**1) QA - QC – тестировщик**

Обеспечение качества (Quality assurance - QA) — часть управления качеством, направленная на обеспечение уверенности в том, что требования к качеству будут выполнены.

Контроль качества (Quality control - QC) — часть управления качеством, ориентированная на выполнение требований к качеству.

Итого: из этих определений следует, что мы либо обеспечиваем качественный продукт, либо проверяем продукт на соответствие качеству.

Тестировщик – непосредственно тестирует продукт.

**2) Что происходит, когда мы вводим адрес в поисковую строку?**

1. DNS-запрос по имени сайта.
2. Подключение к серверу по IP (ТCP-подключение).
3. Установление защищённого соединения при использовании HTTPS (TLS-подключение).
4. Запрос HTML-страницы по URL и ожидание сервера. (HTTP-запрос)
5. Загрузка HTML.
6. Разбор HTML-документа на стороне браузера, построение очереди запросов к ресурсам документа. Загрузка и парсинг CSS-стилей. Загрузка и выполнение JS-кода. Загрузка веб-шрифтов. Загрузка изображений и других элементов.
7. В этом процессе некоторые позиции происходят параллельно, некоторые могут меняться местами, но суть остаётся той же.

**3) Функционирование компьютерной сети тесно связано с интерфейсом и протоколами.**

Интерфейс — это соглашение о взаимодействии между уровнями одной системы, определяющее структуру данных и способ (алгоритм) обмена данными между соседними уровнями *OSI*-модели.

Протокол — это совокупность правил, регламентирующих формат и процедуры взаимодействия процессов одноимённых уровней на основе обмена сообщениями.

**Протоколы передачи данных:**

1. TCP/IP (Это стек протоколов TCP и IP. Первый обеспечивает и контролирует надёжную передачу данных и следит за её целостностью. Второй же отвечает за маршрутизацию для отправки данных.)
2. FTP (Протокол передачи файлов)
3. DNS (Это не только система доменных имён (Domain Name System), но и протокол, без которого эта система не смогла бы работать. Он позволяет клиентским компьютерам запрашивать у DNS-сервера IP-адрес какого-либо сайта, а также помогает обмениваться базами данных между серверами DNS)
4. HTTP (используется для передачи произвольных данных в интернете. Он является протоколом клиент-серверного взаимодействия без сохранения промежуточного состояния. В роли клиента чаще всего выступает веб-браузер, хотя может быть и, например, поисковый робот. Для обмена информацией протокол HTTP в большинстве случаев использует TCP/IP.) HTTP имеет расширение HTTPS, которое поддерживает шифрование. Данные в нём передаются поверх криптографического протокола TLS.
5. SSH (Протокол для удалённого управления операционной системой с использованием TCP. В SSH шифруется весь трафик, причём с возможностью выбора алгоритма шифрования. В основном это нужно для передачи паролей и другой важной информации.)
6. POP3 — протокол для приема сообщений электронной почты. IMAP — более современный протокол, поддерживающий прием и возможность управления электронной почтой прямо на почтовом сервере.
7. web socket протокол связи поверх tcp-соединения для обмена сообщениями между браузером и веб-сервером в режиме реального времени. Данные передаются по нему в обоих направлениях в виде «пакетов», без разрыва соединения и дополнительных HTTP-запросов. WebSocket особенно хорош для сервисов, которые нуждаются в постоянном обмене данными, например онлайн игры, торговые площадки, работающие в реальном времени, и т.д.

**4) Разница между аутентификацией и авторизацией.**

Идентификация — это заявление о том, кем вы являетесь. В зависимости от ситуации, это может быть имя, адрес электронной почты, номер учетной записи, итд.

Аутентификация — предоставление доказательств, что вы на самом деле есть тот, кем идентифицировались (ввел правильный пароль – доказал, что являюсь тем, кем представился –«идентифицировался»)

Авторизация — проверка, что вам разрешен доступ к запрашиваемому ресурсу (проверка, что вы пользователь, и имеете, например, ограниченные права на сайте, а не администратор, который имеет полные права).

**5) Модель Open Systems Interconnection (OSI)**

<https://www.youtube.com/watch?v=HeluN1cH_uo>

Это скелет, фундамент и база всех сетевых сущностей. Модель определяет сетевые протоколы, распределяя их на 7 логических уровней. Важно отметить, что в любом процессе, управление сетевой передачей переходит от уровня к уровню, последовательно подключая протоколы на каждом из уровней.

Нижние уровни отвечают за физические параметры передачи, такие как электрические сигналы. Токи представляются в виде последовательности единиц и нулей (1 и 0), затем, данные декодируются и маршрутизируются по сети. Более высокие уровни охватывают запросы, связанные с представлением данных. Условно говоря, более высокие уровни отвечают за сетевые данные с точки зрения пользователя.

Модель OSI была изначально придумана как стандартный подход, архитектура или паттерн, который бы описывал сетевое взаимодействие любого сетевого приложения.

**#01: ФИЗИЧЕСКИЙ (PHYSICAL) УРОВЕНЬ**

На первом уровне модели OSI происходит передача физических сигналов (токов, света, радио) от источника к получателю. На этом уровне мы оперируем кабелями, контактами в разъемах, кодированием единиц и нулей, модуляцией и так далее.

Среди технологий, которые живут на первом уровне, можно выделить самый основной стандарт - Ethernet.

Отметим, что в качестве носителя данных могут выступать не только электрические токи. Радиочастоты, световые или инфракрасные волны используются также повсеместно в современных сетях.

Сетевые устройства, которые относят к первому уровню это концентраторы и репитеры, которые могут просто работать с физическим сигналом, не вникая в его логику (не декодируя).

**#02: КАНАЛЬНЫЙ (DATA LINK) УРОВЕНЬ**

При получении физического сигнала с первого на втором уровне проверяются и исправляются ошибки передачи. На втором уровне мы оперируем понятием «фрейм», или как еще говорят «кадр». Тут появляются первые идентификаторы – MAC – адреса. Они состоят из 48 бит и выглядят примерно так: 00:16:52:00:1f:03.

Канальный уровень сложный. Поэтому, его условно говоря делят на два подуровня: управление логическим каналом (LLC, Logical Link Control) и управление доступом к среде (MAC, Media Access Control).

