### Понятие проектирования, его результаты и роль в процессе жизненного цикла информационной системы.

### Понятие проекта, объекта проектирования и процесса проектирования.

Проект - совокупность документов (расчётов, чертежей) для создания какого-либо сооружения или изделия.

Объект проектирования - некая система, которая будет и которая должна заполнить функциональную нишу во внешней среде.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Этап 1 - ПОНЯТЬ - Определить задачу

Этап 2 - ИССЛЕДОВАТЬ - Произвести исследования

Этап 3 – ОПРЕДЕЛИТЬ – Выделить спецификации решения

Этап 4 – ПРЕДСТАВИТЬ – Выработать решения

Этап 5 – ПРОТОТИП – Понять, работают ли концепции

Этап 6 – ВЫБРАТЬ – Выделить окончательную концепцию

Этап 7 – ДОРАБОТАТЬ – Детально проработать проект

Этап 8 – ПОКАЗАТЬ – Получить реакцию и утверждение

Этап 9 – ПРИМЕНИТЬ – Применить решение

Этап 10 – ИСПЫТАТЬ – Испытать решение на соответствие рабочих показателей

Этап 11 – ПОВТОРИТЬ

### Понятие информационной системы и необходимость проектирования ИС

Система - комплекс процессов, программных средств, которые удовлетворяют заранее заданному условиям (потребностям, целям) - по ГОСТу

Информационная система (ИС) - система, занимающаяся обработкой информации, но для нее необходимы ресурсы, не только IT’шные, но и технические (сервера, код, etc.), человеческие, финансовые.

Процесс проектирования - основной рабочий процесс разработки ПО, целью которого является создание модели, содержащей проектные решения, удовлетворяющие функциональным и нефункциональным требованиям, а также ограничениям, относящимся к среде реализации.

Процесс проектирования предназначен для подготовки к реализации и тестирования системы.

Необходимость проектирования ИС заключается в:

* Формализации автоматизируемых бизнес-процессов: нашли процесс, который решили автоматизировать с помощью ИС.
* Унификация решений для автоматизации: когда проектируем ИС, можем переиспользовать то, что уже есть. Не надо каждый раз изобретать велосипед.
* Ликвидация человеческого фактора при реализации: разработчик, тестировщик видят по разному, иначе каждый будет делать систему, то как он представляет себе он, и не имея описания того как компоненты должны с собой взаимодействовать получим набор неподдерживаемого софта.
* Выработка подходящих архитектурных решений: в рамках создания ИС есть сущность архитектор программных решений, который позволяет выбрать правильные архитектурные решения, паттерны реализации.
* Подбор технологии реализации: исходя из целей и задач можно понять, на чём будет реализовано. Небольшая ИС - нет смысла делать на enterprise-технологиях.
* Оптимизация бюджета создания ИС: за счёт подбора технологий, за счёт снижения рисков на человеческие перезатраты.
* Определение сроков и порядка реализации ИС: если нет проекта, неизвестно сколько это займет. Если есть четкое описание того, что надо сделать - сможем определить очередность реализации ИС и сроки. Современные подходы: нет такого, что система разом внедряется, некоторые модули.

### Жизненный цикл информационной системы

Жизненный цикл - один процесс, который состоит из разных блоков, которые идут строго в одной последовательности. Каждый этап имеет на входе результаты предыдущего этапа. Также на входе используются сторонние артефакты.

1. Анализ (сбор требований)  
   Цель: сбор требований к ИС, формирование исходных формализованных данных для проектирования ИС  
   Входные данные: исходные неструктурированные документы (какие-то части регламентов, инструкций, служебные записки), интервью заказчика (вытягиваем данные с заказчика).  
   Выходные данные: бизнес-требования (собрали то, что было разрозненной картинкой, получили цельное описание)
2. Проектирование  
   Цель: формирование требований к системе и ее модулям, создание архитектуры системы  
   Входные данные: бизнес-требования, экспертиза в области разработки ИС  
   Выходные данные: техническое задание на разработку ИС (документ, понятный разработчику: ГОСТовый документ, набор таск в трекинговой системе)
3. Реализация (наиболее длительный этап: чем хуже проектирование, тем меньшее качество реализации. Или проектирование хорошее, но реализация не устроила Заказчика)  
   Цель: реализация ИС в соответствии с требованиями Заказчика  
   Входные данные: Техническое задание, технологии и платформы для разработки  
   Выходные данные: Реализованная ИС
4. Внедрение  
   Цель: перевод разработанной ИС в промышленную эксплуатацию  
   Входные данные: разработанная ИС, требования к установке (если установить невозможно, то система работает медленно, Заказчик недоволен), требования к наполнению данными(без данных система не имеет никакой ценности), требования к обучению (то, как должен быть обучен персонал для работы с системой -> снижение КПД взаимодействия)  
   Выходные данные: промышленно эксплуатируемая ИС
5. Сопровождение  
   Цель: поддержка и развитие внедренной ИС  
   Входные данные: внедренная ИС, требования к работоспособности, требования к модернизации (при изменении внешних вводных данных, перестройка бизнес-процессов, требуется внести доработки в приложение)  
   Выходные данные: промышленно эксплуатируемая актуальная ИС
6. (не указан) Вывод из эксплуатации

### Типы информационных систем

Первый тип: ручная информационная система (Example: excel): не автоматизируем при помощи функционала никаких действий, исключительно выполняем необходимые функции для этой системы.

