Что такое разработка

Разработка программного обеспечения

Разработка программного обеспечения (англ. software development) — деятельность по созданию нового программного обеспечения. Разработка программного обеспечения как инженерная дисциплина является составной частью (областью) программной инженерии, наряду с дисциплинами, отвечающими за функционирование и сопровождение программных продуктов.

Разработка ПО является очень емкой и многогранной отраслью, которая требует от своих участников понимания множества дисциплин и областей знаний, для успешной реализации поставленных задач. Данные задачи исходят чаще всего от заказчиков, руководителей и менеджеров проектов. Менеджеры проектов в свою же очередь также получают эти задачи от кого-то, либо генерируют их самостоятельно.

Тут важно отметить, что генерация этих задач чаще всего происходит со стороны целей бизнеса или самого проекта. Далеко не всегда разработка программного обеспечения является просто частью улучшения уже существующего программного обеспечения или же просто созданием нового ПО ради его создания.

IT движется бизнесом, а бизнес движется с помощью IT. В свою очередь, сформированные при данном симбиозе задачи имеют ряд минусов и ограничений по отношению к разработке. Данные задачи могут быть неточны и не ясны. Они могут быть размыты в своей формулировке, что сделает невозможным их делегирование. Такие задачи также могут быть очень требовательны и нереализуемы тем или иным специалистом, или выбранным стеком технологий.

Как менеджер может определить, как и когда, и каким образом ему нужно детализировать задачу? На сколько нужно ее детализировать, и

кому ее адресовать? Как понять, что задача выполнена корректно и как убедиться что после ее выполнения производственный процесс идет в своем русле? Ну и наверное самый главный вопрос – как менеджеру это понять, не имея за плечами 3-5 лет программной разработки?

Ответ очень простой – никак. Никак, если менеджер не начнет в этом разбираться и стараться понять, пусть даже верхнеуровнево, как работает его команда и его проект. Насколько велик этот верхний уровень? Достаточно для того, чтобы при его осознании позволить менеджеру видеть не только перспективу разработки, но и возможность давать рекомендации разработчикам, по использованию тех или иных решений, для улучшения процесса на все проекте.

Мышление менеджера может идти вглубь разработки и это лишь укрепит его позиции, но увы далеко не всегда на это будет хватать и времени и сил. И далеко не всегда познание какой-то из технологий досконально позволит менеджеру увидеть всю картину целиком.

Понимание менеджера должно быть стратегическим и тактическим, он обязан видеть весь процесс целиком и все перспективу в целом. Данное видение может появиться лишь при желании вовлечься во все сферы создания проекта. Данное желание приведет менеджера на дорогу где вопросов будет больше чем ответов. Мы надеемся, что в данной тетради вы найдете ответы на многие из них. Желаем успехов.

Элементы процесса разработки

Если говорить о процессе разработки, его можно разбить на такие этапы:

- Сбор требований и подготовка

- Предварительная оценка
- Прототипирование
- Проектирование
- Подробная оценка
- Создание дизайна
- Настройка среды разработки
- Разработка back-end
- Разработка front-end
- Тестирование
- Внесение изменений и/или исправлений
- Документирование
- Перенос проекта в рабочую среду
- Внедрение
- Сопровождение

Ряд подобных элементов может повторяться на проекте по несколько раз, а именно такие как внесение изменений и исправлений после тестирования.

Оценка и дооценка может осуществляться по мере поступления запросов от Заказчика на изменения в проекте. Документирование проекта желательно вести в процессе всего проекта, в тот же момент нежелательно начинать разработку без проектирования и создания дизайнов.

Часть элементов должна идти последовательно, часть будет повторяющимися, а часть может вообще отсутствовать на том или ином проекте - это элементы как внедрение и/или сопровождение.