На этом уровне обитают такие устройства как коммутаторы и мосты. Стандарт Ethernet тоже тут, он расположился на первом и втором (1 и 2) уровнях модели OSI.

**#03: СЕТЕВОЙ (NETWORK) УРОВЕНЬ**

Сетевой уровень вводит термин «маршрутизация» и, соответственно, IP – адрес.

Именно на этом уровне происходит маршрутизация трафика, как таковая. Если мы хотим попасть на сайт, то мы отправляем DNS – запрос, получаем ответ в виде IP – адреса и подставляем его в пакет.

Из устройств здесь находится маршрутизатор.

Процесс, когда данные передаются с верхних уровней на нижние называется инкапсуляцией данных, а когда наоборот, наверх, с первого, физического к седьмому, то этот процесс называется декапсуляцией данных

**#04: ТРАНСПОРТНЫЙ (TRANSPORT) УРОВЕНЬ**

Транспортный уровень обеспечивает передачу данных по сети по TCP и UDP. Разница в том, что различный транспорт применяется для разной категории трафика.

**#05: СЕАНСОВЫЙ (SESSION) УРОВЕНЬ**

Сеансовый уровень занимается тем, что управляет соединениями, или попросту говоря, сессиями. Он их разрывает.

**#06 УРОВЕНЬ ПРЕДСТАВЛЕНИЯ (PRESENTATION)**

На шестом уровне происходит преобразование форматов сообщений, такое как кодирование или сжатие. Так же уровень ответственен за передачу потока на четвертый (транспортный уровень).

**#07 УРОВЕНЬ ПРИЛОЖЕНИЯ (APPLICATION)**

Седьмой уровень - уровень приложений. Тут находятся сетевые службы, которые позволяют нам, как конечным пользователям, серфить просторы интернета.



**6) Клиент-серверная архитектура:**

**https://habr.com/ru/post/495698/**

Как мы знаем, клиент-серверная архитектура позволяет совершать разнообразные операции на просторах Интернета. В подобной схеме взаимодействия можно выделить 2 базовые стороны: клиент (заказчик) и сервер (поставщик услуг).

Клиентом в данном случае может быть любое программное обеспечение, которое имеет подключение к сети, отправляет сформированные запросы и обрабатывает ответы, исходящие от сервера.

Клиентами могут выступать:

1. веб-браузеры;
2. мобильное ПО;
3. умная встроенная бытовая техника;
4. разнообразные серверные приложения.

Сервер – виртуальная машина, которая имеет подключение к сети. Ее базовая задача – получение запросов, их обработка и отправка клиенту данных в определенном формате.

К серверам причисляют:

1. Локальный веб-сервер;
2. Любой сервер на основе HTTP;
3. Специализированные наборы серверных машин.

Клиент с сервером обмениваются между собой сообщениями по типу «запрос-ответ». Их принято именовать сетевыми протоколами.

Веб-браузер начинает диалог с сервером на базе HTTP-протокола, отправляя запрос, в котором указано, какие именно данные и какого типа он ожидает получить на вход. Сервер, сразу же после получения запроса и его обработки, направляет браузеру определенное сообщение (так называемый HTTP-ответ), который содержит запрошенную информацию (как правило, это HTML документ). Подобные сообщения содержат данные и специальные коды текущего состояния, по которым веб-браузеры могут определять, успешно ли был обработан входящий запрос.

Конечный итог обработки запроса, который всегда содержится в 1 строке ответа сервера, именуется кодом ответа или кодом текущего состояния. Данные сообщения отправляются каждый раз при взаимодействии веб-браузера и сервера, даже если пользователю сайта они не видны.

**ВИДЫ КОДОВ ОТВЕТОВ**

Все коды ответов можно разделить между собой на определенные классы. Класс состояния обозначается 1 цифрой, которая должна определять состояние веб-страницы, две остальные цифры носят уточняющий характер.

**Выделяют 5 классов кодов по состоянию:**

1хх – группа информационных кодов. Их задача состоит в обработке информации, создании сообщения о том, что текущий запрос принят, и передаче данных.

2хх – успешная обработка запроса. Запрос уже получен сервером и успешно выполнен без каких-либо ошибок.

3хх – перенаправление (англ. redirect). Данный код сообщает о текущей необходимости отправить запрос по иному адресу при смене расположения нужного файла.

4хх – ошибки на стороне пользователя. Такие ошибки могут указывать на то, что данный файл по запрашиваемому адресу не найден.

5хх – ошибки на стороне сервера. При выполнении запроса случился сбой сервера.

Верификация и валидация:

Верификация – подтверждение на основе объективных предоставленных фактов того, что установленные нормы были выполнены. (Продукт выполняет ту функцию, что условно описана в Т.З.). Делаем продукт правильно (по Т.З.).

Валидация – подтверждение на основе объективных предоставленных фактов того, что установленные нормы для конкретного применения выполнены (продукт выполняет нужную функцию, но не так, как того хотел заказчик). Делаем правильный продукт (как его видит заказчик, как будет удобно использовать)

Что такое API?

Сообщения по типу запрос-ответ. Типы запросов (get, post, delete, put, options).

Архитектура «Клиент-Сервер»

Определение

Архитектура «Клиент-Сервер» (также используются термины «сеть Клиент-Сервер» или «модель Клиент-Сервер») предусматривает разделение процессов предоставление услуг и отправки запросов на них на разных компьютерах в сети, каждый из которых выполняют свои задачи независимо от других.

В архитектуре «Клиент-Сервер» несколько компьютеров-клиентов (удалённые системы) посылают запросы и получают услуги от централизованной служебной машины – сервера (server – англ. «официант, обслуга»), которая также может называться хост-системой (host system, от host – англ. «хозяин», обычно гостиницы).

Клиентская машина предоставляет пользователю т.н. «дружественный интерфейс» (user-friendly interface), чтобы облегчить его взаимодействие с сервером.