Второй тип: автоматизированная информационная система (та, с которой мы работаем): (ERP, CRM, etc.) - автоматизируют набор функциональностей, но требуют участия человека. Не работают сами по себе, но выполняют наиболее трудоемкие операции.

Третий тип: автоматическая ИС: есть центр, который на основе поступающих данных принимает решения, на основе данных по счёту трафика система сама может регулировать “зеленую волну”

Автоматизированные ИС можно делить на:

1. Системы по сфере применения
   1. Интегрированные: ERM-системы, CRM-системы
   2. Организационного управления: (пример: управления персоналом - на основе имеющихся данных взаимодействовать с персоналом)
   3. Управления ТП (тех.процесс): реализовывать некоторый процесс (пример: конвейерный)
   4. САПР (автоматизированного проектирования)
2. Системы по характеру информации
   1. Информационно-поисковые: позволяют найти информацию (Справочники типо “Консультант”, “Гарант”, “Гугл”)
   2. Информационно-решающие: либо управляют на основе поступающие данных и решений пользователей (Управляющие: через диспетчер влиять на производство), либо экспертные системы (Советующие)

### Классификация информационных систем

Можно классифицировать:

1. По уровням квалификации (несколько уровней пользователей, начиная от менеджеров высшего звена, среднего звена до исполнителей)
2. По уровням управления
   1. оперативный (операционный) - решение ежедневных задач
   2. Функциональный - решение задач в пределах квартала, месяца
   3. стратегический - принятие решений, которые оказывают влияние на всю организацию в долгосрочной перспективе)
3. Профессиональные системы (производство, финансы, маркетинг, кадры)

### Формы представления результатов проектирования ИС

Результатом этапа проектирования можно рассматривать набор сущностей:

* Структура системы - то, как она будет работать - те модули, которые в ней существуют
* Данные системы - определяем набор данных,которые будут в системе. Система без данных не имеет смысла. Может быть указано в документе из полей, типа полей и ограничений
* Интерфейсы взаимодействия - не одна система не варится сама по себе. Либо визуальный интерфейс взаимодействия с пользователем, либо программный интерфейс взаимодействия с другими системами. Сделали систему, выводит пользователю данные, передает через программный интерфейс другим системам. Это закладывается на этапе проектирования.
* Используемые алгоритмы - заранее закладываем те алгоритмы, которые мы используем. (алгоритмы вычисления подходящего специалиста, оператора call-центра)
* Модели бизнес-процессов: его автоматизируем, закладываем его модели, на этапе реализации описываем как будет работать)  
    
  Как могут выглядеть:

1. Схема - наиболее удобный результат, который позволяет вникнуть человеку в результат. Плоха тем, что не передает детали, в отдельном документе нужно описывать как она работает.
2. Документы - ТЗ, частное ТЗ
3. Прототипы - визуальное отображение того, что существует, или интерактивный прототип (показали заказчику, он одобрил, реализуем систему). Плюсы: быстро получить результат, показать его, понять в правильном ли направлении двигаемся. Может быть интерфейсным, или частью бизнес-логики.

### Понятие и характеристика объекта автоматизации

Объект автоматизации - процесс. Любая ИС нацелена на то, чтобы в процессе использования сотрудники/пользователи могли работать с процессом.

Процесс принадлежит организации

Характеристика объекта автоматизации:

* Краткая вводная информация о компании
* Краткая вводная информация о пользователях системы и их организационной структуре
* Краткая информация об автоматизируемом бизнес-процессе или тех.процессе или составе оборудования для которого строится АСУ.
* Краткая информация об условиях эксплуатации (инфа о кол-во пользователей, о временных интервалах доступности системы)

### Критерии оценки ИС

Информационная система должна обеспечивать:

1. Требуемую функциональность системы и степень адаптации к изменяющимся условиям ее функционирования - условия функционирования могут меняться (Модернизация системы). При отсутствии модернизации системы качество системы падает.
2. Требуемую пропускную способность системы
3. Требуемое время реакции системы на запрос - не равно времени обработки запроса, реализуем контейнер.
4. Безотказную работу системы в требуемом режиме, иными словами - готовность и доступность системы для обработки запросов пользователей - если система работает, но есть ошибки с точки зрения надежности, то она не сможет вовремя обработать запросы. Должна работать безошибочно.
5. Простоту эксплуатации и поддержки системы - можно внедрить систему, которая будет для пользователей слишком дорога в поддержке. Когда пользователь вынужден обращаться к вендору или интегратору, он откажется от такой системы или будут пересмотрены подходы к ней.
6. Необходимую безопасность - определяется требованиями законодательства и здравого смысла. Например: работа с персональными данными в соответствии с 152 ФЗ.

### Понятие управления в ИС, объекты управления

Система управления - управляемый объект, процесс, действия или управляющие сигналы команд, которые к нему доставляются.

Любое действие системы управления направлено на улучшение объекта, процесса, либо на поддержание. Постоянно поступающий сигнал изменяет состояние объекта либо по временному триггеру, либо по смене состояния

Два больших объекта:

управляемый объект - элемент системы, который для нормального функционирования нуждается в систематическом контроле и регулировании (пример: документ, шаблон договора, отдельно запущенный экземпляр)

управляющий объект - тот самый элемент, который массой управляемых объектов управляет, т.е. осуществляет с ними взаимодействие для нормального функционирования (пример: система, которая следит за шаблоном договора в соответствии с входными параметрами, диспетчер процессов)

### Понятие модели и типы моделей, применяемых при проектировании ИС

Моделирование системы состоит из пересечения информации, функций и событий. При наступлении определенного события вызывается функция, которая взаимодействует с информацией.