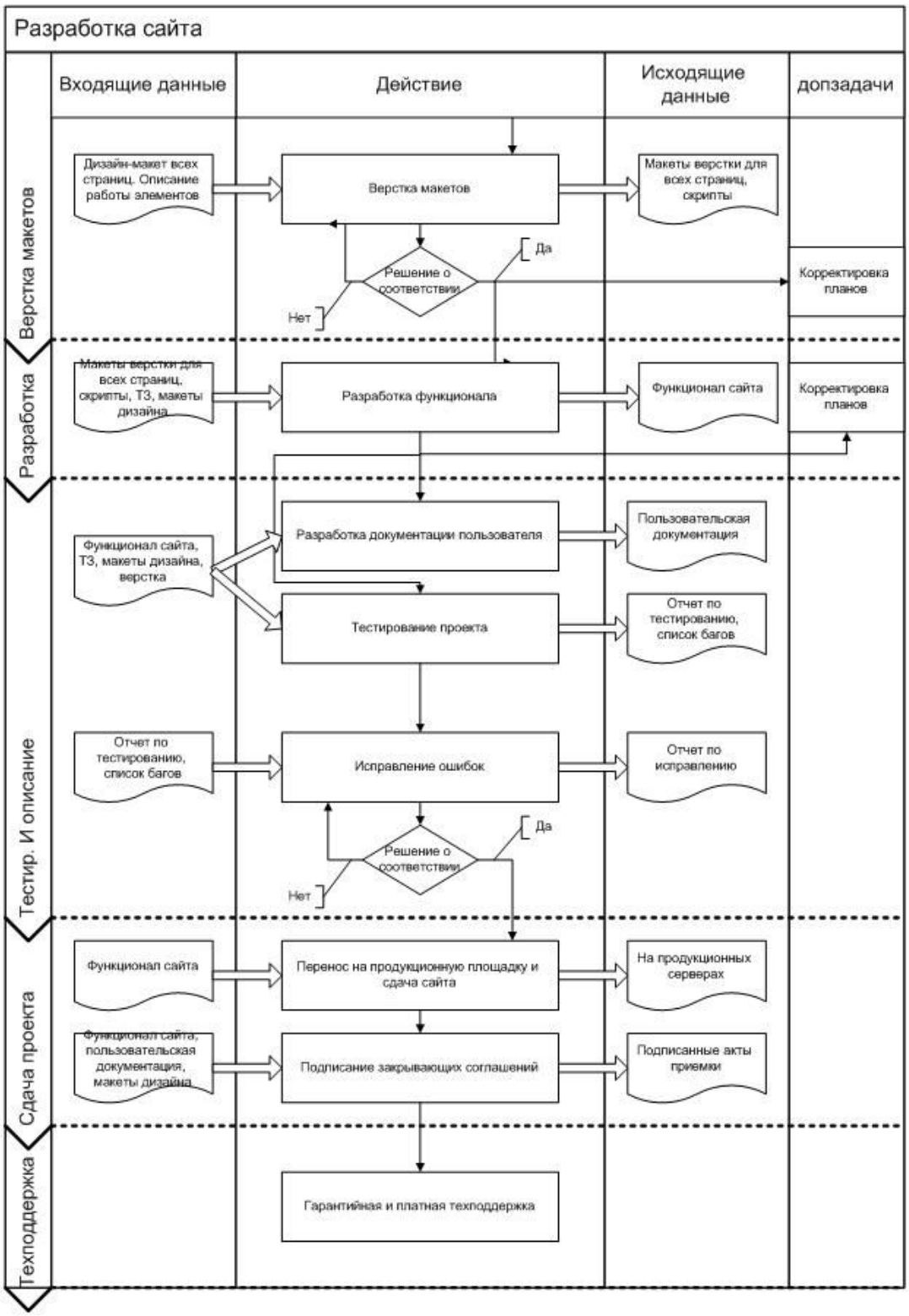
Так или иначе каждый из указанных элементов выше важен и может включать в себя различные группы задач и подзадач, детализация которых как раз и позволит команде успешно завершить проект.

В рабочей тетради внимание будет сконцентрировано в первую очередь на основных элементах процесса разработки проекта, а именно:

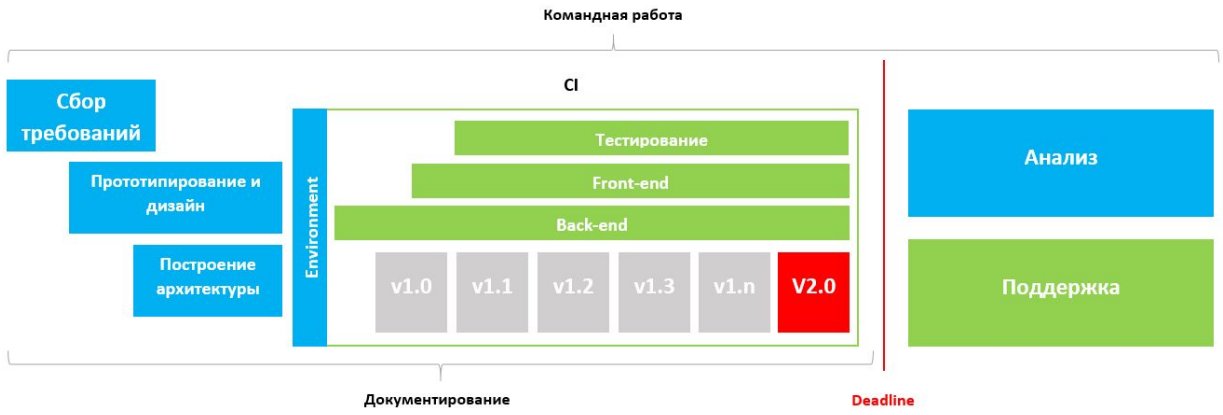
- Правильный сбор требований и формирование задач.
- Прототипирование и проектирование на базовом уровне.
- Как и какую среду разработки нужно использовать.
- Настройка и работа с CI.
- Что относиться к front-end и что включает в себя данный процесс разработки.
- Что относиться к back-end и что включает в себя данный процесс разработки.
- Тестирование во всех своих проявлениях.
- Документирование проекта с технической точки зрения.
- Правильная организация командной работы с точки зрения разработки.
- Сопровождение проекта, его анализ и дальнейшее развитие.

Схема процесса разработки

Для правильной визуализации всего процесса разработки необходимо представить его в виде последовательной схемы процессов и элементов, вот пример разработки сайта.