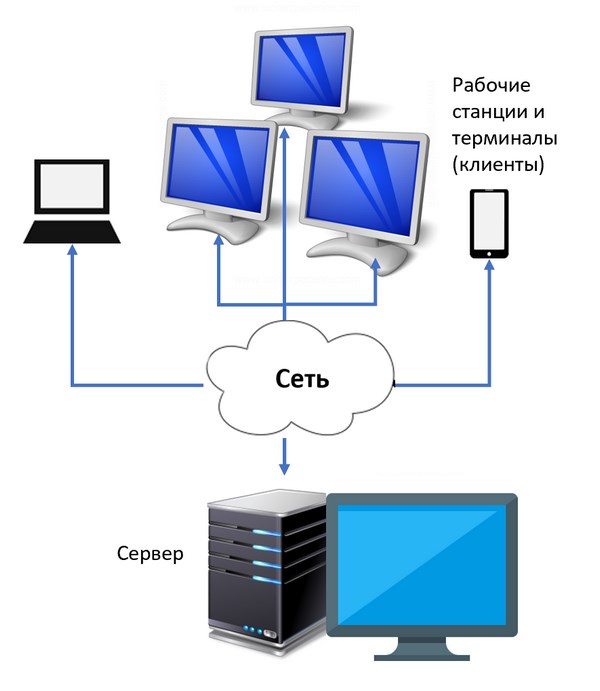


Рис. 1. Архитектура «Клиент-Сервер».

Типы клиент-серверной архитектуры

Архитектуру «клиент-сервер» принято разделять на три класса: одно-, двух- и трёхуровневую. Однако, нельзя сказать, что в вопросе о таком разделении в сообществе ИТ-специалистов существует полный консенсус. Многие называют одноуровневую архитектуру двухуровневой и наоборот, то же можно сказать о соотношении двух- и трёхуровневой архитектур.

Постараемся внести ясность в этот вопрос.

Одноуровневая архитектура (1-Tier)

Одноуровневая архитектура «клиент-сервер» (1-Tier) – такая, где все прикладные программы рассредоточены по рабочим станциям, которые обращаются к общему серверу баз данных или к общему файловому серверу. Никаких прикладных программ сервер при этом не исполняет, только предоставляет данные.

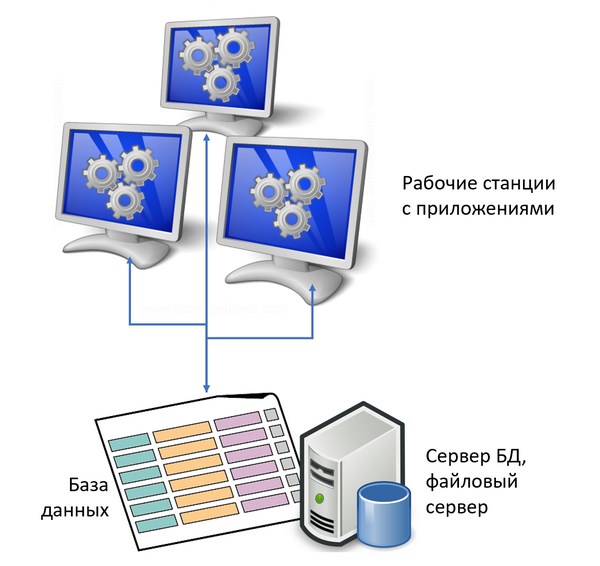


Рис. 2. Одноуровневая архитектура «клиент-сервер» (1-Tier).

В целом, такая архитектура очень надёжна, однако, ей сложно управлять, поскольку в каждой рабочей станции данные будут присутствовать в разных вариантах. Поэтому возникает проблема их синхронизации на отдельных машинах. В общем, как можно видеть из рисунка, в этой архитектуре просматривается ещё один уровень – базы данных, что даёт повод во многих случаях называть её двухуровневой.

Двухуровневая архитектура (2-Tier)

К двухуровневой архитектуре «клиент-сервер» следует относить такую, в которой прикладные программы сосредоточены на сервере приложений (Application Server), например, сервере 1С или сервере CRM, а в рабочих станциях находятся программы-клиенты, которые предоставляют для пользователей интерфейс для работы с приложениями на общем сервере.

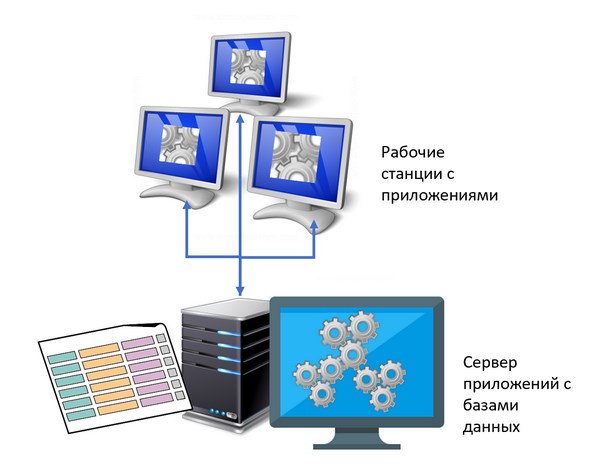


Рис. 3. Двухуровневая архитектура «клиент-сервер» (2-Tier).

Такая архитектура представляется наиболее логичной для архитектуры «клиент-сервер». В ней, однако, можно выделить два варианта. Когда общие данные хранятся на сервере, а логика их обработки и бизнес-данные хранятся на клиентской машине, то такая архитектура носит название “fat client thin server” (толстый клиент, тонкий сервер). Когда не только данные, но и логика их обработки и бизнес-данные хранятся на сервере, то это называется “thin client fat server” (тонкий клиент, толстый сервер). Такая архитектура послужила прообразом облачных вычислений (Cloud Computing).

Преимущества двухуровневой архитектуры:

• Легко конфигурировать и модифицировать приложения;

• Пользователю обычно легко работать в такой среде;

• Хорошая производительность и масштабируемость.

Однако, у двухуровневой архитектуры есть и ограничения:

• Производительность может падать при увеличении числа пользователей;

• Потенциальные проблемы с безопасностью, поскольку все данные и программы находятся на центральном сервере;

• Все клиенты зависимы от базы данных одного производителя;

Трёхуровневая архитектура (3-Tier)

В трёхуровневой архитектуре сервер баз данных, файловый сервер и другие представляют собой отдельный уровень, результаты работы которого использует сервер приложений. Логика данных и бизнес-логика находятся в сервере приложений. Все обращения клиентов к базе данных происходят через промежуточное программное обеспечение (middleware), которое находится на сервере приложений. Вследствие этого, повышается гибкость работы и производительность.