Моделирование системы предназначено для составления полного представления о структуре итоговой системы и ее предназначения

Модель - это абстрактное представление системы, в котором игнорируются некоторые детали системы. Могут быть разработаны дополнительные модели системы, в которых представлена различная информация относительно системы.

Что можем моделировать:

* Системное окружение (понятие объекта автоматизации, определение границ системы, определение взаимосвязей системы с внешними объектами
* Поведенческие модели (модели потоков данных, модели конечных автоматов)
* моделирование объектов системы (определяем состав объектов, связи между объектами, состояния объектов, реакция объектов на события, наследование объектов)

### Типы архитектуры информационных систем, их применимость

Современные программные приложения и информационные системы достигли такого уровня развития, что термин "*архитектура*" в применении к ним уже давно не удивляет. Грамотно построить информационную систему, эффективно и надежно функционирующую не проще, чем сконструировать и возвести современное многофункциональное здание

Когда речь заходит об "архитектуре информационной системы", обычно не возникает недостатка в определениях. Есть даже Web-сайты, которые собирают такие определения

Рассмотрим определение "архитектуры информационной системы", которое дают различные источники:

· *Архитектура* – это организационная структура системы

· *Архитектура* **информационной системы** – концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы

· *Архитектура* – это базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением, а также принципы, определяющие проектирование и развитие системы [[5](https://intuit.ru/studies/courses/611/467/literature#literature.4.5)].

· *Архитектура* **–** это набор значимых решений по поводу организации системы программного обеспечения, набор структурных элементов и их интерфейсов, при помощи которых компонуется система, вместе с их поведением, определяемым во взаимодействии между этими элементами, компоновка элементов в постепенно укрупняющиеся подсистемы, а также стиль *архитектуры*, который направляет эту организацию – элементы и их интерфейсы, взаимодействия и компоновку [[6](https://intuit.ru/studies/courses/611/467/literature#literature.4.6)].

· *Архитектура* **программы или компьютерной системы** – это структура или структуры системы, которые включают элементы программы, видимые извне свойства этих элементов и связи между ними [[7](https://intuit.ru/studies/courses/611/467/literature#literature.4.7)].

· *Архитектура* – это структура организации и связанное с ней поведение системы [[8](https://intuit.ru/studies/courses/611/467/literature#literature.4.8)]. *Архитектуру* можно рекурсивно разобрать на части, взаимодействующие посредством интерфейсов, связи, которые соединяют части, и условия сборки частей. Части, которые взаимодействуют через интерфейсы, включают классы, компоненты и подсистемы.

· *Архитектура* **программного обеспечения** системы или набора систем состоит из всех важных проектных решений по поводу структур программы и взаимодействий между этими структурами, которые составляют системы [[9](https://intuit.ru/studies/courses/611/467/literature#literature.4.9)]. Проектные решения обеспечивают желаемый набор свойств, которые должна поддерживать система, чтобы быть успешной. Проектные решения предоставляют концептуальную основу для разработки системы, ее поддержки и обслуживания.

Хотя определения несколько отличаются, можно заметить немалую степень сходства. Например, большинство определений указывают на то, что *архитектура* связана со структурой и поведением, а также только со значимыми решениями, может соответствовать некоторому архитектурному стилю, на нее влияют заинтересованные в ней лица и ее окружение, она воплощает решения на основе логического обоснования.

Под архитектурой программных систем будем понимать совокупность решений относительно

· организации программной системы;

· выбора структурных элементов, составляющих систему и их интерфейсов;

· поведения этих элементов во взаимодействии с другими элементами;

· объединение этих элементов в подсистемы;

· *архитектурного стиля*, определяющего логическую и физическую организацию системы: статические и динамические элементы, их интерфейсы и способы их объединения.

*Архитектура* программной системы охватывает не только ее структурные и поведенческие аспекты, но и правила ее использования и интеграции с другими системами, функциональность, производительность, гибкость, надежность, возможность повторного применения, полноту, экономические и *технологические ограничения*, а также вопрос пользовательского интерфейса.

По мере развития программных систем все большее значение приобретает их интеграция друг с другом с целью построения единого информационного пространства предприятия. Как можно видеть из вышеприведенных определений интеграция является важнейшим элементом *архитектуры*.

Для того чтобы построить правильную и надежную *архитектуру* и грамотно спроектировать интеграцию программных систем необходимо четко следовать современным стандартам в этих областях. Без этого велика вероятность создать *архитектуру*, которая неспособна развиваться и удовлетворять растущим потребностям пользователей ИТ. В качестве законодателей стандартов в этой области выступают такие международные организации как SEI (Software Engineering Institute), WWW (консорциум World Wide Web), OMG (Object Management Group), организация разработчиков Java – *JCP* (Java Community Process), IEEE (Institute of Electrical and Electronics Engineers) и другие.

Рассмотрим классификацию программных систем по их *архитектуре*:

· *Централизованная архитектура*;

· *Архитектура "файл-сервер"*;

· *Двухзвенная архитектура "клиент-сервер"*;

· *Многозвенная архитектура "клиент-сервер"*;

· *Архитектура распределенных систем*;

· *Архитектура Веб-приложений*;

· *Сервис-ориентированная архитектура*.