Разработка сайта
Входящие данные
Действие
Исходящие данные
допзадачи
Верстка макетов
Разработка
Тестир. И описание
Сдача проекта
Техподдержка
Дизайн-макет всех страниц. Описание работы элементов
Верстка макетов
Макеты верстки для всех страниц, скрипты
Решение о соответствии
Да
Нет
Корректировка планов
Макеты верстки для всех страниц, скрипты, ТЗ, макеты дизайна
Разработка функционала
Функционал сайта
Корректировка планов
Функционал сайта, ТЗ, макеты дизайна, верстка
Разработка документации пользователя
Пользовательская документация
Тестирование проекта
Отчет по тестированию, список багов
Отчет по тестированию, список багов
Исправление ошибок
Отчет по исправлению
Решение о соответствии
Да
Нет
Функционал сайта
Перенос на продукционную площадку и сдача сайта
На продукционных серверах
Функционал сайта, пользовательская документация, макеты дизайна
Подписание закрывающих соглашений
Подписанные акты приемки
Гарантийная и платная техподдержка

Схема, указанная выше, включает в себя многие элементы, но не дает подробной информации о том, что именно происходит в процессе планирования или прототипирования сайта, а также не раскрывает базовых положений процесса разработки. Данную схему можно дополнить следующими элементами и представить ее уже в другом виде.



Эту схему также следует дополнить несколькими чек-листами, которые позволят понять, что является базовым входом и базовым выходом одного или другого процесса. Нужно также обратить внимание на то, что данный список входов и выходов не является конечным и может быть дополнен или скомбинирован в другом порядке, в зависимости от требований проекта.

Сбор требований

Сбор требований — процесс подразумевающий получение и формирование документации, которая позволит четко определить что должно быть сделано на проекте, что является продуктом проекта и какие выдвигаются требования к этому продукту.

Входы:

- Бизнес-концепция и идея
- Техническое задание или спецификация
- Описание пользовательских кейсов и целевой аудитории
- Конкурентный анализ и анализ решений-аналогов
- Ограничение, риски и возможности со стороны заказчика

Выходы:

- Техническое задание на разработку
- Базовый план проекта
- Устав проекта и контрактная документация

Прототипирование

Прототипирование — это процесс, основной целью которого является создание визуальной модели продукта проекта, модели настолько подробной, насколько это нужно, чтобы можно было определить ошибки в логике и визуализации идеи.

Входы:

- Входы в процесс сбора требований
- Техническое задание на разработку
- Сессии с заказчиком для формирования общего видения

Выходы:

- Описание пользовательских сценариев со стороны UI/UX

- Прототип согласно оговоренных форматов (wireframe, mock-up, etc.)
- Внесение изменений в техническое задание

Дизайн

Дизайн — основным результатом данного процесса является готовый к перенесению в производство дизайн, который также будет выступать частью документации проекта.

Входы:

- Прототип
- Описание пользовательских сценариев со стороны UI/UX
- Часть ТЗ, касающаяся дизайна

Выходы:

- Готовый дизайн в макетах, согласно стандартов front-end разработки
- Style guide и текстовые рекомендации к нему

Построение архитектуры

Построение архитектуры — этот процесс является основополагающим для создания любого программного продукта, так как он закладывает основу для дальнейшего масштабирования продукта, возможности добавления новых функций, и перестройки продукта согласно новых требований или технологий.

Входы:

- Техническое задание на разработку
- Сессии с заказчиком для формирования общего видения
- Четкие требования к возможной нагрузке, масштабируемости и гибкости системы, а также к стеку технологий

Выходы:

- Описание планируемой архитектуры приложения, включающее в себя все элементы и инфраструктуру
- Описание выбранного стека технологий, стандартов кодирования и процесса разработки
- Дополнения к техническому заданию на разработку
- Визуализация архитектуры
- Подробная оценка проекта и переоценка базового плана

Environment или настройка рабочей среды

Environment или настройка рабочей среды — данный процесс может быть как очень простым, так и очень сложным, в зависимости от модели, которая была выбрана как основа для производства (Continuous Integration / Continuous Delivery). Данный процесс подразумевает создание такой рабочей среды, которая позволит максимально автоматизировать все процессы разработки и минимизировать влияние человеческого фактора.

Входы:

- Часть ТЗ, касающаяся требований к процессу разработки, стеку технологий и необходимому ПО

- Стандарты разработки проектов в компании
- Доступное программное обеспечение (платное и бесплатное)

Выходы:

- Выбор процесса разработки и парка программного обеспечения
- Распределение ролей и ответственных среди членов команды
- Определение правил взаимодействия с автоматизированным процессом
- Настроенные сервера и другое “железо”
- Наличие необходимых инструментов для мануального тестирования

Continuous Integration (CI)

Continuous Integration (CI) — один из самых технологичных процессов, определяющий темп и ритм всего проекта, также данный процесс ставит для себя целью ускорение поставок продукта проекта заказчику с минимальной потерей качества. Мы еще рассмотрим его подробней на следующих занятиях. Пока даем общую информацию по процессу.