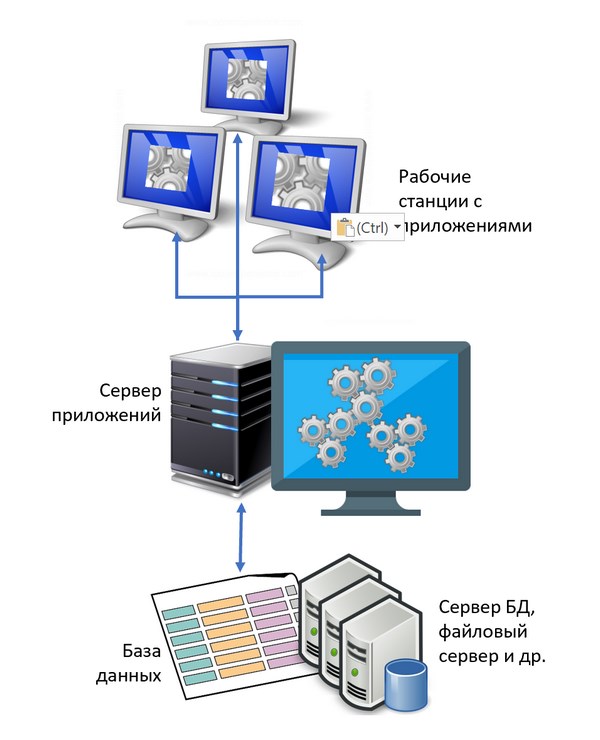


Рис. 4. Трёхуровневая архитектура «клиент-сервер» (3-Tier).

Преимущества трёхуровневой архитектуры:

• Целостность данных;

• Более высокая безопасность, по сравнению с двухуровневой архитектурой;

• Защищённость базы данных от несанкционированного проникновения.

Ограничения:

• Более сложная структура коммуникаций между клиентов и сервером, поскольку в нём также находится middleware.

Многоуровневая архитектура (N-Tier)

В отдельный класс архитектуры «клиент-сервер» можно вынести многоуровневую архитектуру, в которой несколько серверов приложений используют результаты работы друг друга, а также данные от различных серверов баз данных, файловых серверов и других видов серверов.

По сути, предыдущий вариант, трёхуровневая архитектура – не более, чем частный случай многоуровневой архитектуры.

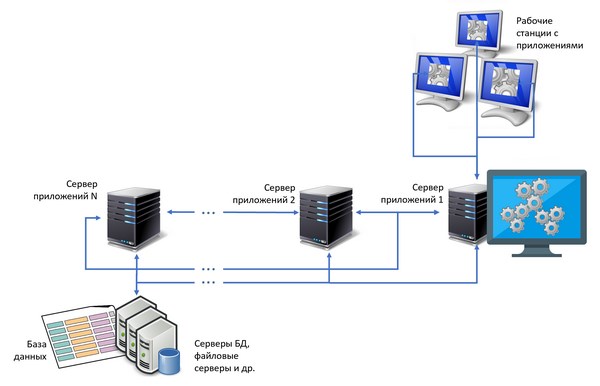


Рис. 5. Многоуровневая архитектура «клиент-сервер» (N-Tier).

Преимуществом многоуровневой архитектуры является гибкость предоставления услуг, которые могут являться комбинацией работы различных приложений серверов разных уровней и элементов этих приложений.

Очевидным недостатком является сложность, многокомпонентность такой архитектуры.

Характеристики архитектуры «клиент-сервер»

• Асимметричность протоколов. Между клиентами и сервером существуют отношения «один ко многим». Инициатором диалога с сервером обычно является клиент.

• Инкапсуляция услуг. После получения запроса на услугу от клиента, сервер решает, как должна быть выполнена данная услуга. Модификация («апгрейд») сервера может производиться без влияния на работу клиентов, поскольку это не влияет на опубликованный интерфейс взаимодействия между ними. Иными словами, максимум, что может при этом почувствовать пользователь – незначительная задержка отклика сервера в течение небольшого времени апгрейда.

• Целостность. Программы и общие данные для сервера управляются централизованно, что снижает стоимость обслуживания и защищает целостность данных. В то же время, данные клиентов остаются персонифицированными и независимыми.

• Местная прозрачность. Сервер – это программный процесс, который может исполняться на той же машине, что и клиент, либо на другой машине, подключенной по сети. Программное обеспечение «клиент-сервер» обычно скрывает местоположение сервера от клиентов, перенаправляя запрос на услуги через сеть.

• Обмен на основе сообщений. Клиенты и сервер являются нежёстко связанными («loosely-coupled») процессами, которые обмениваются сообщениями: запросами на услуги и ответами на них.

• Модульный дизайн, способный к расширению. Модульный дизайн программной платформы «клиент-сервер» придаёт ей устойчивость к отказам, то есть, отказ в каком-то модуле не вызывает отказа всего приложения. В такой системе, один или больше серверов могут отказать без остановки всей системы в целом, до тех пор, пока услуги отказавшего сервера могут быть предоставлены с резервного сервера. Другое преимущество модульности в том, что приложение «клиент-сервер» может автоматически реагировать на повышение или понижение нагрузки на систему, путём добавления или отключения услуг или серверов.

• Независимость от платформы. Идеальное приложение «клиент-сервер» не зависит от платформ оборудования или операционной системы. Клиенты и серверы могут развёртываться на различных аппаратных платформах и разных операционных системах.

• Масштабируемость. Системы «клиент-сервер» могут масштабироваться как горизонтально (по числу серверов и клиентов), так и вертикально (по производительности и спектру услуг).

• Разделение функционала. Система «клиент-сервер» — это соотношение между процессами, работающими на одной или на разных машинах. Сервер – это процесс предоставления услуг. Клиент – это потребитель услуг.

• Общее использование ресурсов. Один сервер может предоставлять услуги множеству клиентов одновременно, и регулировать их доступ к совместно используемым ресурсам.