Следует заметить, что, как и любая классификация, данная классификация архитектур информационных систем не является абсолютно жесткой. В *архитектуре* любой конкретной информационной системы часто можно найти влияния нескольких общих архитектурных решений.

Далее подробно рассмотрим особенности каждой *архитектуры*.

**5.1.2. Централизованная архитектура**

*Централизованная архитектура* вычислительных систем была распространена в 70-х и 80-х годах и реализовывалась на базе *мейнфреймов* (например, IBM-360/370 или их отечественных аналогов серии ЕС ЭВМ), либо на базе мини-ЭВМ (например, PDP-11 или их отечественного аналога СМ-4) [[11](https://intuit.ru/studies/courses/611/467/literature#literature.4.11)]. Характерная особенность такой *архитектуры* – полная "неинтеллектуальность" *терминалов*. Их работой управляет хост-ЭВМ.

Достоинства такой *архитектуры*

· пользователи совместно используют дорогие ресурсы ЭВМ и дорогие периферийные устройства;

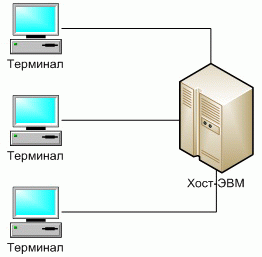
· централизация ресурсов и оборудования облегчает обслуживание и эксплуатацию вычислительной системы;

· отсутствует необходимость администрирования рабочих мест пользователей;

Главным недостатком для пользователя является то, что он полностью зависит от администратора хост-ЭВМ. Пользователь не может настроить рабочую среду под свои потребности – все используемое программное обеспечение является коллективным.

Использование такой *архитектуры* является оправданным, если хост-ЭВМ очень дорогая, например, *супер-ЭВМ*.

Классическое представление *централизованной архитектуры* показано на рисунке



**Рис. 5.1.** Классическое представление централизованной архитектуры

Центральная ЭВМ должна иметь большую память и высокую производительность, чтобы обеспечивать комфортную работу большого числа пользователей.

Все приложения, работающие в такой *архитектуре*, полностью находятся в основной памяти хост-ЭВМ. *Терминалы* являются лишь устройствами ввода-вывода и таким образом в минимальной степени поддерживают интерфейс пользователя.

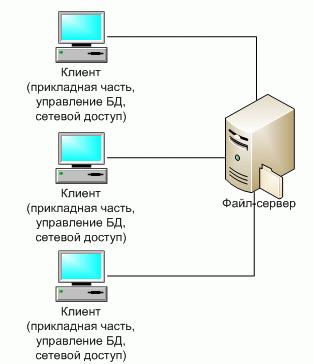
**5.1.3. Архитектура "файл-сервер"**

**Файл-серверные приложения** – приложения, схожие по своей структуре с локальными приложениями и использующие сетевой ресурс для хранения программы и данных

· Функции сервера: хранения данных и кода программы.

· Функции клиента: обработка данных происходит исключительно на стороне клиента.

Классическое представление информационной системы в *архитектуре* "файл-сервер" представлено рисунке



**Рис. 5.2.** Классическое представление архитектуры "файл-сервер"

Организация информационных систем на основе использования выделенных файл-серверов все еще является распространенной в связи с наличием большого количества персональных компьютеров разного уровня развитости и сравнительной дешевизны связывания PC в локальные сети

Конечно, основным достоинством данной *архитектуры* является простота организации. Проектировщики и разработчики информационной системы находятся в привычных и комфортных условиях IBM PC в среде MS-DOS, Windows или какого-либо облегченного варианта Windows Server. Имеются удобные и развитые средства разработки графического пользовательского интерфейса, простые в использовании средства разработки систем баз данных и/или СУБД.

Достоинства такой *архитектуры*

· *многопользовательский режим* работы с данными;

· удобство централизованного управления доступом;

· низкая стоимость разработки;

· высокая скорость разработки;

· невысокая стоимость обновления и изменения ПО.

Недостатки

· проблемы многопользовательской работы с данными: последовательный доступ, отсутствие гарантии целостности;

· низкая производительность (зависит от производительности сети, сервера, клиента);

· плохая возможность подключения новых клиентов;

· ненадежность системы.

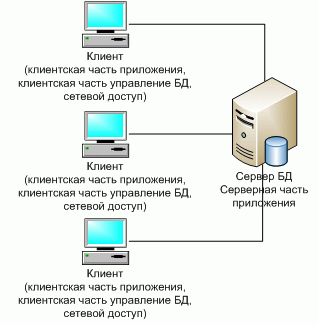
Простое, работающее с небольшими объемами информации и рассчитанное на применение в однопользовательском режиме, файл-серверное приложение можно спроектировать, разработать и отладить очень быстро [[14](https://intuit.ru/studies/courses/611/467/literature#literature.4.14)]. Очень часто для небольшой компании для ведения, например, кадрового учета достаточно иметь изолированную систему, работающую на отдельно стоящем PC. Однако, в уже ненамного более сложных случаях (например, при организации информационной системы поддержки проекта, выполняемого группой) файл-серверные архитектуры становятся недостаточными.

