, Uncertainty, Complexity, Ambiguity).
Управление заинтересованными сторонами проекта
Определение заинтересованных сторон
Управление вовлечением заинтересованных сторон
Планирование управления заинтересованными сторонами
Прогресс проекта
Мониторинг и управление работами проекта
Milestones или вехи проекта
Отчетность по исполнению
Администрирование контрактов
Завершение проекта
Критерии завершенности и метрики успешности проекта
Exit check-list проекта
Ретроспектива
Инструменты руководителя проекта
Календарь
Jira, Trello, Monday, Confluence, Nuclino
Гибкие методологии разработки ПО
Agile, Scrum
Kanban
XP
LEAN
Управление командой
Что такое команда проекта
Устав команды
Как набирать команду проекта

Постановка и контроль задач
Как и какие задачи нужно ставить
SMART
Делегирование
Управление временем
Основы коммуникации
Матрица компетенций проектного менеджера
Коммуникации с клиентом
Ведение переговоров и управление конфликтами
Передача проекта
Управление третьими сторонами
Деловая переписка
Фасилитация
Управление обсуждением
Кто такой фасилитатор
Что и как контролировать в обсуждении
Переговоры и конфликты
Что такое конфликты и зачем нужны переговоры
Агрессия
Техники отказа
Деловая коммуникация
Правила деловой переписки
Пирамида деловой переписки
Форматирование и структура письма
Командное развитие
Коучинг в проектном управлении
Реализация стратегии
Персональная мотивация и развитие
Что дальше?
Сертификаты (PMI, Scrum Alliance)
Networking и как его построить

Part VI. Полезные ресурсы

Курсы и статьи:

1. https://knowledgemap.pm/ru/ - ресурс, для подготовки PMP сертификации
2. https://redcamp.super.site/ - курс для ПМ (методологии, основы IT)
3. https://technically.dev/
4. Статья Project Management 101: The Complete Guide to Agile, Kanban, Scrum and Beyond: https://zapier.com/blog/project-management-systems/
5. Статья The Secret History of Agile Innovation: https://hbr.org/2016/04/the-secret-history-of-agile-innovation
6. A Brief History of Lean: https://www.lean.org/explore-lean/a-brief-history-of-lean/
7. History of PMI: https://www.pmi.org/about/learn-about-pmi/history-of-pmi
8. Foundation of Project Management - Google
9. Project Management - Yalantis
10. Project Management - Skills Up

Кейсы для практики:
https://upravlenie-proektami.ru/kak-peredavat-dela-na-vremya-otpuska-versiya-2-0-peredacha-del
Видео-курс Ивана Селиховкина: PMI/Scrum/Kanban:
https://www.youtube.com/playlist?list=PL4b81pr3uYis4VvkRNpGFuX-XwxOCeq9t
О Scrum, PMI и Kanban, выжимка из видео-курса и книг И. Селиховкина:
https://docs.google.com/document/d/1wo1u5nYHHE42nz2kq5Kygx91cTZX50Fl8wYE3AXE61M/edit#heading=h.m6k4pxw0yygh
Notion подборка материалов для ПМа от меня :)
https://alluring-stealer-b66.notion.site/IT-Project-Manager-7a62087bf6fc487e9c68d2272d6e42a3
Полезный документ с примером Proposal Example:
https://docs.google.com/document/d/18aGd73K4jpRUDfF4RHyJfsTD3rFC-5W3NgLqmBD6oPA/edit
Карта развития ПМа, компетенции:
https://docs.google.com/spreadsheets/d/1Z2Sh34U3PpvRJPZl9srdT7yvLArkisTLh08hjJVELWw/edit#gid=0
PM Glossary: https://web.telegram.org/cc40b493-fe0e-4d1e-b2d0-97b4bfcba694

!!! vc.ru большие подборки для проджектов и продактов:

1. https://vc.ru/hr/331944-neskuchno-zhit-podborka-resursov-dlya-prodzhekt-menedzherov
2. https://vc.ru/life/425233-podborka-poleznyh-resursov-dlya-prodzhekt-menedzherov-glava-i-kursy-i-sayty
3. https://vc.ru/hr/248319-bolshaya-podborka-dlya-menedzhera-digital-proektov-100-poleznyh-materialov-dlya-obucheniya-i-rosta
4. https://vc.ru/hr/437499-bolshaya-podborka-kanalov-dlya-prodakt-menedzherov

Книги:

1. **Марк Розин “Успех без стратегии”** - качественный обзор методов стратегического управления по разным сегментам деятельности, который автор, консультант и управляющий партнер крупной компании ЭКОПСИ внедрял в различных корпорациях
2. **Джо Оуэн “Как управлять людьми” -** практическое руководство для менеджеров всех уровней, как менеджерам развивать в себе качества и типы поведения, ведущие к успеху.
3. **Джим Кемп “Сначала скажите “Нет”**
4. **Гэвин Кеннеди “Договориться можно обо всем”** - о современной тактике и стратегии ведения переговоров с секретами и хитростями
5. **Фергус О’Коннел “Как успешно руководить проектами. Серебряная пуля”.**
6. **Генри Форд “Моя жизнь, мои достижения”**
7. **Ли Якокка “Карьера менеджера” -** о том, как Ли Якокка благодаря своему труду и упорству, достиг высот в бизнесе, став одним из самых богатых и влиятельных людей Америки
8. **Георгий Щедровицкий “Оргуправленческое мышление: идеология, методология, технология”**
9. **Дмитрий Ершов Без ТЗ: Как запустить сервис и ничего не упустить -** про будни менеджера продукта дополняется врезками с личным практическим опытом из собственной карьеры
10. **Аутсорсинг разработки цифровых продуктов**

11. **Дэн Кеннеди “Жесткий менеджмент. Заставьте людей работать на результат”.**
12. **Алексей Каптерев “Мастерство презентации. Как создавать презентации, которые могут изменить мир”**
13. **Павел Безручко “Без воды. Как писать предложения и отчеты для первых лиц”**
14. **Кали Весслер и Джоди Томпсон “Офис в стиле фанк”**
15. **Бен Хоровиц “Легко не будет”**
16. **«Вальсируя с медведями» Том ДеМарко**
17. **«Человеческий фактор» Том ДеМарко**
18. **“Человеческий фактор” Том ДеМарко**
19. **“Deadline” Том ДеМарко**
20. **«Искусство управления IT проектами» Скотт Беркун**
21. **«Цель. Процесс непрерывного совершенствования» Ильяху Голдрат**
22. **Джефф Сазерленд. “Scrum. Революционный метод управления проектами” -** о том, как организовать слаженную командную работу.
23. **Дэвид Андерсон. “Канбан: альтернативный путь в Agile”**
24. **Роб Фитцпатрик “Спроси маму”**
25. **“Проект Феникс” Джин Ким**
26. **“Антихрупкость. Как извлечь выгоду из хаоса” Нассим Талеб**
27. **“Чёрный лебедь. Под знаком непредсказуемости” Нассим Талеб**
28. **Рэй Далио. “Принципы”**
29. **“Как пасти котов. Наставление для программистов, руководящих другими программистами” Дж. Ханк Рейнвотер**
30. **“Клиенты на всю жизнь” Карл Сьюэлл и Пол Браун**
31. **“Alibaba и умный бизнес будущего” Цзэн Мин -** о бизнес-модели компании Alibaba Group, о принципах и особенностях смарт-бизнеса.
32. Бизнес без MBA под редакцией М. Ильяхова
33. Мифический человеко-месяц
34. How To Speak Tech

Telegram-каналы по проектному менеджменту:

1. https://t.me/pro_pm
2. https://t.me/temno
3. https://t.me/psilonsk
4. https://t.me/pm_and_it
5. https://t.me/pm_god
6. https://t.me/OpenManagement
7. https://t.me/ekstrapolyatsiya_menedzhmenta
8. https://t.me/selihovkin
9. https://t.me/PM_Events
10. https://t.me/its_capitan
11. https://t.me/pm_sovet