Практические применения архитектуры «клиент-сервер»

Архитектуры «клиент-сервер» - один из основных принципов работы сети Интернет. Любой веб-сайт, или приложение в Интернет работает на сервере, а его пользователи являются клиентами. Социальные сети (Фейсбук, ВК и пр.), сайты электронной коммерции (Amazon, Озон и др.) , мобильные приложения (Instagram и т.д.), устройства Интернета вещей (умные колонки или смарт-часы) работают на основе клиент-серверной архитектуры.

Хорошим примером работы системы «клиент-сервер» является автомобильный навигатор. Приложение навигации на сервере собирает данные с многих смартфонов пользователей, на которых установлены клиенты приложения. Кроме того, приложение навигации использует ещё и данные с сервера базы данных – геоинформационной системы, который предоставляет данные, например, о текущих ремонтах дорог, о появлении новых дорог и пр. Данные со многих клиентов (местоположение, скорость) обрабатывается сервером навигации и выдаётся на смартфоны пользователей в виде информации о средней скорости движения по тому или иному участку маршрута.

Практически любая корпоративная сеть или ИТ-система предприятия, как правило, строится по архитектуре «клиент-сервер». В небольших сетях (3-5 компьютеров в компании) функции сервера может выполнять один из рабочих компьютеров. Если число машин в организации более 10, то лучше сделать выделенный сервер (почтовый сервер, приложений, баз данных и пр.), который будет заниматься обслуживанием клиентов – компьютеров и телефонов сотрудников организации.

В домашних сетях архитектура «клиент-сервер» тоже используется довольно часто. Например, в домашнюю сеть могут быть объединены компьютеры членов семьи, один из которых выполняет функции сервера. В домашнюю сеть также могут быть включены такие устройства, как умные колонки, умные домашние устройства (пылесосы-роботы, фотоаппараты, DVD-плееры и пр.), а также «умные» счётчики (вода, электричество) и т.д. Тогда в системе управления сервера, будут видны все параметры, данные и медифайлы (музыка, видео, фото), а также «умные устройства».

Преимущества и недостатки архитектуры «клиент-сервер»

К преимуществам архитектуры «клиент-сервер» можно отнести:

• Централизованность, поскольку все данные и управление сосредоточены в центральном сервере;

• Информационная безопасность, поскольку ресурсы общего пользования администрируются централизованно;

• Производительность, использование выделенного сервера повышает скорость работы ресурсов общего пользования;

• Масштабируемость, количество клиентов и серверов можно увеличивать независимо друг от друга.

К недостаткам архитектуры «клиент-сервер» следует отнести:

• Перегрузку трафика в сети, что является главной проблемой в сетях «клиент-сервер». Когда большое число клиентов одновременно запрашивают одну услугу на сервере, то число запросов может создать перегрузку в сети;

• Наличие единой точки отказа в небольших сетях с одним сервером. Если он отказывает, все клиенты остаются без обслуживания;

• Превышение пределов ресурсов сервера, когда новые клиенты, запрашивающие услугу, остаются без обслуживания. В таких случаях, требуется срочное расширение ресурсов сервера;

• Иногда клиентские программы могут не работать на терминалах пользователей, если не установлены соответствующие драйверы. Например, пользователь посылает запрос на печать документа, а на сервере нет подходящего драйвера для печати данного формата документа на определённом принтере.

**7)Жизненный цикл программного обеспечения**

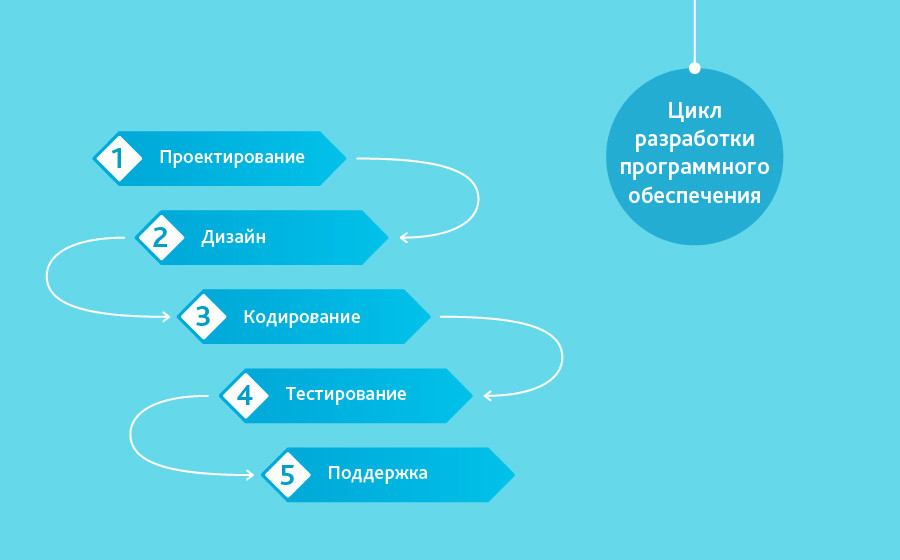
Это период времени, который начинается с момента принятия решения о создании программного продукта (средства) и заканчивается в момент его полного изъятия из эксплуатации.

Процесс жизни любой системы или программного продукта может быть описан с помощью модели жизненного цикла, состоящей из стадий: ***Идея → Анализ требований → Проектирование → Разработка → Тестирование → Стабилизация и релиз → Поддержка и сопровождение***

Определенная последовательность стадий, зависящая от ряда условий, в том числе от финансовой составляющей, времени, наличия компетенций в команде и так далее характеризует разные модели жизненного цикла.