**5.1.4. Архитектура "клиент-сервер"**

**Клиент-сервер** (*Client-server*) – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемых серверами, и заказчиками услуг, называемых клиентами . Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

Первоначально системы такого уровня базировались на классической двухуровневой клиент-серверной архитектуре (Two-*tier* architecture). Под клиент-серверным приложением в этом случае понимается информационная система, основанная на использовании *серверов баз данных*.

Схематически такую *архитектуру* можно представить, как показано на рисунке



**Рис. 5.3.** Классическое представление архитектуры "клиент-сервер"

На стороне клиента выполняется код приложения, в который обязательно входят компоненты, поддерживающие интерфейс с конечным пользователем, производящие отчеты, выполняющие другие специфичные для приложения функции.

Клиентская часть приложения взаимодействует с клиентской частью программного обеспечения управления базами данных, которая, фактически, является индивидуальным представителем СУБД для приложения.

Заметим, что интерфейс между клиентской частью приложения и клиентской частью *сервера баз данных*, как правило, основан на использовании языка SQL. Поэтому такие функции, как, например, предварительная обработка форм, предназначенных для запросов к базе данных, или формирование результирующих отчетов выполняются в коде приложения.

Наконец, клиентская часть *сервера баз данных*, используя средства сетевого доступа, обращается к *серверу баз данных*, передавая ему текст оператора языка SQL.

Посмотрим теперь, что же происходит на стороне *сервера баз данных*. В продуктах практически всех компаний сервер получает от клиента текст оператора на языке SQL.

· Сервер производит компиляцию полученного оператора.

· Далее (если компиляция завершилась успешно) происходит выполнение оператора.

Разработчики и пользователи информационных систем, основанных на архитектуре "клиент-сервер", часто бывают неудовлетворены постоянно существующими сетевыми накладными расходами, которые следуют из потребности обращаться от клиента к серверу с каждым очередным запросом. На практике распространена ситуация, когда для эффективной работы отдельной клиентской составляющей информационной системы в действительности требуется только небольшая часть общей базы данных. Это приводит к идее поддержки *локального кэша* общей базы данных на стороне каждого клиента.

Фактически, концепция локального кэширования базы данных является частным случаем концепции реплицированных баз данных. Как и в общем случае, для поддержки *локального кэша* базы данных программное обеспечение рабочих станций должно содержать компонент управления базами данных – упрощенный вариант *сервера баз данных*, который, например, может не обеспечивать *многопользовательский режим* доступа. Отдельной проблемой является обеспечение согласованности (когерентности) кэшей и общей базы данных. Здесь возможны различные решения – от автоматической поддержки согласованности за счет средств базового программного обеспечения управления базами данных до полного перекладывания этой задачи на прикладной уровень.

Преимуществами данной *архитектуры* являются

· возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети;

· все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов, а также на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;

· поддержка многопользовательской работы;

· гарантия целостности данных.

Недостатки

· неработоспособность сервера может сделать неработоспособной всю вычислительную сеть;

· администрирование данной системы требует квалифицированного профессионала;

· высокая стоимость оборудования;

· бизнес логика приложений осталась в клиентском ПО.

При проектировании информационной системы, основанной на *архитектуре* "клиент-сервер", большее внимание следует обращать на грамотность общих решений. Технические средства пилотной версии могут быть минимальными (например, в качестве аппаратной основы *сервера баз данных* может использоваться одна из рабочих станций). После создания пилотной версии нужно провести дополнительную исследовательскую работу, чтобы выяснить узкие места системы. Только после этого необходимо принимать решение о выборе аппаратуры сервера, которая будет использоваться на практике.

Увеличение масштабов информационной системы не порождает принципиальных проблем. Обычным решением является замена аппаратуры сервера (и, может быть, аппаратуры рабочих станций, если требуется переход к локальному кэшированию баз данных). В любом случае практически не затрагивается прикладная часть информационной системы.

Также данный вид *архитектуры* называют *архитектурой* с "толстым" клиентом.

### Правила выбора программных платформ для реализации ИС

### Понятие CASE-технологии и инструментальных CASE-средств

CASE - набор инструментов и методов программной инженерии для проектирования программного обеспечения, который помогает обеспечить высокое качество программ, отсутствие ошибок и простоту в обслуживании программных продуктов. Также под CASE понимают совокупность методов и средств проектирования информационных систем с использованием CASE-инструментов.

Инструментальные CASE-средства - инструменты автоматизации процессов проектирования и разработки программного обеспечения для системного аналитика, разработчика ПО и программиста. Первоначально под CASE-средствами понимались только инструменты для упрощения наиболее трудоёмких процессов анализа и проектирования, но с приходом стандарта ISО/IEC 14102 CASE-средства стали определять как программные средства для поддержки процессов жизненного цикла ПО.