Part VII. Блокчейн технологии

- **Блокчейн** - распределенная база данных, которая содержит информацию обо всех транзакциях, проведенных участниками системы
- **Блок** -набор данных, являющийся звеном цепи блокчейна
- **FinTech** - это предоставление финансовых услуг и сервисов с использованием инновационных технологий, таких как «большие данные» (Big Data), искусственный интеллект и машинное обучение, роботизация, блокчейн, облачные технологии, биометрия и других
- **Бридж** - это звено между двумя блокчейнами, которое помогает перемещать токены из одной сети в другую
- **Эндпоинты** - адреса в сети, URI: последовательность символов, идентифицирующая абстрактный или физический ресурс. Uniform Resource Identifier — унифицированный идентификатор ресурса. Иногда конечную точку, или URI называют путем (path) — путем до ресурса
- **Конфиги** - совокупность консольных команд; текстовый файл с расширением .cfg и загружаемый перед игрой
- **Логи** - это файл с информацией о действиях ПО или пользователей
- **«Биржевой стакан» (order book)** — это список всех ордеров, как на покупку, так и на продажу, на отдельно взятой платформе
- **API** - (программный интерфейс приложения) — описание способов (набор классов, процедур, функций, структур), которыми одна компьютерная программа может взаимодействовать с другой программой.
- **Маркетмейкеры (MM)** - фирма - брокер / дилер, берёт на себя риск приобретения и хранения на своих счетах ценных бумаг определённого эмитента с целью организации их продаж
- **Yield Farming** - движение, участники которого стремятся выжать из своих инвестиций максимальную прибыль за счет DeFi-протоколов
- **Frontend (FE)** - клиентская сторона пользовательского интерфейса к программно-аппаратной части сервиса
- **Тестнет** - альтернативная цепочка криптовалюты исключительно для разработчиков
- **Мейннет** - реальная сеть, в которой происходят все реальные транзакции.
- **Сервер** - выделенный или специализированный компьютер для выполнения сервисного программного обеспечения
- **Solana** – блокчейн-сеть для быстрых транзакций с высокой пропускной способностью, которая использует уникальный способ упорядочивания транзакций для повышения скорости
- **Solidity** -язык программирования для написания смарт-контрактов
- **TRON**- это децентрализованная операционная система на основе блокчейна с открытым исходным кодом, функциональностью смарт-контрактов, принципами подтверждения ставки в качестве алгоритма консенсуса и собственной криптовалютой, известной как Tronix (TRX)
- **Шардинг** - это разделение сети блокчейна на индивидуальные сегменты (шарды). Каждый шард содержит уникальный набор смарт-контрактов и балансов счетов
- **Каналы состояния** - это решение второго уровня, позволяющее участникам совершать бесконечное множество приватных транзакций вне основного блокчейна

- **Сайдчейны** — технология, позволяющие токенам и другим цифровым активам одного блокчейна безопасным образом использоваться в другом блокчейне и затем (в случае необходимости) быть возвращенными в оригинальный блокчейн
- **SDK** (software development kit, «комплект для разработки программного обеспечения») — набор средств разработки, позволяющий специалистам по программному обеспечению создавать приложения для определённого пакета программ, программного обеспечения базовых средств разработки, аппаратной платформы, компьютерной системы, игровых консолей, операционных систем и прочих платформ.
- **Cosmos SDK** (Cosmos Software Development Kit) — платформа и набор инструментов для программистов, облегчающие написание кода
- **Форк** — это использование программного кода одной системы для разработки другого проекта
- **Коин** – это виртуальная монета, именно она и считается полноценной криптовалютой, валюта блокчейна
- **Токен** - это цифровой актив, который выпускается на определенном блокчейне компанией с целью использования как главного платежного инструмента
- **BTC** -цифровая или криптовалюта, которая работает исключительно в интернете. Ее относят к новому поколению, поскольку выпуск и курс никто не контролирует, а производство осуществляется на компьютерах пользователей, которые подключились к системе
- **Альткоин** - это альтернативные криптовалюты, разработанные ради того, чтобы побороть монополизацию биткоина, а также для решения проблемы масштабируемости, децентрализации и безопасности, в которых погряз биткоин
- **Convex** - это протокол для оптимизации урожайности за счет упрощения процесса повышения кривой. Convex позволяет поставщикам ликвидности Curve получать комиссию за торговлю и требовать увеличения CRV без привязки к собственной CRV
- **Uniswap** – это децентрализованный сетевой торговый протокол на Ethereum, который использует пулы ликвидности и форму автоматических маркет-мейкеров (AMM) вместо книг заказов
- **DeFi** — это финансовые инструменты в виде сервисов и приложений, созданных на блокчейне. Главная задача децентрализованных финансов стать альтернативой банковскому сектору и заменить традиционные технологии нынешней финансовой системы протоколами с открытым исходным кодом
- **PancakeSwap** - это децентрализованная биржа (DEX), работающая в сети Binance Smart Chain, что делает ее первой в своем роде. На ней имеется множество функций, которые позволяют зарабатывать токены платформы (CAKE) и других проектов, а также выигрывать денежные призы
- **Пулы ликвидности** — это пулы токенов, заблокированных в смарт-контрактах, которые обеспечивают ликвидность на децентрализованных биржах, с целью нивелировать проблемы, вызванные неликвидностью, типичной для таких систем
- **ASIC** - Интегральная схема, разработанная специально для выполнения определённой задачи, в данном случае для майнинга, с чем справляется гораздо более эффективно, чем GPU или FPGA

- **DApp**- децентрализованное приложение. Набор смарт-контрактов. Данные хранятся на блокчейне, присутствует система стимулирования в виде токенов, работа подобного приложения осуществляется в автономном режиме
- **DAO** - децентрализованная автономная организация. Сложная система, функционирующая на основании запрограммированных правил.
- **EVM** - виртуальная машина Эфириума. Каждая нода блокчейна использует EVM для достижения глобального консенсуса
- **Нода** - это любой компьютер, подключенный к блокчейн-сети. Ноды децентрализованной сети контактируют посредством P2P-протоколов для обмена информацией о блоках и транзакциях
- **Стейкинг** — это способ пассивного заработка, при котором пользователи хранят монеты на алгоритме Proof of Stake (PoS) и обеспечивают работоспособность блокчейна
- **Минт** - это создание в различных криптовалютах новых блоков в блокчейне на основе подтверждения доли владения с возможностью получить вознаграждение в форме новых единиц и комиссионных сборов
- **Майнинг** - название процесса нахождения новых блоков было выбрано не случайно: невольно возникают ассоциации с процессом добычи золота
- **Приватный ключ** -строка символов, которая обеспечивает вам доступ к токенам, хранящимся на конкретном кошельке.
- **Публичный адрес** - представляет собой криптографический хэш публичного ключа. Аналог адреса электронной почты, который вы можете смело отправлять других людям.
- **Оракулы** - поставщики данных из реального мира в блокчейн
- **KYC** — Know Your Customer или Know Your Client (Знай своего клиента). Это принцип деятельности финансовых институтов (банков, бирж, букмекерских контор, инвестиционных и паевых фондов), обязывающий их идентифицировать личность контрагента, прежде чем проводить финансовую операцию
- **AML** — Anti-Money Laundering (противодействие отмыванию денег). Если быть совсем точным, то аббревиатура должна была быть длиннее, AML CFT CWMDF — anti-money laundering and counter-terrorist financing and counter-weapons of mass destruction financing (противодействие отмыванию денег, полученных преступным путем, противодействие финансированию терроризма и финансированию создания оружия массового уничтожения)
- **Multi-Signature** (множественная подпись) - является дополнительным средством безопасности: для проведения транзакции требуется как ключ отправителя, так и ключи нескольких посторонних участников.
- **Атака большинства a.k.a. Атака 51%** - умышленно нанесенный блокчейну вред со стороны группы людей, обладающих более чем 50% всех вычислительных мощностей
- **Хэшрейт** -совокупная вычислительная мощность компьютеров в сети, поддерживающих работоспособность блокчейна.
- **Халвинг** - процесс уменьшения скорости генерирования новых единиц криптовалюты. В частности, это относится к периодически происходящему событию, последствием которого является уменьшение награды майнеров за успешно добытый блок.