Входы:

- Выбранное ПО и правила работы с ним
- Стандарты компании или проекта, касающиеся требований к скорости поставки новых версий продукта и его тестирования
- Определение систем контроля версий и автоматизированного тестирования

Выходы:

- Настройка процесса CI, согласно выбранных стандартов и доступного программного обеспечения
- Определение расписания поставок и характеристик их качественности
- Автоматизация процесса контроля поставок и общая синхронизация всех членов команды через инструменты управления проектом

Back-end

Back-end — процесс разработки серверной части любого приложения, включающий в себя использование серверных технологий для обеспечения проекта работоспособной базой данных и серверной логикой.

Входы:

- Архитектура проекта
- Техническое задание на разработку
- Настроенный environment

Выходы:

- Первые поставки проекта, до старта front-end
- Готовая серверная часть, согласно требований стандартов архитектуры и API
- База данных и системы ее масштабирования, клонирования, адаптации к нагрузкам и прочее
- Интеграция с третьесторонними сервисами
- Техническое описание проекта со стороны back-end

Front-end

Front-end — процесс разработки визуальных элементов взаимодействия приложения с пользователями, также может содержать в себе дополнительную логику и хранение данных.

Входы:

- Дизайн и описание пользовательских сценариев со стороны UI/UX
- Style guide
- Техническое задание на разработку
- Техническое описание проекта со стороны back-end

Выходы:

- Реализованные интерфейсы приложения и часть логики работы, которая должна быть на front-end
- Готовность интерфейсов к адаптации на разные платформы (mobile)
- Синхронизация back-end с front-end напрямую или через API
- Готовность проекта к тестированию

Тестирование

Тестирование — этот процесс отвечает за качество и стабильность работы программного решения и ставит для себя основной целью поставку программного обеспечения не только согласно поставленных требований, но и согласно выбранных стандартов качества.

Входы:

- Вся существующая проектная документация для возможности тестирования любой внешней логики с помощью интерфейсов
- Доступы в environment
- Выбор типа тестирования и инструментов

Выходы:

- Тест-план или чек-листы
- Тестовые сценарии
- Отчеты по тестированию
- Автоматические тесты
- Рекомендации к улучшению и защите продукта

Анализ и поддержка

Анализ и поддержка — данный процесс может уже не входить в проектную деятельность, но сам по себе важен тем, что может показать как используется продукт проекта конечными пользователями, дать основу для новых доработок и показать изъяны созданных решений.

Входы:

- Исходные коды и доступы
- Готовый продукт проекта
- Настроенные сервера и бекапирование
- Вся существующая проектная документация
- Настроенная защита программного решения
- Системы аналитики работы пользователей с приложением

- Контрактные документы на поддержку

Выходы:

- Обратная связь от пользователей
- Аналитические данные по работе сервиса
- Список необходимых или желательных доработок
- Масштабирование системы при нагрузке

Пошаговый сценарий

Если вышеуказанные чек-листы представить как пользовательский пошаговый сценарий с ролями возможных участников, тогда это может выглядеть следующим образом.

Менеджерский состав с ведущими разработчиками собирают всю возможную информацию и документацию от заказчика и его доменного рынка.

Полученная информация перерабатывается в техническое задание на разработку, которые уже подвергается оценке, и на основании которого подписывается контракт.

Кроме основного контракта, также возможен и предварительный, целью которого будет создать MVP или прототип проекта, для правильного формирования задач уже к всему проекту в целом.

Этап прототипирования выполняется дизайнером или UI/UX специалистом для визуализации целей и идей заказчика, а также детализации возможных сценариев работы пользователей с сервисом. На данном этапе также очень важно, чтобы творческие

порывы дизайнера были совместимы с техническими возможностями реализации в команде разработки.

После подготовки прототипа и его согласования готовится следствие дизайн. Дизайн должен соответствовать всем определенным требованиям к функционалу и стандартом front-end разработки. Дизайнер после данной активности может еще понадобиться на этапе тестирования.

Далее, или в самом процессе, идет разработка архитектуры решения ведущими специалистами команды или компании, дабы определить единый стандарт создания приложения. Разработка архитектуры подразумевает не только мыслительный процесс, но и строгое документирование результатов в виде описание структуры архитектуры, инфраструктуры сервиса и его визуализации с помощью диаграмм и графиков.

Как только архитектура готова, дальше ведущие специалисты, вместе с тестировщиками и системными администраторами (DevOps) определяют процесс разработки, необходимое программное обеспечение, и стараются максимально автоматизировать действия по поставке готового продукта.

Если процессным решением было выбрана CI, тогда следующим шагом идет настройка всех необходимых составляющих CI системными администраторами (DevOps). В этот процесс также напрямую вовлекаются back-end и front-end разработчики. CI требует также тщательной документации последовательности шагов, так как он не только ускоряет поставку версий продукта, но и упрощает масштабирование команды и обучение новичков.

Далее начинается разработка, в которой обычно back-end слегка опережает front-end, но в целом эти оба процесса идут параллельно. В

процессе разработки три составляющие - это back, front и CI - идут нога в ногу, и являются в принципе самым главным создающим звеном проекта.

Как только со стороны front-end готов первый интерфейс синхронизированный с back-end, можно приступать к тестированию. Тестирование также требует от тестировщиков полного погружения в проект, причем желательно сначала через документацию, а потом уже через мануальное и QA.