### Методологии моделирования

Всего есть 3 методологии:

* SADT (Structured Analysis and Design Technique) - структурный анализ
* DFD (Data Flow Diagrams) - диаграмма потоков данных: как данные меняются при переходе
* ERD (Entity-Relationship Diagrams) - отображение структуры базы данных

Основной инструмент моделирования - UML

### Типы CASE-систем

Можно выделить четко следующие системы:

* средства анализа - предназначены для построения и анализа модели предметной области
* средства проектирования баз данных - позволяют описать структуру баз данных
* средства разработки приложений - интегрированные среды, которые можем выделить (IDE)
* средства реинжиниринга процессов - позволяют в какой-либо из нотаций описать бизнес-процесс
* средства планирования и управления проектом - Microsoft Project, Primaware - позволяют управлять проектом в разрезе ресурсов, дат, затрат - понимаем, сколько все стоит по шагам
* средства тестирования - TestRail (средство управления тестами)
* средства документирования - любая ИС как-то документируется и в дальнейшем, в зависимости от доки, работаем с системой

### Понятие аппаратного обеспечения и его роль при проектировании ИС

### Элементы аппаратного обеспечения

* Процессорная инфраструктура (на сервере: x86, линейки процессоров Intel Xeon или AMD Epyc, когда-то это также могли бы быть Sun SPARC-процессоры; на клиенте: если это веб-решение, то любой процессор способный запустить современный браузер, если это толстый клиент, то необходимо определить какое количество процессорных ресурсов ему необходимо; в случае мобильных-решений, то тут все зависит от платформы и необходимо рассматривать ARM-процессоры для конкретной платформы)
* Оперативная память (в первую очередь необходимо учитывать используемый стек технологий и его аппетиты по памяти. В любом случае нужно брать с запасом, т.к. стоимость оперативной памяти не очень высока, а современные приложения довольно прожорливы, чтобы в будущем не докупать железки)
* Инфраструктура накопителей (HDD - в случае, если эта информация не требуется постоянно, а только по редким запросам/SSD - в случае, если информация постоянно запрашивается, также возможны решения на дисковых HDD массивах с кэшем на SSD накопителях. Intel Optane еще более быстрые SSD, например, для потокового видео. Также возможно создание RAID-массивов из дисков, в зависимости от требований к избыточности <https://ru.wikipedia.org/wiki/RAID>)
* Инфраструктура резервного копирования (приоритетна для enterprise решений, но менее приоритетна для небольших внутриорганизационных ИС. В enterprise это ленточные библиотеки (новые ленты от IBM и Fujitsu способны хранить до 480Гб данных на одной ленте без сжатия), в малых предприятиях это дисковые массивы или NAS-сервера)
* Инфраструктура электропитания (это могут быть небольшие ибп для поддержания работоспособности сервера в течение получаса-часа, либо дорогостоящие системы из банков, для того, чтобы было больше времени для запуска дизель-генератора. Также на материнских платах могут быть установлены специальные ионисторы (конденсаторы большой емкости) для питания основных компонентов, чтобы переместить данные, находящиеся в обработке на дисковое хранилище)

Приоритет каждого из элементов зависит от многих факторов, таких как: архитектура программных компонентов ИС, нагрузка, ЦА ИС, тип нагрузки, географическая распределенность вычислительных узлов, объемы перегоняемых данных. Все это должно быть рассмотрено в рамках проектирования.

### Элементы сетевой инфраструктуры

* Пропускная способность и типы ЛВС (зачастую самая важная характеристика для enterprise-решений) : нужно рассчитывать пропускную способность не только с учетом прямых пользователей, но и для сервисного подключения, например, для параллельного резервного копирования)
* Типы аппаратного обеспечения (Ethernet, high-load enterprise - Fibre Channel, если есть необходимость в быстром времени доступа и минимальном времени простоя, то сеть может быть построена для поддержки BGP, протокола маршрутизации с прямым доступом к другим устройствам и получением автономного куска сети на выходе. В случае опять-таки high-load могут потребоваться аппаратные Firewall-решения (NGFW))
* Интерфейсы взаимодействия
* Требования к сертификации (все сетевое оборудование работает по различным стандартам и должны учитываться стандарты, чтобы не возникло ситуации несовместимого оборудования)

При проектировании сетевой инфраструктуры очень важно узнать требования клиента к скорости ответа ИС, требуемый географический охват (т.е. требуемое время ответа для клиентов географически разноудаленных).

### Критерии формирования требований к аппаратному обеспечению

### Требования к сертификации аппаратного обеспечения

* Сертификация РосТест (формально любое аппаратное обеспечение должно работать в соответствии с РТ на территории РФ. Например, станции 5G должны работать на частотах, разрешенных к использованию на территории РФ)
* Отраслевые сертификации (например, для сектора связанного с госбезопасностью необходимо прохождение сертификации на предмет закладок, бэкдоров и прочего)
* Сертификация производителя ПО (обычно представляет из себя сертификацию железа под использование определенного ПО. Например, HP договаривается с Microsoft о прохождении сертификации на совместимость Windows Server и серверов HP Helion. В первую очередь это маркетинговая поддержка и, во-вторых, гарантия, что софт будет работать на этом железе)

Сертификация, в основном, важна для различных госзакупок, где всегда присутствуют жесткие требования к сертификации поставляемых аппаратных комплексов. Также сертификация может потребоваться для аппаратной части, к которой предъявлены высокие требования к безопасности (аппаратные комплексы для военных частей). Нет жестких требований к сертификации по части open-source проектов, только в случае планируемой разработки проприетарного ПО на основе open-source решений.

### Понятие интерфейса, его типы и роль при проектировании ИС

**Интерфейс** - совокупность возможностей, способов и методов одновременного действия (в том числе посредством обмена информацией между ними) двух имеющих общее разграничение, то есть не связанных линейно, информационных систем, устройств или программ, а также характеристиками соединения, сигналов обмена и т.д.

Интерфейс определяется характеристиками, которые предоставляет. Это характеристики по типу соединения, по скорости, по сигналу, который передается.