Выделяют следующие модели разработки:

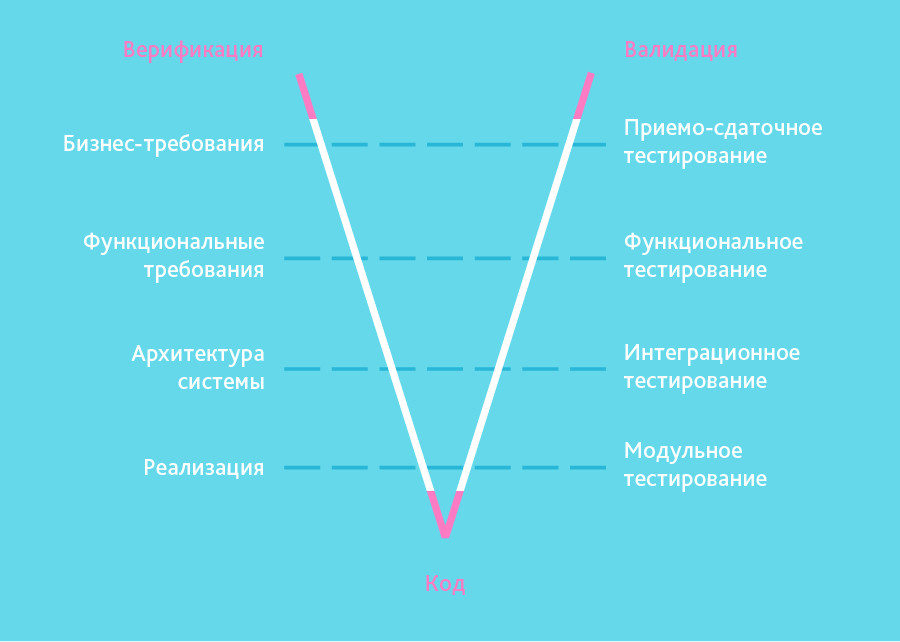
***Каскадная***

Одна из самых старых, подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей. В модели Waterfall легко управлять проектом. Благодаря её жесткости, разработка проходит быстро, стоимость и срок заранее определены. Но это палка о двух концах. Каскадная модель будет давать отличный результат только в проектах с [четко и заранее определенными требованиями](http://www.edsd.ru/ru/uslugi/poektirovanie) и способами их реализации. Нет возможности сделать шаг назад, тестирование начинается только после того, как разработка завершена или почти завершена. Продукты, разработанные по данной модели без обоснованного ее выбора, могут иметь недочеты (список требований нельзя скорректировать в любой момент), о которых становится известно лишь в конце из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта. Тем не менее, фиксированная стоимость часто перевешивает минусы подхода. Исправление осознанных в процессе создания недостатков возможно, и, по нашему опыту, требует от одного до трех дополнительных соглашений к контракту с небольшим ТЗ.  
  


С помощью каскадной модели мы создали множество проектов «с нуля», включая разработку только ТЗ. Проекты, о которых написано на Хабре: средний — [рентгеновский микротомограф](http://habrahabr.ru/company/edison/blog/273295), мелкий — [автообновление службы Windows на AWS](http://habrahabr.ru/company/edison/blog/270777).

**Когда использовать каскадную методологию?**

* Только тогда, когда требования известны, понятны и зафиксированы. Противоречивых требований не имеется.
* Нет проблем с доступностью программистов нужной квалификации.
* В относительно небольших проектах.

***В модель***  


Унаследовала структуру «шаг за шагом» от каскадной модели. V-образная модель применима к системам, которым особенно важно бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности в транспортных средствах и так далее. Особенностью модели можно считать то, что она направлена на тщательную проверку и [тестирование продукта](http://www.edsd.ru/ru/uslugi/testirovanie_po), находящегося уже на первоначальных стадиях проектирования. Стадия тестирования проводится одновременно с соответствующей стадией разработки, например, во время кодирования пишутся модульные тесты.

Пример нашей работы на основе V-методологии — мобильное приложение для европейского сотового оператора, который экономит расходы на роуминг во время путешествий. Проект выполняется по четкому ТЗ, но в него включен значительный этап тестирования: удобства интерфейса, функционального, нагрузочного и в том числе интеграционного, которое должно подтверждать, что несколько компонентов от различных производителей вместе работают стабильно, невозможна кража денег и кредитов.  
**Когда использовать V-модель?**

* Если требуется тщательное тестирование продукта, то V-модель оправдает заложенную в себя идею: validation and verification.
* Для малых и средних проектов, где требования четко определены и фиксированы.
* В условиях доступности инженеров необходимой квалификации, особенно тестировщиков.

***Спиральная***  


«Спиральная модель» похожа на инкрементную, но с акцентом на анализ рисков. Она хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек, при необходимости научных исследований и практической апробации.  
  
**Спиральная модель предполагает 4 этапа для каждого витка:**

1. планирование;
2. анализ рисков;
3. конструирование;
4. оценка результата и при удовлетворительном качестве переход к новому витку.

Эта модель не подойдет для малых проектов, она резонна для сложных и дорогих, например, таких, как разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование. На проекте по [разработке СЭД для ОДУ Сибири СО ЕЭС](http://edsd.biz/odu-sibiri) два совещания об изменении кодификации разделов электронного архива занимают в 10 раз больше времени, чем объединение двух папок программистом. Государственные проекты, в которых мы участвовали, начинались с подготовки экспертным сообществом дорогостоящей концепции, которая отнюдь не всегда бесполезна, поскольку окупается в масштабах страны.

***Инкрементальная***

В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система.  
  


Инкрементные модели используются там, где отдельные запросы на изменение ясны, могут быть легко формализованы и реализованы. В наших проектах мы применяли ее для создания читалки DefView, а следом и сети электронных библиотек Vivaldi.

Как пример опишем cуть одного инкремента. [Сеть электронных библиотек Vivaldi](http://habrahabr.ru/company/edison/blog/232033/) пришла на смену DefView. DefView подключалась к одному серверу документов, а теперь может подключаться ко многим. На площадку учреждения, желающего транслировать свой контент определенной аудитории, устанавливается сервер хранения, который напрямую обращается к документам и преобразует их в нужный формат. Появился корневой элемент архитектуры — центральный сервер Vivaldi, выступающий в роли единой поисковой системы по всем серверам хранения, установленным в различных учреждениях.  
**Когда использовать инкрементную модель?**

* Когда основные требования к системе четко определены и понятны. В то же время некоторые детали могут дорабатываться с течением времени.
* Требуется ранний вывод продукта на рынок.
* Есть несколько рисковых фич или целей.