Процесс тестирования также очень последователен и требует от тестировщиков сначала составления тест-плана или чек-листов, по которым впоследствии они смогут проводить регрессию найденных и решенных ошибок. После чего они могут приступать к мануальному тестированию.

После исследования проекта мануально, могут быть написаны авто-тесты, которые уже совместно с CI могут сами проверять как правильно проходят поставки продукта со стороны front-end и back-end.

После подключения тестировщиков в производственном процессе будет уже четыре элемента, которые должны двигаться в едином ритме. Отдельным типом взаимодействия также следует отметить работу тестировщиков вместе с разработчиками. Процесс нахождения и подтверждения, а после исправления ошибок, требует максимальной командной работы. Отладка системы перед последним релизом также будет требовать максимального внимания от всей команды.

В течении всех этих процессов менеджерским и техническим составом должна вестись проектная документация.

Визуализация и прототипирование в проекте

Сценарий использования, вариант использования, прецедент использования (англ. usecase) — в разработке программного обеспечения и системном проектировании это описание поведения системы, когда она взаимодействует с кем-то (или чем-то) из внешней среды. Система может отвечать на внешние запросы Актёра (англ. actor) (может применяться термин Актант), может сама выступать инициатором взаимодействия. Другими словами, сценарий использования описывает, «кто» и «что» может сделать с рассматриваемой системой, или что система может сделать с «кем» или «чем». Методика сценариев использования применяется для выявления требований к поведению системы, известных также как пользовательские и функциональные требования. Также в ссылках ниже представлены форматы создания и описания use cases (user story) и user flow, и примеры того, как объяснять их заказчику: документация, которая в итоге будет передана заказчику.

Процесс сдачи проекта в эксплуатацию и перевод на поддержку подразумевает, что в приложении установлены системы аналитики для сбора данных о возникающих ошибках, при работе конечных пользователей системой, вся проектная документация и исходные коды сданы заказчику, и сам продукт проекта определен, как соответствующий изначальным требованиям.

Шаги описанные выше, выступают как сценарий работы членов команды с проектом, также существует понятие пользовательского сценария работы с продуктов проекта.

1. Как создать user story?
2. Scenarios

3. Как объяснить заказчику, что такое пользовательские сценарии?

Еще одной полезной ссылкой является подбор статей из сервиса Habrahabr - рекомендуется к постепенному и последовательному изучения.

Мы еще будем сталкиваться с созданием user story в лекциях по документации. Пока советуем просто ознакомиться с процессом в общих терминах.

Также, еще одним хорошим примером работы с визуализацией ожидаемого результата проекта является прототип. Прототипирование разделено на несколько видов и для его реализации используются различные инструменты и методики. Вот также несколько полезных ссылок.

1. Что такое прототипирование?
    - Вайрфреймы, прототипы и мокапы
    - О бедном мокапе замолвите слово
    - 16 инструментов для создания прототипов
2. Как создавать прототипы?
    - Простое создание прототипа сайта №1
    - Простое создание прототипа сайта №2
3. Какие инструменты можно использовать?
    - Ninjamock
    - Moqups
    - Axure
    - InVisionApp
    - Balsamiq