Part VIII. Основы и терминология WEB и Digital

- Digital - интернет-пространства для распространения или обмена информацией различными способами.
- Seo - Это большой комплекс работ по увеличению видимости сайта по поисковым запросам в выдаче поисковых систем.
- CMS - приложение для управления сайтом и/или его содержимым. Позволяет пользователю, который не владеет языками программирования или разметкой HTML, редактировать содержимое сайта. CMS расшифровывается как **C**ontent **M**anagement **S**ystem, переводится — система управления контентом. Как правило, CMS реализовано в виде веб-приложения, однако, это может быть и программа, которая устанавливается на операционную систему.

Работа CMS заключается в предоставлении возможности изменения содержимого базы данных в удобном визуальном редакторе. Таким образом, она позволит оперативно отредактировать данные не только владельцу веб-ресурса, но и техническому специалисту, а ее установка повысит удобство работы в большинстве случаев.

Примеры наиболее популярных систем управления:

- ➔ WordPress
- ➔ 1С-Битрикс
- ➔ Joomla
- ➔ Drupal
- ➔ OpenCart
- ➔ MODx
- ➔ Magento
- ➔ TYPO3
- ➔ Tilda

- Домен - это имя сайта. (доменное имя, доменный адрес) — это, простыми словами, «название» сайта. Понятия «домен» и «сайт» часто путают, но это не одно и то же. Сайт — это веб-страницы, которые отображаются в интернете, т. е. контент. А домен сайта — это его уникальный «адрес». Если у вашего сайта не будет домена, пользователи просто не найдут к нему дорогу и не увидят содержимое.
- CSS – это формальный язык, служащий для описания оформления внешнего вида документа, созданного с использованием языка разметки (HTML, XHTML, XML). Название происходит от английского Cascading Style Sheets, что означает «каскадные таблицы стилей».

- CSS можно охарактеризовать простыми словами как набор правил, описывающих, как должен выглядеть элемент.

Что происходит, когда пользователь вводит запрос в браузер?
Пользователь вводит в браузере адрес сайта>браузер ищет IP адрес сервера (такая информация хранится в распределенной системе серверов — DNS)

Перед тем, как обращаться к DNS, браузер пытается найти запись об IP-адресе сайта в ближайших местах, чтобы сэкономить время:

- Сначала в своей истории подключений. Если пользователь уже посещал сайт, то в браузере могла сохраниться информация с IP-адресом сервера.
- В операционной системе. Не обнаружив информации у себя, браузер обращается к операционной системе, которая также могла сохранить у себя DNS-запись. Например, если подключение с сайтом устанавливалось через одно из установленных на компьютере приложений.
- В кэше роутера, который сохраняет информацию о последних соединениях, совершенных из локальной сети.

Не обнаружив подходящих записей в кэше, браузер формирует запрос к DNS-серверам, расположенным в интернете.

Например, если нужно найти IP-адрес сайта mail.vc.ru, браузер спрашивает у ближайшего DNS-сервера «Какой IP-адрес у сайта mail.imperium.im?».

Сервер может ответить: «Я не знаю про mail.imperium.im, но знаю сервер, который отвечает за imperium.im». Запрос переадресовывается дальше, на сервер «выше», пока в итоге один из серверов не найдет ответ об IP-адресе для сайта.

Как только браузер узнал IP-адрес нужного сервера, он пытается установить с ним соединение. В большинстве случаев для этого используется специальный протокол — TCP.

TCP — это набор правил, который описывает способы соединения между устройствами, форматы отправки запросов, действия в случае потери данных и так далее.
После установки соединения браузер отправляет специальный запрос, в котором просит сервер отправить данные для отображения страницы. В этом запросе содержится информация о самом браузере, временные файлы, требования к соединению и так далее.
Задача браузера — как можно подробнее объяснить серверу, какая именно информация ему нужна.

В общении браузера и сервера выделяют два типа запросов. GET-запрос используется для получения данных с сервера — например, отобразить картинку, текст или видео. POST-запрос — используется для отправки данных из браузера на сервер, например, когда пользователь отправляет сообщение, картинку или загружает файл.
Сервер получил запрос от браузера с подробным описанием того, что ему требуется. Теперь ему нужно обработать этот запрос. Этой задачей занимается специальное серверное программное обеспечение — например, nginx или Apache. Чаще всего такие программы принято называть веб-серверами.
Веб-сервер в свою очередь перенаправляет запрос на дальнейшую обработку к программе-обработчику — например, PHP, Ruby или ASP.NET. Программа внимательно изучает содержимое запроса — например, понимает, в каком формате нужно отправить ответ и какие именно файлы нужны. И собирает ответ.
Когда ответ сформирован, он отправляется веб-сервером обратно браузеру. В ответе как правило содержится контент для отображения веб-страницы, информация о типе сжатия данных, способах кэширования, файлы cookie, которые нужно записать и так далее.
Браузер распаковывает полученный ответ и постепенно начинает отображать полученный контент на экране пользователя — этот процесс называется рендерингом. Сначала браузер загружает только основную структуру HTML-страницы. Затем последовательно проверяет все теги и отправляет дополнительные GET-запросы для получения с сервера различных элементов — картинки, файлы, скрипты, таблицы стилей и так далее. Поэтому по мере загрузки страницы браузер и сервер продолжают обмениваться между собой информацией.
Параллельно с этим на компьютер как правило сохраняются статичные файлы пользователя — чтобы при следующем посещении не загружать их заново и быстрее отобразить пользователю содержимое страницы.
Как только рендеринг завершен — пользователю отобразится полностью загруженная страница сайта

Part IX. Инструменты для Проджекта в IT

ПО для трека прогресса и отображения задач, таск-системы:

- ➔ Asana
- ➔ Jira
- ➔ Trello
- ➔ Redmine
- ➔ Mantis

ПО для коммуникаций

- ➔ Slack
- ➔ Mattermost

ПО для составления схем

- ➔ MindManager
- ➔ XMind
- ➔ Miro
- ➔ FlowMapp
- ➔ Canva

ПО для хранения документации

- ➔ Notion
- ➔ GoogleDocs
- ➔ Slite

Git:

- ➔ BitBucket
- ➔ Gitlab
- ➔ Github

Design:

- ➔ Figma
- ➔ CMS no-code platforms: Tilda

Платформы для поиска вакансий и нетворкинга:

- ➔ Indeed
- ➔ LinkedIn
- ➔ WeWork
- ➔ Glassdoor
- ➔ Career Builder
- ➔ IdeaList
- ➔ Dice
- ➔ Monster.com
- ➔ ZipRecruiter
- ➔ GulfTalent
- ➔ Angel.co
- ➔ Bayt.com
- ➔ Jobble
- ➔ Trud.com
- ➔ teamblind.com - сайт с анонимными отзывами и обсуждениям различных компаний от их сотрудников

- ➔ levels.fyi - данные по з/п в различных компаниях с разбивкой по уровням и локации
- ➔ interviewing.io - платформа, где можно тренировать навыки для интервью с сотрудниками из MAANG и др. компаний
- ➔ RemoteJobs
- ➔ We work remotely.com
- ➔ Remote jobs (In Google Search)

Документация для Проджекта в IT:

- ➔ Устав проекта
- ➔ WBS/ИСР
- ➔ RFA “Request for Application” (информация, полезная при передаче проекта другому ПМу): о чем проект/сроки/команда/клиент/пожелания к руководителю проектов
- ➔ Краткая характеристика стейкхолдеров по методологии DISC
- ➔ Файл Handover при передаче дел (Наименование проекта, Статус, Задача, Контакты, Сроки, Замечания)
- ➔ SRS (Software Requirement Specification)//SOW - ТЗ, спецификация, требования
- ➔ Project Scope // Roadmap
- ➔ Stakeholder Map - Реестр заинтересованных сторон, матрица влияния стейкхолдеров
- ➔ Project/Product Vision
- ➔ Continuity Business Plan // Feature List // Backlog - для будущих идей и масштабирования проектов
- ➔ Follow-up // Summary - документ с договоренностями после разговора/созвона/встречи
- ➔ **Meeting Notes** (Kick off Meeting, Planning Meeting, Stand-up Meeting, Retrospective)
- ➔ Реестр рисков (Risk Register)
- ➔ Risk Chart
- ➔ Журнал изменений (Change Requests)
- ➔ SLA - Service Level Agreement соглашение об уровне оказываемых услуг – это документ, который составляется между двумя сторонами (потребитель – исполнитель) и отражает условия предоставляемых услуг
- ➔ Daily Plan, Dev Plan (план активности команды и трек прогресса - для клиентов)
- ➔ Gantt Chart
- ➔ Action Plan

Part X. Интервью на позицию Senior PM

Вопросы + Опорные пункты для практики ответов

Группы вопросов:

- ➔ Как ты делаешь? Расскажи на примерах
- ➔ Ситуационные вопросы
- ➔ Поведенческие кейсы и проигрывание ситуаций с уточняющими вопросами
- ➔ Про опыт и реальные кейсы на практике

Что спрашивают на интервью?