***Быстрая модель RAD***

RAD Model» (rapid application development model или быстрая разработка приложений)

RAD-модель — разновидность инкрементной модели. В RAD-модели компоненты или функции разрабатываются несколькими высококвалифицированными командами параллельно, будто несколько мини-проектов. Временные рамки одного цикла жестко ограничены. Созданные модули затем интегрируются в один рабочий прототип. Синергия позволяет очень быстро предоставить клиенту для обозрения что-то рабочее с целью получения обратной связи и внесения изменений.

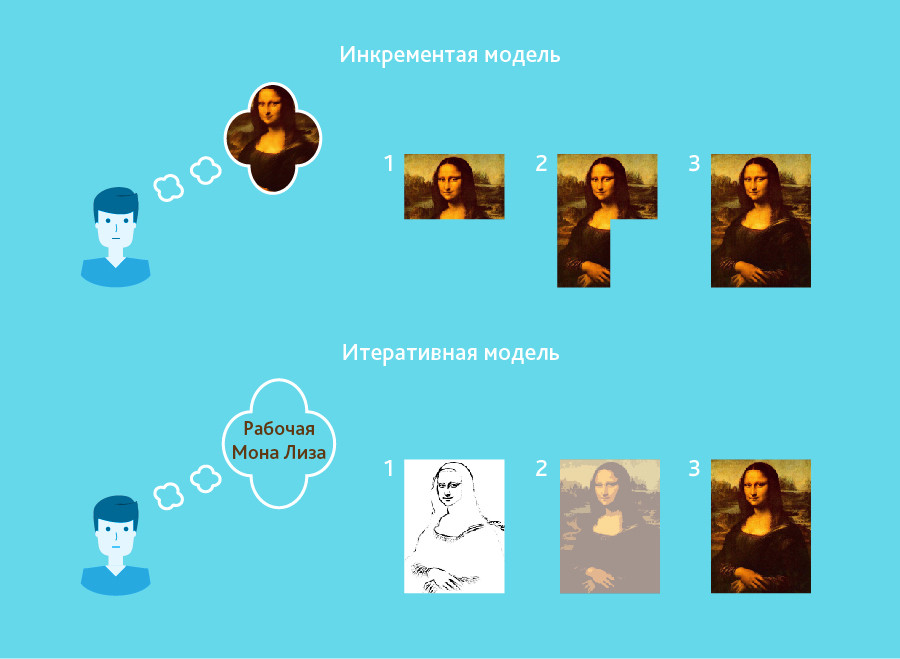
**Модель быстрой разработки приложений включает следующие фазы:**

* Бизнес-моделирование: определение списка информационных потоков между различными подразделениями.
* Моделирование данных: информация, собранная на предыдущем этапе, используется для определения объектов и иных сущностей, необходимых для циркуляции информации.
* Моделирование процесса: информационные потоки связывают объекты для достижения целей разработки.
* Сборка приложения: используются средства автоматической сборки для преобразования моделей системы автоматического проектирования в код.
* Тестирование: тестируются новые компоненты и интерфейсы.

**Когда используется RAD-модель?**

Может использоваться только при наличии высококвалифицированных и узкоспециализированных архитекторов. Бюджет проекта большой, чтобы оплатить этих специалистов вместе со стоимостью готовых инструментов автоматизированной сборки. RAD-модель может быть выбрана при уверенном знании целевого бизнеса и необходимости срочного производства системы в течение 2-3 месяцев.

***Итеративная***

Итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия — работоспособна.  
  


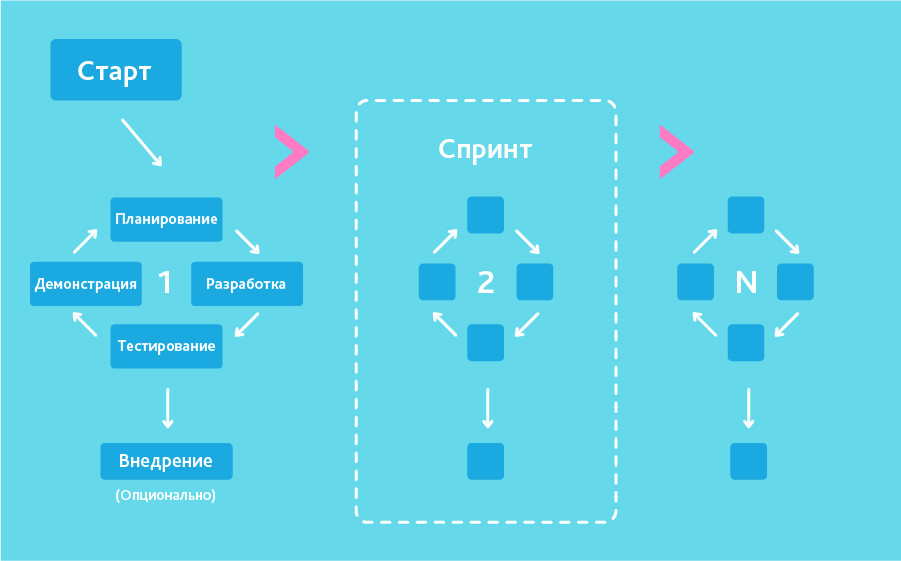
На диаграмме показана итерационная «разработка» Мона Лизы. Как видно, в первой итерации есть лишь набросок Джоконды, во второй — появляются цвета, а третья итерация добавляет деталей, насыщенности и завершает процесс. В инкрементной же модели функционал продукта наращивается по кусочкам, продукт составляется из частей. В отличие от итерационной модели, каждый кусочек представляет собой целостный элемент.

Примером итерационной разработки может служить распознавание голоса. Первые исследования и подготовка научного аппарата начались давно, в начале — в мыслях, затем — на бумаге. С каждой новой итерацией качество распознавания улучшалось. Тем не менее, идеальное распознавание еще не достигнуто, следовательно, задача еще не решена полностью.  
**Когда оптимально использовать итеративную модель?**

* Требования к конечной системе заранее четко определены и понятны.
* Проект большой или очень большой.
* Основная задача должна быть определена, но детали реализации могут эволюционировать с течением времени.

***Гибкая***

«Agile Model» (гибкая методология разработки)



В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике.