**Типы интерфейсов:**

* Аппаратные интерфейсы - система шин, разъемов, согласующих устройств, алгоритмов и протоколов, обеспечивающих связь всех частей микропроцессорной системы между собой. От характеристик интерфейса зависит быстродействие и надежность системы
* Программные интерфейсы - функциональность,которую некий программный компонент предоставляет другим программным компонентам
* Пользовательский интерфейс

### Аппаратные и программные интерфейсы

**Аппаратный интерфейс** - система шин, разъемов, согласующих устройств, алгоритмов и протоколов, обеспечивающих связь всех частей микропроцессорной системы между собой. От характеристик интерфейса зависит быстродействие и надежность системы

**Программный интерфейс** - функциональность,которую некий программный компонент предоставляет другим программным компонентам

**Интеграция** - процесс объединения частей в целое

**Интеграция данных** - объединение данных, находящихся в различных источниках и предоставление данных пользователям в унифицированном виде.

**Интеграция приложений** - реализация взаимодействия различных приложений на основе унифицированных механизмов.

**Протокол взаимодействия** - стандарт, описывающий правила взаимодействия функциональных блоков при передаче данных. Протоколы взаимодействия систем, как правило, располагаются на седьмом уровне модели OSI

### Технологии интеграции при применении программных интерфейсов

**Ключевые элементы интеграции:**

* Набор взаимодействующих ИС
* Состав передаваемых данных
* Способ взаимодействия
* Набор событий
* Протокол взаимодействия

**Общие проблемы интеграции:**

* Ограниченность типов данных (типы данных должны поддерживаться обеими системами, иначе придется приводить одни типы к другим)
* Сложность соблюдения стандартов (если стандарты плохо описаны/недостаточно квалификации персоналу, интегрирующему одну из частей в другую, то возникают трудности интеграции. Когда приходится много коммуницировать при реализации, то это повод упростить стандарты, сделать взаимодействие максимально простым)
* Каждая из систем должна реализовывать интерфейсы для интеграции (когда система реализуются впервые и еще нет четкого круга пользователей, использующих систему, то она может быть реализована вообще без публичных интерфейсов. Тогда одна из систем складывает данные в общую бд, а другая забирает. В случае, если обе системы реализуют программные интерфейсы, то проблем дальше нет никаких)
* Необходимость поддержки обратной совместимости (плохие варианты: самый плохой это поменять наружу апи v1.0 на апи v2.0, в итоге сигнатуры методов меняются, меняется структура данных, всем кто использовал апи приходится подстраиваться под изменения, чтобы восстановить работоспособность. Либо в другом варианте мы сохраняем все сигнатуры и типы данных, но при этом внутри меняем логику работы с данными и возвращаем наружу совсем не то, что от нас ожидают. Для обратной совместимости используется, во-первых, версионирование апи, во-вторых, поддерживать легаси и, в-третьих, api gateway - представление шлюза, который роутит апи клиента в зависимости от информации о самом клиенте, куда он ходит и т.д.)
* Снижение скорости работы (если система работает в рамках одного физического сервера, то виртуальная машина, как правило, будет до определенного масштаба работать быстрее: части системы разнесены по разным нодам, которые интегрируются между собой, но опять же до определенной нагрузки. Любая интеграция снижает скорость раоты за счет сетевых издержек. С другой стороны можно масштабировать систему на другие виртуальные машины, когда появятся проблемы с производительностью на одной машине)

**Типы интеграции:**

* **Локальная:**
  + **Плагин** - независимо компилируемый программный модуль, динамически подключаемый к основной программе и преднозначенный для расширения и/или использования ее возможностей. Плагины обычно выполняются с помощью библиотек общего пользования
  + Реализуется на базе типовых интерфейсов приложения
  + Создаются на языке, поддерживаемым приложением
  + Не работает вне приложения
* **Распределенная**

**Интеграция корпоративных ИС (КИС):**

* Распределенное взаимодействие, в т.ч. сетевое
* Двусторонняя передача данных
* Большой объем и сложные структуры данных

**Способы интеграции:**

* **API (application programming interface)**  - интерфейс программирования приложения
* **COM (component object model)** - пример API. Технологический стандарт от Microsoft, предназначенный для создания программного обеспечения на основе взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно. Стандарт воплощает в себе идеи проголиморфизма и инкапсуляции объектно-ориентированного программирования. В современных версиях Windows COM используется очень широко. На основе COM были реализованы технологии? Microsoft OLE Automation, ActiveX, DCOM, COM+, DirectX, а также XPCOM
* **OLE (англ. Object Linking and Embedding, связывание и встраивание объекта) -** технология связывания и внедрения объектов в другие документы и объекты, разработанные корпорацией Microsoft. Позволяет передавать часть работы от одной программы редактирования к другой и возвращать результаты назад. Например, установленная на персональном компьютере издательская система может послать некий текст на обработку в текстовый редактор, либо некоторое изображение в редактор изображений с помощью OLE-технологии.
* **ActiveX** - фреймворк для определения программных компонентов, пригодных к использованию из программ, написанных на разных языках программирования. Программное обеспечение может собираться из одного или более таких компонентов, чтобы использовать их функциональность. Впервые эта технология была внедрена в 1996 году компанией Microsoft как развитие технологий Component Object Model (COM) и Object Linking and Embedding (OLE) и широко используется в операционных системах семейства Microsoft Windows, хотя сама технология не привязана к операционной системе.