Самый сложный проект?

Что говорить:

- ➔ Масштаб проблем, с которыми работал ПМ?
- ➔ Что в понимании ПМа “сложно”?
- ➔ Важно! Рассказать, как этой сложностью управляли и справились

Самая большая команда, которой управляли?

Что говорить:

- ➔ С каким масштабом, структурой команды работал ПМ
- ➔ Отметить организаторские, управленческие, лидерские навыки на примере
- ➔ Как организовал команду
- ➔ Были ли в подчинении менеджеры, тимлиды
- ➔ Какие scaled фреймворки использовали

Ваш опыт управления проектами?

- ➔ Опыт в разных бизнес-доменах
- ➔ Опыт в различных технических областях
- ➔ Опыт в работе с различными контрактными моделями

Задачи, которые решали на проектах?

- ➔ Какую конкретную роль и функцию выполнял? Что делал на проектах?
- ➔ Какая была зона ответственности? Масштаб - пример
- ➔ Руководили ли скоупом/сроками и бюджетом? Пример

Опыт управления другими менеджерами?

- ➔ В какой роли управлял другими?
- ➔ Позволяло ли это в вашей компании оргструктура?
- ➔ Какая зона ответственности была при управлении другими ПМ`ами: отвечали за менторство или за качество деливери?

Как вы оцениваете качество работы подчиненных ПМов?

- ➔ Насколько системно подходите к оценке подчиненных?
- ➔ Какой стиль лидерство используете?
- ➔ На каких критериях фокусируетесь?
- ➔ Понимание успешного/провального проекта
- ➔ Метрики для оценки качества работы ПМов (при работе с клиентами, командой, выполнение финансовых целей)

Мотивация других ПМов?

- ➔ Как ПМ строит систему мотивации (материальную/нематериальную)?
- ➔ Что по вашему мнению заряжает людей?
- ➔ Какие критерии и методы материальной мотивации использовали? (квартальные/годовые), как рассчитывали бонусы для каждого?
- ➔ Ваши инструменты для мотивации команды? Что мотивировало вашу команду?

Отработка эскалации в старшей роли?

- ➔ Насколько успешно умеешь отрабатывать острые ситуации с командой и клиентами?
- ➔ Как будете общаться со своим ПМом?
- ➔ Насколько глубоко будете вмешиваться в его работу?
- ➔ Как найти баланс, чтобы решить ситуацию и не нарушить право менеджера управлять командой?
- ➔ Как будете строить коммуникацию с эксалирующей стороной?
- ➔ Как будете выходить из кризисной ситуации и какое поведение вам свойственно в этой ситуации?

Собеседование ПМов?

- ➔ Как будете строить процесс собеседования?
- ➔ Рассказать о структуре собеседования
- ➔ Какие критерии будете использовать для оценки ПМа по пунктам: решение кризисов, управление проектами, командой?

Чек-лист для кандидата перед интервью

- ☐ **Ответы на вопросы должны быть полными и детальными**
- ☐ **Будьте готовы к уточняющим вопросам и моделированию примеров в контексте разных ситуаций**
- ☐ **Всегда подкрепляйте свои примеры конкретикой, цифрами, примерами, а не расплывчатыми фразами и абстрактными и общими понятиями**
- ☐ **Рассказать, как со своей позицией ПМ будет решать боли работодателя, помогать компании и приносить ценность – в чем сильные стороны**
- ☐ **Нужно продавать себя, за вас это никто не сделает. Говорить на языке результатов и достижений (важно подчеркивать глаголами совершенного действия: “сделал”, “улучшил”, “создал”, “организовал”)**
- ☐ **Подготовить заранее структурное описание самого сложного проекта/кейса и вашего провала**
- ☐ **Подготовить блок аргументов в мотивации начать новую позицию (почему, зачем, для чего это, каких результатов хотите добиться, почему именно эта позиция)**
- ☐ **Точные, структурные ответы по существу вопроса! Важно: отвечать именно на тот вопрос, который задается**

* Сервис для тренировки ответов на интервью на английском для Project Manager:

https://grow.google/certificates/interview-warmup/

Список вопросов:

1. What are you looking for in your next job?
2. Can you please tell me a bit about yourself?
3.
4. Please share a time when you set a goal for yourself and achieved it. How did you go about that?
5. Imagine you are tasked with running the production of a site build. What method would you prefer to work with, agile or waterfall?
6. Projects require different teams and stakeholders to work together. Tell me about a project where you had to engage people with different or competing interests.
7. Please tell me about some of your strengths and weaknesses.
8. Tell me about a time you had to deliver on multiple competing priorities. What did you do, and what were the results?
9. Tell me about a time you managed a project, either personal, at work, or at school. How did you determine what order the tasks had to be completed in?
10. Describe a situation when you disagreed with someone at work. What did you do, and what was the result?
11. Tell me about one of your favorite projects. Why is it special to you?
12. Imagine you're assigned to a project upgrading internet speed on a street. During the feasibility analysis, you realize the upgrade is scheduled for a time of year with heavy rainfall. What would you do?
13. Imagine there was no formal project management tool available. How would you keep track of project information?
14. Please tell me about a time when you resolved a conflict between team members.
15. Tell me about a time when you felt you did not communicate well or your communications were poorly received. How did you correct the situation?

Как менеджеру пройти техническое собеседование: 37 часто задаваемых вопросов

12 главных пунктов, по которым задают вопросы на техническом собеседовании

- Жизненный цикл продукта
- Методологии
- Бизнес-аналитика
- Техническая документация
- Инструментарий проекта

- Архитектура
- Клиент-сервер
- REST
- Фреймворки, библиотеки, код
- Git
- Натив/кросс-платформы
- Тестирование

Жизненный цикл продукта

Какие этапы есть в жизненном цикле ПО

Формирование требований к ПО на стадии анализа, проектирование, реализация, тестирование, внедрение, эксплуатация и сопровождение, снятие с эксплуатации. Сначала идет стадия, которая отвечает на вопрос: «Что мы будем делать?». Вторая стадия отвечает на вопрос «Как мы это будем делать?». Дальше стадия реализации задуманного проекта, после — тестирование и внедрение. Завершающая стадия — сдача проекта и дальнейшая техническая поддержка.

Чем отличается формирование требований от проектирования

Формирование требований — это процесс, при котором менеджер выясняет у заказчика, какие именно бизнес-процессы он хочет автоматизировать и что он хочет получить. На этом этапе работают аналитики и проектный менеджер. Проектирование — процесс, при котором собирается команда разработчиков и определяет, как будут реализованы требования.

Что такое поддержка

Техническая поддержка не означает, что вы будете отвечать на звонки клиентов вашего заказчика. Речь идет о решении проблемных задач в случае нарушения работы продукта. Если что-то «ломается», перестает работать, команда разработчиков должна оперативно отреагировать и сделать так, чтобы все было хорошо.