В основе такого типа — непродолжительные ежедневные встречи — «Scrum» и регулярно повторяющиеся собрания (раз в неделю, раз в две недели или раз в месяц), которые называются «Sprint». На ежедневных совещаниях участники команды обсуждают:

* отчёт о проделанной работе с момента последнего Scrum’a;
* список задач, которые сотрудник должен выполнить до следующего собрания;
* затруднения, возникшие в ходе работы.

Методология подходит для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка. Соответственно, в процессе реализации требования изменяются. Стоит вспомнить класс творческих людей, которым свойственно генерировать, выдавать и опробовать новые идеи еженедельно или даже ежедневно. Гибкая разработка лучше всего подходит для этого психотипа руководителей. Внутренние стартапы компании мы разрабатываем по Agile. Примером клиентских проектов является [Электронная Система Медицинских Осмотров](http://www.kvzrm.ru/), созданная для проведения массовых медосмотров в считанные минуты. Во втором абзаце [этого отзыва](http://www.edsd.ru/public/images/otzyv-ndvision.jpg), наши американские партнеры описали очень важную вещь, принципиальную для успеха на Agile.  
**Когда использовать Agile?**

* Когда потребности пользователей постоянно меняются в динамическом бизнесе.
* Изменения на Agile реализуются за меньшую цену из-за частых инкрементов.
* В отличие от модели водопада, в гибкой модели для старта проекта достаточно лишь небольшого планирования.

**Scrum и Kanban — представители методологий Agile-семейства.**

[**https://agilemanifesto.org/iso/ru/manifesto.html**](https://agilemanifesto.org/iso/ru/manifesto.html)

[**https://www.youtube.com/watch?v=cDvZaXzQezs**](https://www.youtube.com/watch?v=cDvZaXzQezs)

В чем отличие Scrum и Kanban?

Есть мнение, что Scrum — более жесткий метод проектирования, так как для него характерны строгие регламенты проведения мероприятий, наличие ролей, беклогов, работа короткими итерациями (спринтами) и тому подобное.

Основу Scrum составляют короткие итерации или спринты, как правило, 2-3-х недельные. Перед началом спринта команда сама формирует список фич на итерацию, далее запускается спринт.

После окончания спринта выполненные фичи заливаются на продакшн, а невыполненные — переносятся в другой спринт. Как правило, фичи, которые делаются во время спринта, не меняются: что было на старте спринта — должно быть сделано любой ценой к окончанию спринта.

Основная разница между Scrum и Канбан — в длине итераций. В Scrum итерации — 2 недели, в Kanban задачи программисту можно «подсовывать» хоть каждый день.

В Scrum задачи принято оценивать в Story points или в часах. Без оценки не получится сформировать спринт: ведь нам нужно знать, успеем ли мы сделать задачи за 2 недели. Через 2 недели мы получаем ценную статистику — сколько часов или Story points команда смогла сделать за спринт. Velocity — это производительность команды за один спринт. Этот параметр позволяет Scrum менеджеру предсказать, где команда будет через 2 недели.

В Kanban не принято делать оценку. Это опционально, команда решает сама. Здесь нет понятия «скорость работы команды», считается только среднее время на задачу. Время это считается с помощью специального отчета — Cycle Time.

Cycle Time для задачи = время выполнения задачи минус время начала работы над задачей. Например, у вас есть колонки: to do, reopened, developing, testing, stage testing, deployed. Cycle time для задачи будет равен deployed-developing, то есть сколько времени прошло от момента, когда задачу начали делать до момента пока она попала в deployed.

Итак, в Scrum наша цель — закончить спринт, в Kanban — задачу.

Kanban более мягкий. В нём всего два главных «предписания»:

1. Визуализация процессов
2. Сокращение работы в процессе (WIP, Work In Progress).

При этом остальное остаётся на ваше усмотрение. Например, вам необязательно выделять мероприятия и роли. Конечно, в большой команде это желательно, однако напрямую это не предписано, в отличие от Scrum.

Проще говоря, работайте, как считаете оптимальным, в том числе, сочетая Kanban со Scrum, только визуализируйте процессы и не берите в работу больше задач, чем нужно.

**Артефакты Скрама (Scrum Artifacts)**

Артефакты – это материальное представление работы или ценности. В Скраме существует три артефакта: Бэклог Продукта, Бэклог Спринта, Инкремент.

Они спроектированы таким образом, чтобы обеспечить максимальную прозрачность ключевой информации, и чтобы все участники процесса имели единое понимание каждого из артефактов.

1) Бэклог Продукта (Product Backlog)

Бэклог – это упорядоченный по приоритету список работ, которые планируется выполнить с учетом знаний, имеющихся на данный момент. Бэклог Продукта – это упорядоченный и постоянно обновляемый список всего, что планируется сделать для создания и улучшения продукта. Он является единственным источником работы для Скрам-команды. Владелец Продукта несет ответственность за Бэклог Продукта, включая его содержимое, доступность и упорядочение.

2) Бэклог Спринта (Sprint Backlog)

Бэклог – это упорядоченный по приоритету список работ, которые планируется выполнить с учетом знаний, имеющихся на данный момент.

Бэклог Спринта – это Цель Спринта, набор Элементов Бэклога Продукта, выбранных для выполнения в текущем Спринте, а также план разработки Инкремента продукта и достижения Цели Спринта. Служит для наглядного представления работы, которую Команда определила для достижения Цели Спринта.

3) Инкремент (Increment)

Инкремент объединяет реализацию Элементов Бэклога Продукта, сделанную во время текущего Спринта. Является одним из трех Артефактов Скрама и отражает шаг на пути к Цели Продукта. Каждый Спринт должен включать минимум один Инкремент, чтобы считаться завершенным успешно.

Элемент Бэклога Продукта (Product Backlog Item)

Элемент Бэклога представляет собой часть работы, которую планируется сделать с учетом знаний, имеющихся на данный момент.

Элемент Бэклога Продукта – изменение, которое планируется выполнить в следующих Инкрементах продукта (например, фичи, функции, требования, усовершенствования или информация по исправлению дефектов). Элементы, расположенные в верхней части Бэклога Продукта, обычно более понятны и содержат больше деталей, чем те, что расположены ниже. 