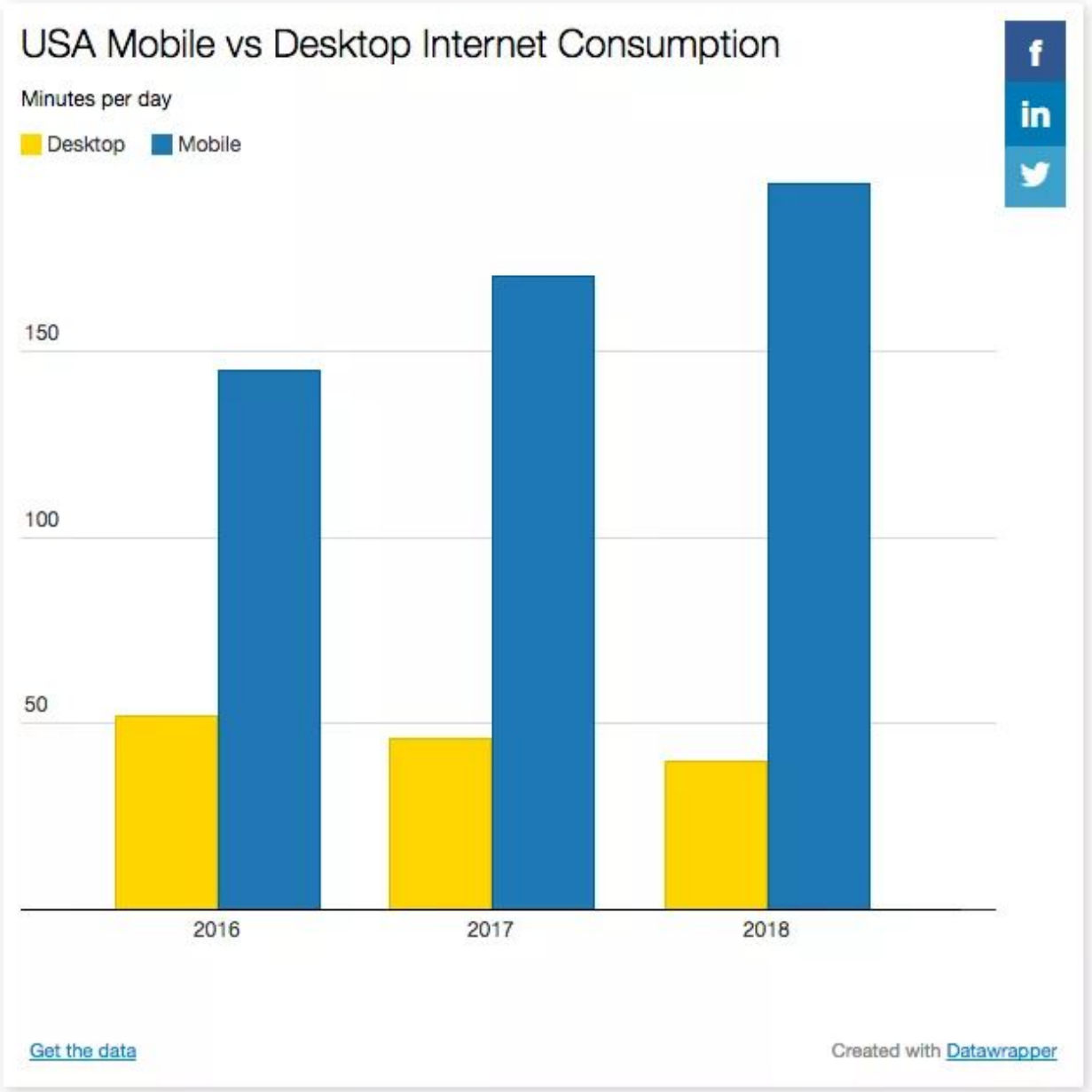
Тренды по технологиям

Технологии развиваются и этот процесс влияет на всю мировую экономику. На сегодняшний день нам пророчат, что уже через 2 десятка лет многие профессии вымрут из-за повального внедрения искусственного интеллекта, а количества интернет пользователей почти в полтора раза.

PWC прогнозирует, что к 2050 году:

- Мировая экономика увеличится вдвое.
- Развивающиеся страны могут вырасти примерно в два раза быстрее, чем рынки стран с развитой экономикой.
- В результате, шестью из семи крупнейших экономик в мире, по прогнозам, будут страны с развивающейся экономикой во главе с Китаем, Индией и Индонезией.
- США может опуститься на третье место в мировом рейтинге ВВП, а доля ЕС может упасть ниже 10%.
- Мексика, Турция и Вьетнам могут перегнать Великобританию, Италию и Францию.

Кроме того, пользователи с каждым годом проводят в интернете все больше времени. Интернет становится, в первую очередь, мобильным.


USA Mobile vs Desktop Internet Consumption
Minutes per day
Desktop
Mobile
150
100
50
2016
2017
2018
Get the data
Created with Datawrapper

Web

Сейчас web-технологии вырываются вперед. Web начинает постепенно вытеснять Desktop благодаря скорости интернета и появлению Single Page Applications.

Устойчивые тренды в веб-дизайне:

- Веб-дизайн в традиционном смысле умирает, теперь это проектирование интерфейсов, основанное на User Experience. Веб-дизайнеры (или UI/UXспециалисты) уже не просто стараются сделать веб-сайты "красивыми". Скорее, они изучают опыт пользователей и их поведение и формируют Flow.
- Все большую популярность набирают боты и “разговорный” UI. Люди начинают общаться с компаниями через чаты и мессенджеры как с людьми.
- GIF-изображения и анимации. Множество веб-сайтов используют анимированные изображения и украшают ими свой интерфейс.
- Адаптивная верстка становится не просто желательной, но обязательной. Google поменял алгоритм ранжирования и теперь может сильно понижать видимость сайтов с плохим отображением на мобильных устройствах.
- Минимализм. Меньше текста больше изображений и видео.
- Визуализация данных и инфографики. Такой контент набирает большую популярность и часто становится виральным.
- Использовать стоковые изображения уже не модно. Компании все чаще создают свой собственный визуальный контент, используя реальные фотографии клиентов, сотрудников, офисов или просто съемку товара.
- Material Design. Он был разработан Google и закрепился во всех приложениях, таких как Gmail, Google Maps, Google Drive, и YouTube. Теперь он популярен повсеместно и продолжает набирать обороты.

- Веб-сайты с длинным или бесконечным скролом. Это стало популярно благодаря специфическому пользовательскому опыту. Пользователи могут пролистывать такие сайты часами, смотря на постоянно обновляющийся контент.
- Брендинг становится заметней. Компании больше не стесняются выпячивать свое название и громко заявлять о себе.

Mobile

На сегодняшний день мобильного трафика становится все больше, однако ситуация в борьбе основных рынков приложений Android и Apple не поменялась. Android по доброй традиции пытается брать количеством, забывая о качестве. Однако с ростом рынков Азии ситуация в скором времени может поменяться.