Как происходит снятие с эксплуатации

Если продукт теряет свою актуальность, или его заменяют на более новое и техничное решение, происходит снятие с эксплуатации. Разрабатывается план, согласовывается, после чего начинается постепенное снятие с эксплуатации.

Пример: на предприятии есть десктопная версия продукта, но компания решила перейти на новое приложение, которое работает с мобильных устройств.

Разрабатывается стратегия перехода, подготавливается продукт, и постепенно переносятся все данные, обучаются сотрудники, внедряются новые инструменты.

Какие сотрудники нужны на каждом этапе жизненного цикла

При формировании требований участвуют бизнес-аналитики, на стадии проектирования ключевые люди — архитекторы и лиды разработчиков. На реализации проекта нужны программисты, дизайнеры. Тестированием занимаются QA инженеры. Для внедрения нужны практически все участники проекта, а «звезда» — проектный менеджер. Если команда выделяет инженера

сопровождения, то он отвечает за техподдержку. Если его нет, то участвуют QA, разработчики и проектный менеджер.

Методологии

Выбор методологии не относится к техническому собеседованию, но есть вопросы, которые косвенно влияют на команду разработчиков. Например, какую форму оплаты труда выбрать. Вы можете столкнуться с такими вопросами:

В чем разница между Fix Price и Time&Materials

Fix Price подразумевает, что мы ведем проект за определенный бюджет. Компания оговаривает с заказчиком, что она делает и за какие деньги. Time&Materials применяется в том случае, если у клиента есть неопределенности и он не знает, каким будет финальный продукт. Он готов выкупать команду и платить за сделанную работу. Time&Materials часто применяется в гибкой методологии Scrum.

Какая модель лучше

Fix Price отлично работает для проектов, которые имеют строгий фиксированный бюджет и они четко знают, что хотят получить. В таком случае любые дополнения, изменения или предложения просто не реализуются. Time&Materials подходит для стартапов и проектов, где сложно понять точное количество задач и как будет выглядеть финальный продукт.

Бизнес-анализ

В чем разница между бизнес-аналитиком и системным аналитиком

Бизнес-аналитик — это человек, который разбирается в бизнес-процессах компании, которые необходимо автоматизировать. Детально разбирается в их особенностях и ищет подходы, которые помогут улучшить процессы. Изучает требования, цели и задачи заказчика.

Системный аналитик переводит информацию из формата «что нужно сделать» в формат «Как это сделать». Он составляет схемы, диаграммы, работает с технической командой и погружается в архитектуру.

Что такое User Story и Use Cases

User Story — это «история» клиента компании, который проходит путь с ней от начала до конца. Например, от входа в интернет-магазин до регистрации, оформления заявки и покупки товара.

Use Cases — это подробное рассмотрение User Story на определенном этапе. Например, клиент оставил заявку на сайте. Это один из этапов User Story. Но сам кейс — это заполнение конкретных полей при заказе.

Разница между функциональными и нефункциональными требованиями

Функциональные требования предполагают описание того, что нужно сделать. Изучаются требования заказчика. Нефункциональные требования касаются нетехнической части. Например, время отклика, на каких устройствах должно работать приложение.

Техническая документация

Менеджеру приходится постоянно работать с технической документацией. Очевидно, что на собеседовании будут вопросы в этой теме.

Что такое техническое задание

Без ТЗ — результат ХЗ. Это самый главный документ, задание на разработку, в котором описывается, что нужно сделать, как это сделать. Туда же пишутся функциональные и нефункциональные требования. Техническое задание — это документ, который описывает, что должно получиться в итоге. Эти требования фиксируются и являются основой для работы всей команды. ТЗ дает защиту как разработчикам, так и клиенту. Потому что каждый соблюдает требования ТЗ.

Use Cases, Test Cases

Use Cases описывает сценарий взаимодействия участников. Он может быть представлен в виде диаграмм или в текстовом виде. Они нужны разработчикам, тестировщикам и всей проектной команде, чтобы понимать полную спецификацию требований. Test Cases нужны для того, чтобы тестировщик прошел по всему продукту и ничего не упустил. Они должны быть написаны так, чтобы любой тестировщик из другого проекта мог разобраться в них.

Где хранить и менеджерить требования

Все зависит от взаимоотношений с заказчиком. Для этой задачи подходит Confluence, причем удобнее хранить у себя, а не на сервере заказчика. Потому что менеджер может писать ту информацию, которую не хотел бы показывать заказчику. Также можно использовать Jira или Notion.

Инструменты проекта

Какой Task Tracker лучше выбрать

Каждый таск трекер имеет много классных фич, поэтому нужно смотреть возможности инструментов и подбирать под конкретные проекты. Часто пользуются Jira. Это не значит, что она лучшая, но этот инструмент в 33% вакансий на должность проектного менеджера.

Для чего нужен Project Planning

В первую очередь, чтобы построить диаграмму Ганта. Это базовый инструмент для проектного менеджера, где можно отслеживать этапы ведения проекта, строить его от начала до конца. Инструменты Project Planing нужны для того, чтобы отслеживать результативность выполнения задач, смотреть, сколько остается времени и бюджета.

Архитектура

Это отдельная большая тема, в которой менеджер должен хорошо разбираться, если хочет качественно вести IT-проекты, на котором мы подробно разбираем, что это такое, как устроена и зачем менеджеру вообще знать архитектуру.

Архитектура для проектного менеджера

PM должен понимать, где мы храним данные, с помощью каких данных мы вынимаем эти данные, будет ли архитектура клиент-серверной или другой.

Архитектура — это схематическое пояснение, откуда и куда перетекают данные, как они обрабатываются, на каких этапах предоставляются пользователям.

На собеседованиях могут спрашивать детальнее про архитектуру, но это тема отдельной статьи. В нашем блоге вы можете почитать, как выглядит архитектура и как менеджеру понимать ее.

Чем отличаются бэкенд и фронтенд

К примеру, у нас есть мобильное приложение. То, что у вас на устройстве, это фронтенд. То, с чем пользователь взаимодействует и видит. Бэкенд — это процессы, которые не видны пользователю. Обработка данных, хранение, передача, обращение к серверам. Этот же принцип работает с любым продуктом. То, что видит пользователь — фронтенд. Все, что происходит «за дверьми» — это бэкенд.

Что значит тонкий клиент и толстый клиент

Нет, речь не об объемах талии. На этом вопросе «валятся» все, кто не имеет технического образования и скиллов.

«Толщина» клиента зависит от того, сколько логики находится на клиенте. Если у приложения минимальная логика на самом клиенте, а основная — на сервере, то образуется тонкий клиент. Если при загрузке приложения обрабатывается сразу большое количество данных на самом клиенте, то «толщина» увеличивается.

Разница между ними в вариантах обработки данных. Толстые клиенты преимущественно работают с информацией на основе собственных программных возможностей, в то время, как тонкие клиенты применяют ПО центрального сервера для обработки. Все браузеры и веб приложения, онлайн игры — это тонкие клиенты. А вот Microsoft Outlook или Office 365 — толстые.

Что такое база данных

Это место, где хранятся и обрабатываются все данные. Представлены в виде таблиц, в которых разделяются данные на столбцы и строчки. Например, имя пользователя, его номер телефона, почта. Все таблицы связаны между собой, поэтому при запросе одних данных, можно получить и другую информацию.

DNS — что это такое

На собеседованиях часто приводят подобную ситуацию: пользователь вводит какие-то данные в поисковую строку браузера и нажимает кнопку «Найти». Как происходит весь процесс? На этот вопрос можно отвечать детально, но рекрутер ждет пояснения, что такое DNS.

Если есть физические сервера, на которых расположена информация, к ним нужно как-то получить доступ. У каждого сервера есть имя, которое выглядит в виде IP-адреса. Чтобы не запоминать большое количество цифр, была разработана DNS система. DNS имеет читабельный вид и иерархическую структуру, благодаря чему пользователь дает запрос и получает доступ к информации.

Зачем нужна репликация

Репликация нужна, чтобы делать копии баз данных. Сама база данных хранится на сервере, Но что с ней произойдет, если с сервером что-то случится? Сгорит, повредится, будет взломан? База данных потеряется. Годы работы могут исчезнуть. Избежать этого помогает репликация. Делаются копии баз данных и размещаются на разных серверах и локальных носителях. Регулярно БД обновляется, и с настраиваемой периодичностью реплики пополняются обновленной информацией.

В чем разница между микросервисом и монолитом

Монолит — это централизованная обработка запросов, а микросервисы — индивидуальная обработка запросов. Монолитное приложение выглядит, как единый общий модуль, в котором происходит вся работа. Архитектура микросервиса имеет несколько небольших развертываемых служб.

Что лучше — нельзя однозначно сказать, потому что каждый имеет свои особенности. Монолитная архитектура считается более традиционной, независимой от других сервисов, имеет единую базу кода. Чтобы внести изменения, необходимо обновлять весь стек. На начальных этапах проекта это выгодное решение, итак как развертывание происходит легче. Микросервис легче масштабировать, тестирование и обновление происходят быстрее и дешевле, приложение становится более гибким.

Клиент-сервер

Что такое клиент-сервер

Это элементы архитектуры, которые подразумевают, что есть клиентская часть, которая взаимодействует с пользователем, а есть серверная часть — логика приложения. Клиентская часть отвечает за то, как показать пользователю данные, каким цветом вывести окошки, где будет кнопка. Серверная часть отвечает за сбор и хранение данных, обработку запросов и логику взаимодействия пользователя с продуктом.

Зачем нужен REST

Это набор определенных правил, которые показывают, как нужно общаться между клиентом и сервером. Это некий «транспорт», по которым передаются данные. Клиент запрашивает у сервера какие-то данные. Этот запрос идет на сервер и он отдает их. REST — это свод правил, который помогает серверу и клиенту понимать друг друга.

Куда выносить какую логику

Тут нет универсального ответа. Одни считают, что лучше всю логику выстраивать на бэкенде, а на клиент не давать нагрузку. Другие утверждают ровным счетом наоборот. Ответ, который понравится: все зависит от того, что мы делаем и в какой среде находимся. Можно вносить все на клиент, но все, что связано с безопасностью, выстраивать на бэкенд. Сложные вычисления которые связаны с оплатой, тоже лучше оставлять в бэкенд.

REST

В двух словах мы уже сказали, что REST — это определенный стиль взаимодействия между клиентом и сервером. Он помогает наладить

коммуникацию между запросом и получением данных. В теме REST могут возникнуть следующие вопросы:

Что такое JSON, XML, HTML

Это форматы файлов, которыми обменивается клиент и сервер. Они не перечисляют все данные через запятую, их нужно структурировать. Это и есть варианты структуры. Json используется для обмена данными в мобильных устройствах, xml, html работают в вебе.

Методы update, delete, get, post

Мобильные девайсы взаимодействуют с бэкендом какими-то запросами. Условно говоря, это ссылка и какие-то дополнительные поля. Мы идем на ссылку и даем одну из команд: update, delete, get, post.

- get — мы что-то забираем, например «Дай список контактов»
- post — говорим, что нужно что-то добавить, например, нового друга в социальной сети
- delete — этой командой мы говорим, что нужно удалить данные, например, файл с телефона
- update — обновление файла: переименование, внесение новых данных.

Фреймворки и библиотека

В чем разница между фреймворком и библиотекой?

Фреймворк — это некая структура с кодом, которая позволяет удобно что-то разрабатывать. Библиотека — это тоже структура с кодом, но она берется извне. Ее берут, как готовое решение для разработки. Кажется, что это более удобный и легкий способ разработки. Но библиотека — это стороннее решение, которое сложно дорабатывать.
Например, нужен календарь. Можно взять готовое решение из библиотеки и внедрить в приложение. Но если заказчик попросит добавить цвета в календарь, поменять шрифт, изменить начало недели не с воскресенья, а с понедельника, править ее будет сложно. Библиотека подходит для типовых и быстрых решений.
Git
Git — это система управления версиями. У нас есть разработчики, которые постоянно что-то дорабатывают, внедряют новые фичи. Чтобы не запутаться и получить строгую структуру, вводится система управления версиями. По ней нетехническому специалисту могут задавать следующие вопросы:

Как работает Git
Чтобы не потерять готовые результаты, создается главная ветка — Мастер. Это уже рабочие утвержденные решения, которые подходят для релиза. Дальше приходит идея внедрить новую фичу. В мастере фиксируется «отправная точка», делается фича, тестируется и проверяется. В Git она идет отдельной веткой. В ней ведется разработка, связанная только с этой новой фичей. Когда она сделана, протестирована и принято решение внедрять ее, она мёрджится в Мастер.

Зачем нужен Git
Чтобы не запутаться. Самая сложная ветка — разработчиков. Там происходят основные процессы, появляется много багов в ходе работы и отслеживать их становится сложно. Но Git помогает выстроить ветки так, чтобы было наглядно понятно, что уже работает, а что нуждается в доработке.

Что такое merge, push, pull
1. **Merge** — это процесс слияния кода из одной ветки в другую. Мержить — переносить код. Когда вам понятно, что фича работает, она добавляется в основную версию Git.
2. **Push** — процесс, когда отдельный кусок кода нужно добавить в какую-то ветку.
3. **Pull** — обратный процесс, когда нужно что-то убрать из ветки с кодом.

СОЗДАНИЕ РЕПОЗИТОРИЯ

С нуля -- Создать новый локальный репозиторий
$ git init [project name]
Скачать с существующего репозитория
$ git clone my_url

ОТСЛЕЖИВАНИЕ РЕПОЗИТОРИЯ

Просмотреть список новых или измененных файлов, которые еще не закомитены
$ git status

Показать изменения в файлах, которые еще не поставлены
$ git diff

Показать изменения в индексированных файлах
$ git diff —cached

Показать все индексированные и не индексированные изменения файлов
$ git diff HEAD

Показать различия между двумя комитами
$ git diff commitl commit2

Показать дату изменения и автора для файла
$ git blame [file]

Показать изменения для определенного комита и/или файла
$ git show [commit] : [file]

Показать полную историю изменений
$ git log

Показать историю изменений для файла/папки включая различия(diffs)
$ git log -p [file/directory]

РАБОТА С ВЕТКАМИ

Показать все локальные ветки
$ git branch

Показать все локальные и удаленные ветки
$ git branch –av

Переключится к ветке my_branch и обновить рабочую директорию
$ git checkout my_branch

Создание новой ветки с именем new_branch
$ git branch new_branch

Удалить ветку с именем my_branch
$ git branch -d my_branch

Объединить branch_a в branch_b
$ git checkout branch_b
$ git merge branch_a

Добавить Tag к текущему комиту
$ git tag my_tag

ИЗМЕНЕНИЯ

Индексировать файл готовый к комиту
$ git add [file]

Индексировать все файлы готовые к комиту
$ git add .

Зафиксировать индексированные файлы с комментарием в историю
$ git commit -m "commit message"

Зафиксировать все отслеживаемые файлы с комментарием
$git commit -am "commit message"

Неиндексированные файлы, получить изменения файла
$ git reset [file]

Откатить все до последней фиксации
$ git reset —hard

СИНХРОНИЗАЦИЯ

Получить последние изменения с удаленного сервера (без слияния)
$ git fetch

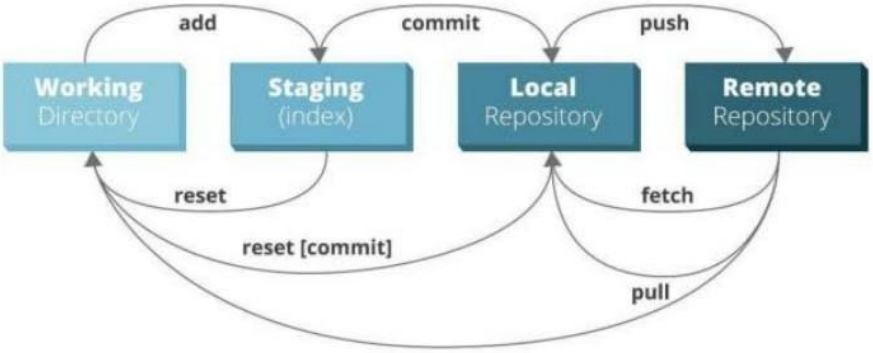
Получить последние изменения с удаленного сервера и слияния
$ git pull

Получить последние изменения с удаленного сервера и перебазировать
$ git pull —rebase

Применить локальные изменения на удаленный сервер
$ git push

ФИНАЛ

В случае сомнений, используйте помощь
$ git command —help



Поддержка серверов и ПО, DevOPS
https://system-admins.ru

Кросс-платформа

В современной разработке все реже используются только десктопные или только мобильные версии продукта. Но отдельно разрабатывать под каждую систему продукт достаточно дорого. Поэтому часто используется кросс-платформа. Нетехнический специалист должен понимать, что это за процессы и как они устроены.