Устойчивые тренды в мобильной разработке:

- Дополненная реальность набирает обороты. Появление Pokemon Go стимулировало интерес к этим технологиям и теперь он не угасает.
- Android завоевывает популярность. Google улучшил руководство для разработчиков и безопасность операционной системы. Некоторые эксперты ожидают, что к 2020 году Google Play будет опережать App Store как по числу установок приложений, так и с точки зрения монетизации.
- Невидимые или скрытые меню. Они не являются каким-то новшеством. К этому давно все шло, просто настало время, когда этот тренд сможет проявиться в полную силу. Дизайнеры, наконец, уже готовы принять этот как данность и начать использовать скрытые меню – ведь такой подход соответствует ожиданиям пользователя.
- Корпоративные мобильные приложения. Тенденция Bring Your Own Device меняется. Наконец, компании стали понимать, что их

существующее корпоративное ПО не служит повышению производительности, тем более сокращению эксплуатационных расходов. Поэтому спрос на корпоративные мобильные приложения будет расти.
- Легковесные приложения. На наши смартфоны загружены десятки приложений, и мы хотим расширять их список. Неудивительно, что память устройства быстро заканчивается. Вывод напрашивается сам - нужно больше места для загрузки.

Desktop

Десктоп-приложения постепенно сдают позиции, хотя все еще популярны для работы бизнеса, в первую очередь, больших компаний.

Устойчивые тренды в разработке desktop приложений:

- Desktop приложения продолжают терять популярность.
- Не смотря на это в B2B сегменте desktop все еще в моде для больших корпораций.
- Open Source набирает обороты.
- Микросервисная архитектура становится все более популярной.

Экстремальное программирование

Экстремальное программирование (XP) – это упрощенная методология организации разработки программ для небольших и средних по размеру команд разработчиков, занимающихся созданием программного продукта в условиях неясных или быстро меняющихся требований.

Основными целями XP являются повышение доверия заказчика к программному продукту путем предоставления реальных

доказательств успешности развития процесса разработки и резкое сокращение сроков разработки продукта. При этом XP сосредоточено на минимизации ошибок на ранних стадиях разработки. Это позволяет добиться максимальной скорости выпуска готового продукта и дает возможность говорить о прогнозируемости работы. Практически все приемы XP направлены на повышение качества программного продукта.

Основные принципы:

- Итеративность. Разработка ведется короткими итерациями при наличии активной взаимосвязи с заказчиком. Итерации как таковые предлагается делать короткими, рекомендуемая длительность – 2-3 недели и не более 1 месяца. За одну итерацию группа программистов обязана реализовать несколько свойств системы, каждое из которых описывается в пользовательской истории.
- Простота решений. Принимается первое простейшее рабочее решение. Экстремальность метода связана с высокой степенью риска решения, обусловленного поверхностностью анализа и жестким временным графиком.
- Интенсивная разработка малыми группами (не более 10 человек) и **парное программирование** (когда два программиста вместе создают код на одном общем рабочем месте), активное общение в группе и между группами.
- Обратная связь с заказчиком, представитель которого фактически вовлечен в процесс разработки.

Правила XP

Обычно XP характеризуют набором из 12 правил (практик), которые необходимо выполнять для достижения хорошего результата. Ни одна

из практик не является принципиально новой, но в XP они собраны вместе:

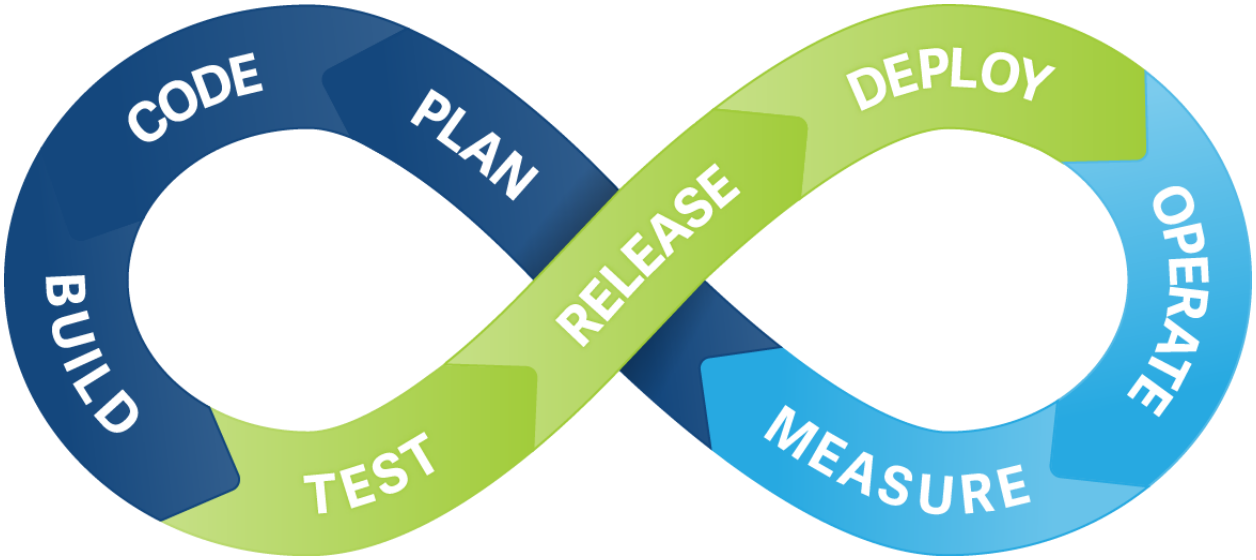
- Планирование процесса. Вся команда разработчиков собирается вместе, принимается коллективное решение о том, какие свойства системы будут реализованы в ближайшей итерации.
- Тесное взаимодействие с заказчиком. Представитель заказчика должен быть членом XP-команды. Он пишет ПИ, выбирает истории, которые будут реализованы в конкретной итерации, и отвечает на вопросы, касающиеся бизнеса.
- Общесистемные правила именования. Хорошие системные правила именования предполагают простоту именования классов и переменных. Команда разработчиков должна иметь единые правила именования.
- Простая архитектура. Любое свойство системы должно быть реализовано как можно проще.
- Рефакторинг. Это оптимизация существующего кода с целью его упрощения. Такая работа должна вестись постоянно.
- Парное программирование. Все программисты должны работать в парах: один пишет код, другой смотрит. Таким образом, необходимо размещать группу программистов в одном месте. XP наиболее успешно работает в нераспределенных коллективах программистов и пользователей.
- 40-часовая рабочая неделя. Программист не должен работать более 8 часов в день. Необходимость сверхурочной работы – это четкий индикатор проблемы на данном конкретном направлении разработки. Поиск причин сверхурочной работы и их скорейшее устранение – одно из основных правил.
- Коллективное владение кодом. Каждый программист в коллективе должен иметь доступ к коду любой части системы и право вносить изменения в любой код. Обязательное правило: если программист внес изменения и система после этого