Что такое натив

Если разработка ведется на родном языке для определенной платформы, это называется натив. Например, если делается продукт под операционную систему iOs, то используется Swift.

Что такое кросс-платформа
Это процесс, при котором приложение разрабатывается с кодовой базой, которая работает на разных операционных системах. То есть приложение будет работать как на «яблоках», так и на андроидах. Несмотря на то, что это кажется более выгодным, кросс-платформенные решения имеют довольно много недостатков и не всегда работают качественно. Под каждый проект определяется стек технологий, и при разработке архитектуры решается, использовать натив или кросс-платформу.

Что дешевле
Для быстрого запуска стартапа с какой-нибудь функциональностью дешевле сделать кросс-платформенное решение. Если нужна хорошая функциональность, выгоднее делать ее на нативе. Потому что на кросс-платформе придется «натягивать» многие решения, в итоге это займет больше времени и сил. С точки зрения общей оценки, дешевле кросс-платформа. Но не всегда она подходит для проектов.

Тестирование

Какие типы тестирования бывают
Функциональное тестирование — мы берем и тестируем конкретную функциональность. Смоук тестирование применяется, когда обнаруживается маленький баг, не влияющий на функциональность. Регрессионное тестирование — это полный процесс проверки на баги всего кода, как правило, применяется перед релизом продукта.
Нагрузочное тестирование применяется для сложных продуктов с нефункциональными требованиями, связанными с нагрузкой. Например, приложение должно работать при одновременной нагрузке в 20 000 пользователей.

Интеграционное тестирование применяется тогда, когда сервер уже написан, а клиент еще меняется. Автотесты работают так: кодируются определенные действия или запросы, и проверяются ответы. Часто применяется при интеграционном тестировании.

Что такое приоритеты
Каждому багу присваивают статус, который определяет важность тестирования и исправления. Как правило, используются статусы «Критичный», «Высокий», «Средний», «Низкий», «Блокер». При критическом статусе продукт не может работать, пока его не исправят, но при этом функциональность можно проверять. Блокер — это статус, при котором невозможно дальнейшее проведение тестов, пока он не будет устранен.

Bonus: Interview Tips to Get a Job Offer

Информация по платному Вебинару “Job Offer”

Пошаговый план по поиску:

1. **Подготовить список из 25 компаний**

- определиться с индустрией, компанию, департамент
- проверить, насколько подходишь ты под требования компании и что требуется
- Как и где искать: LinkedIn, Fortune 500, Glassdoor.

2. **Приоритизировать и “деприоритизировать”** (вживую пообщаться с сотрудниками компании) - потренироваться в собеседованиях на тех компаниях, в которые вам хочется попасть меньше всего и уже потом подаваться в те организации, которые вам нравятся и где вы действительно хотите попробовать себя

- Изучить ценности компании
- Изучить страну - традиции, уровень жизни
- Город, в котором потенциально будешь жить
- Посмотри perks компании: оплачивается ли жилье, питание, наличие спорт-зала, фото офиса

3. **Получить Referral** (человека, который вас порекомендует в компанию)
   Важно: зачастую сотрудники сами заинтересованы в том, чтобы привести хорошего кандидата в компанию, так как за это им полагается бонус.
   Поэтому первое впечатление важно, нужно расположить к себе возможно, будущего коллегу!

Примеры письма сотруднику понравившейся вам компании:

Hello, (name of the Referral)!
My name is...

I find your background really impressive!
I am currently in the position of Project Manager in the field of (IT, Blockchain and FinTech) areas.
And I am motivated to get to ... (name of the company) as a (name of the position).
I would immensely appreciate it if you could share your experience as well as some tips for a potential candidate!

Thank you!

4. **CV + Cover Letter** (общие принципы и как выделиться?)
   !!! Любое заявление нужно продкреплять фактом и историей
   Покажите логику своего пути:
   В CV - хронологическое изложение фактов не только о вашей работе, о и о вас, как о человеке
   В Cover Letter - не просто перечисление фактов из CV, который очевидно привел вас к решению податься в эту компанию
   Готовиться! Связаться с сотрудниками, узнать о тесте, попросить совета в подготовке, погуглить эти пробные/подобные тесты
5. Digital Assessment
6. **Interview** (11 вопросов, которые нужно подготовить в 100% случаях, самое главное в любом интервью)

11 вопросов:

1. Расскажи о себе (2-3 минуты) (задает тон всему интервью и дальнейшему разговору на собеседовании)
2. 3 главных достижения
3. 3 провала (и чему вы научились/как вышли из ситуации провала/что выявили)
4. 3 сильные стороны// 3 слабые стороны (подкрепленные примером и историей, как вы работаете над слабыми сторонами)
5. Почему именно эта индустрия
6. Почему именно эта компания (предварительно нужно познакомиться с сайтом компании и ее миссией, видением, понять, какие продукты выпускает и чем занимается)
7. Пример лидерства? Расскажите о ситуации, когда вы проявили себя как лидера
8. Пример, как вы кого-либо переубедили (Например, когда команде и вам пришлось убеждать клиента в перспективности предлагаемого вами решения). На примере важно показать ваше умение быть убедительным в нужных ситуациях.
9. Пример конфликтной ситуации
10. Пример работы под давлением
11. Пример “entrepreneurial drive” (как вы что-то с нуля создали, сделали, внедрили и какие результаты были получены по итогу вашей инициативы)

Mock Interview

1. Можно пройти пробное интервью (mock interview)
2. Или воспользоваться услугами карьерного эксперта (Ссылка: https://h.careers/curators-list)
3. Как составить резюме на английском + советы: https://devby.io/news/rezume-english

Проработать кейсы и внезапные вопросы: Онлайн-такси, Доставка, Стриминг музыки
На примерах компаний: Gett, TikTok, Microsoft, Yandex, Tinkoff, Sber, Revolt
Примерные вопросы:

1. Что бы вы изменили в ... компании?
2. Как бы вы разработали фичу для сайта компании/платформы..?

3. Распишите стратегию компанию ...
4. Что не хватает компани ..? Какие бы фичи вы предложили?

Вебинар: 5 секретов поиска работы внутри страны и за рубежом

5 шагов поиска работы за рубежом:

1. Выбрать карьерную цель
2. Резюме
3. Собеседование
4. Нетворкинг
5. JOb Offer

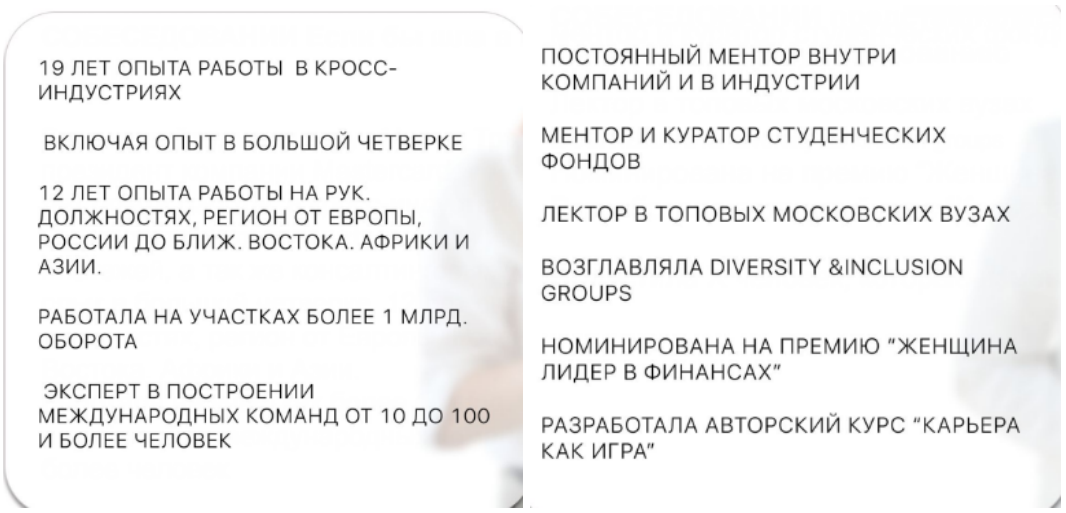
** HR в среднем тратит 10 секунд на резюме и первичный скрининг*

Какие блоки должны быть в резюме:
Резюме – это ваша упаковка, оно должно быть продающим (!!!)

- **Шапка** (контактные данные)
- **Саммари**//Branding Statement – самая главная информация о вас, ваша мотивация в параграфе
- **Достижения** (перечислить сильные глаголы+язык вакансии): детали, цифры, конкретика

Чек-лист по составлению резюме:
https://drive.google.com/file/d/1L7OmyORMP3iS_wfvKBK-V--5DsMoPJFM/view?usp=sharing

*****Пример саммари и рассказа о себе:**



Основные ошибки при поиске:

- Думаете, что работодатель из тысячи заявок заметит именно вас
- Не отслеживать тренды на рынке труда (2020-е – AI)

- Незнание своей рыночной стоимости
- Не используете нетворкинг

Тренды поиска работы:

- Удаленная работа
- Навык самопрезентации - готовиться к удаленной встрече
- Спрос на soft skills
- Качественные условия труда

Тест на выявления ведущего мотива работы: https://psytests.org/profession/herz.html

Собеседование

Ошибки:

- Не узнали ничего о компании
- Не смогли вспомнить ситуацию, где вы проявили себя, как.. (лидер/ответственный/...)
- Не подготовили и не продумали свои условия заранее
- 90% ответов на вопросы у вас уже должно быть заготовлено заранее (!!!)
- Не подготовились к неудобным вопросам
- Не задали свои вопросы в конце беседы или в середине диалога с HR

Каналы поиска работы:

- Рекрутинговые агентства
- LinkedIn
- Нетворк и рефералы

Как расти в доходе на 30-50%?

- Оцифровать свои компетенции
- Изучить рынок и средние зарплаты
- Подготовиться к диалогу с руководителем (3-4 беседы и согласования)

Карта компетенций Project Manager в IT

	Junior PM	Middle PM	Senior PM
Soft Skills	1. Базовое понимание ролей в команде в команде (кто за что отвечает) 2. Участие в пресейле, сопровождение по оценкам 3. Контроль и мониторинг сроков, бюджета, объема задач 4. Общение с клиентом 5. Ведение и управление проектами Студии с нуля до сдачи 6. Проведение переговоров с клиентами 7. Разработка креативных идей в работе над проектами 8. Проработка и изучение Технических заданий, полученных от Заказчика 9. Создание прототипов 10. Составление презентаций и презентация работ клиенту 11. Планирование графика работ над проектами и оптимизация процессов для достижения максимального результата	1. Мотивация команды 2. Демонстрация результатов проекта 3. Переговоры 4. Брифование клиента, составление ТЗ, составление декомпозиции задач 5. Работа с рисками проекта 6. Ведение бюджета проекта 7. Масштабирование проекта - Upsale 8. Тайм менеджмент 9. Лидерство 10. Решение конфликтных ситуаций в команде и с заказчиком 11. Параллельное ведение нескольких проектов 12. Проведение митингов с командой 13. Ведение истории проекта и описание use case 14. Знание английского B2//C1 15. Управление интеграцией проекта	1. Delivery Management 2. Release Management 3. Менторинг, обучение других ПМов, создание ПМ офиса, ведение программ и портфелей проекта 4. Сертификация и повышение квалификации//участие в конференциях – трансляция опыта и передача best-практик//Sharing знаний 5. Знание английского B2//C1

	12. Планирование ресурсов на проект, просчет рентабельности проекта, утилизации ресурсов 13. Постановка задач и сроков, контроль исполнения и качества 14. Составление еженедельной отчетности по каждому проекту 15. Развитие внутренних проектов приложения, сайты, блоги, web3...		
Tech Background Hard Skills	1. Работа с оценками перед началом проекта 2. PSM I, PSM II, PSM III сертификация 3. Курс Ивана Селиховкина 4. Уверенное владение одним из таск трекеров (JIRA, Asana, Trello) 5.	6. Построение прототипов в Figma 7. Умение отправлять запросы через Postman 8. Умение работать с GitLab, GitHub (знание CI/CD) 9. OTRS, Redmine, Asana, Miro, Figma, Google Docs 10. Владением фреймворками и методологиями. Опыт в построении CJM, описание buyer persona, user stories, проведение конкурентного анализа, jobs-to-be-done, design sprint, usability testing and audit 11. Навыки проектирования интерфейсов в след программах: Figma, Moqups, Marvel	1. Знание одного языка программирования 2. Понимание проектных методологий (Agile/Kanban/Scrum) - умение выстраивать процесс с нуля в проекте 3. Опыт работы в продуктовой команде (тестирование гипотез, создание MVP, продуктовая аналитика, пивотинг, unit экономика, ICE/RICE, Go-to-market strategy) 4. Организация работы большой распределенной команды 5. Разработка стратегии развития проектов и ее адаптация совместно с командой 6. Самостоятельное принятие решений по процессам и команде внутри своей зоны ответственности 7. Планирование разработки и контроль выполнения планов 8. Формирование процессов производства фичей и контента в соответствии с потребностями продукта 9. Развитие, обучение и оценка команды

			менеджеров и ключевых сотрудников разработки 10. Участие в процессе найма и адаптации сотрудников 11. Распространение успешного опыта внутри своей команды и в компании 12. Технический бекграунд и знание принципов и процессов разработки ПО 13. Опыт составления требований к разработке ПО и интеграционных решений 14. Опыт проектирования SOAP- и REST-сервисов 15. Знание нотаций UML/BPMN, инструментов моделирования процессов (Visio, Draw.io) 16. Опыт написания SQL–запросов, работа с ELK
Personal Traits	1. Аналитическое//Системное мышление 2. Критическое мышление 3. Проактивность 4. Стрессоустойчивость 5. Адаптивность 6. Гибкость мышления	1. PMP сертификация 2. Решительность//Умение брать ответственность на себя 3. Компетентность//Экспертиза в вопросах управления, личный бренд	1. Умение выстраивать процессы с нуля 2. Самоходность, зрелость в вопросах управления

Траектории развития в смежные области для PM

1. Business Developer

2. Product Owner
3. Project Delivery Manager
4. Product Manager = Project manager, который управляет проектом=продуктом
5. Product Owner это более высокая позиция которая расширяется и маркетинговой составляющей, ресерчем, test&try гипотезами и т.п.
6. Chief Product Owner – это уже ответственность за экосистему продуктов или целый ряд линейку продуктов

Вариативно, в зависимости от компании, сферы и рынка, у IT PM могут быть следующие карьерные линии:
1. Project manager - program manager - Portfolio Manager - Head of PMO
2. Project manager - Delivery Manager - Delivery Director
3. Project manager - Product manager - Product Owner (не путать с ролью в скрам) - Chief product owner.
4. **Операционные руководители** вроде COO//CIO//CTO// CEO.
5. **BDM** (Business Development Manager)

Мои контакты по всем вопросам:

- **LinkedIn:** https://www.linkedin.com/in/nadia-firsova/
- **Telegram:** https://t.me/nadia_fir
- **Мой Telegram-канал:** https://t.me/pro_pm