работает некорректно, то именно этот программист должен исправить ошибки.

- Единые стандарты кодирования. Стандарты кодирования нужны для обеспечения других практик: коллективного владения кодом, парного программирования и рефакторинга. Без единого стандарта выполнять эти практики как минимум сложнее, а в реальности вообще невозможно: группа будет работать в режиме постоянной нехватки времени. Команда работает над проектом продолжительное время. Люди приходят и уходят. Никто не кодирует в одиночку и код принадлежит всем. Следовательно, все должны подчиняться общим стандартам кодирования.
- Небольшие релизы. Минимальная итерация – один день, максимальная – месяц; чем чаще осуществляются релизы, тем больше недостатков системы будет выявлено.
- Непрерывная интеграция. Интеграция новых частей системы должна происходить как можно чаще, как минимум раз в несколько часов. Основное правило интеграции следующее: интеграцию можно производить, если все тесты проходят успешно.
- Тестирование. В отличие от большинства остальных методологий тестирование в XP – одна из важнейших составляющих. Экстремальный подход предполагает, что тесты пишутся до написания кода. Каждый модуль обязан иметь unit test – тест данного модуля. Еще один важный принцип: тест определяет код, а не наоборот (test-driven development), то есть кусок кода кладется в хранилище тогда и только тогда, когда все тесты прошли успешно, в противном случае данное изменение кода отвергается.

Процесс XP является неформальным, но требует высокого уровня самодисциплины. Если это правило не выполняется, то XP мгновенно превращается в хаотичный и неконтролируемый процесс.

Непрерывная интеграция (CI, Continuous Integration) — это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоемкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий.



Преимущества:

- проблемы интеграции выявляются и исправляются быстро, что оказывается дешевле;
- немедленный прогон модульных тестов для свежих изменений;
- постоянное наличие текущей стабильной версии вместе с продуктами сборок - для тестирования, демонстрации, и т. п.

- немедленный эффект от неполного или неработающего кода приучает разработчиков к работе в итеративном режиме с более коротким циклом.

Недостатки:

- затраты на поддержку работы непрерывной интеграции;
- потенциальная необходимость в выделенном сервере под нужды непрерывной интеграции;
- немедленный эффект от неполного или неработающего кода отучает разработчиков от выполнения периодических резервных включений кода в репозиторий. В случае использования системы управления версиями исходного кода с поддержкой ветвления, эта проблема может решаться созданием отдельной «ветки» проекта для внесения крупных изменений (код, разработка которого до работоспособного варианта займет несколько дней, но желательно более частое резервное копирование в репозиторий). По окончании разработки и индивидуального тестирования такой ветки, она может быть объединена с основным кодом или «стволом» проекта.

Code review - инженерная практика в терминах гибкой методологии разработки. Это анализ (инспекция) кода с целью выявить ошибки, недочеты, расхождения в стиле написания кода, в соответствии написанного кода и поставленной задачи.

Очевидные плюсы этой практики:

- Улучшается качество кода.
- Находятся «глупые» ошибки (опечатки) в реализации.
- Повышается степень совместного владения кодом.
- Код приводится к единому стилю написания.
- Хорошо подходит для обучения «новичков», быстро набирается навык, происходит выравнивание опыта, обмен знаниями.

- Эта практика имеет большую социальную значимость, так как предполагает «общественное признание» или «работу над ошибками», в зависимости от качества проверенного кода.