**Достоинства:** можно использовать, например, в браузере как расширение и взаимодействовать с компьютером пользователя, решая проблему одностороннего браузерного взаимодействия.

**Недостаток:** не очень безопасно. Устанавливая компонент из непроверенных источников мы подвергаем риску нашу ОС, т.к. ActiveX имеет к ней прямой доступ.

* **SOAP (от англ. Simple Object Access Protocol - простой протокол доступа к объектам; вплоть до спецификаций 1.2)** - протокол обмена структурированными сообщениями в распределенной вычислительной среде. Первоначально SOAP предназначался в основном для реализации удаленного вызова процедур (RPC). Сейчас протокол используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур. Официальная спецификация последней версии 1.2 протокола никак не расшифровывает название SOAP (перестало быть акронимом). SOAP является расширением протокола XML-RPC. Применяется в системах достаточно крупных, корпоративных, т.к. достаточно тяжел в поддержке и реализации.

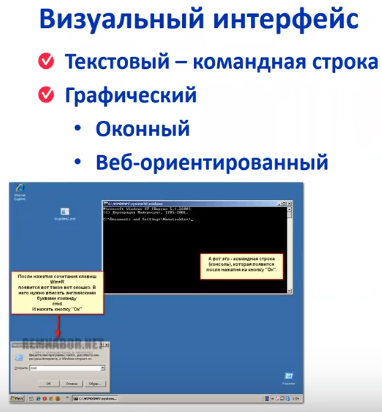
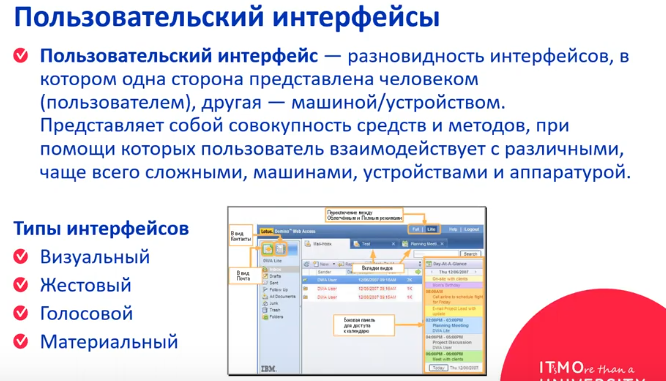
**Основные принципы SOAP:**

* Использование XML для формирования пакетов данных
* Применение HTTP в качестве пассивного наблюдателя
* Формализация API в формате WSDL
* Использование интерфейсов, основанных на объектах и методах
* **REST (сокр. от англ. Representational State Transfer - “передача состояния представления”)** - архитектурный стиль взаимодействия компонентов распределенного приложения в сети. Не является языком программирования или нотацией. Используется удаленный вызов процедур по HTTP.

**Основные принципы REST:**

* Все является ресурсами с уникальным идентификатором (URL)
* Все операции клиента с сервером stateless, т.е. сервер не должен хранить вообще никакой информации о клиенте, следовательно, никакой сессии
* Все запросы можно поделить на 4 типа в соответствии с CRUD, причем каждому типу сопоставляется HTTP метод - Post, Get, Put и Delete
* Вся логика крутится вокруг ресурсов, а не операций

### Пользовательский интерфейс и его типы

**Пользовательский интерфейс** - разновидность интерфейсов, в котором одна сторона представлена человеком (пользователем), другая - машиной/устройством.

Большинство корпоративных систем используют веб-интерфейс.



Сущес

### Принципы проектирования пользовательского интерфейса

Принципы разработки интерфейса — это концепции и представления высокого уровня, которые могут использоваться при проектировании программного обеспечения. Необходимо определить, какой из принципов наиболее важен и приемлем для вашей системы. Три принципа разработки пользовательского интерфейса формулируются так:

1.Контроль пользователем интерфейса;

2.Уменьшение загрузки памяти пользователя;

3.Последовательность пользовательского интерфейса.

На сегодняшний день данные принципы не совсем подходят для графических пользовательских интерфейсов, однако, в скором будущем они бесспорно будут применяться и получат дальнейшее развитие и распространение.

https://moluch.ru/archive/291/66058/

### Прототипирование пользовательского интерфейса, его цель и принципы. Инструменты прототипирования пользовательского интерфейса

**Прототипирование** - быстрая “черновая” реализация базовой функциональности для анализа работы системы в целом. На этапе прототипирования малыми усилиями создается работающая система (возможно, неэффективно, с ошибками, и не в полной мере). Во время прототипирования видна детальная картина устройства системы.

Небольшая реализация того, что будет сделано. Т.е. мы небольшими трудозатратами реализуем систему, просто чтобы показать, что мы будем реализовывать. Это даже не MVP, а некоторое представление.

### Понятие качества. Качественные показатели ИС, критерии оценки качества

**Качество** - совокупность свойств продукции, обусловливающих её пригодность удовлетворять определённые потребности в соответствии с её назначением.

(продукция удовлетворяет ровно те потребности, которые ей предъявляются)

**Управление качеством** - методы и виды деятельности оперативного характера, используемые для выполнения требований к качеству

(Про ИС):

**Качественные показатели** закладываются в систему на этапе проектирования в качестве нефункциональных требований)

Типовые показатели:

* Соответствие требованиям - реализовано то, что закладывалось в тех. задании;
* Удобство использования - юзабилити - должна быть удобная оболочка (зачастую игнорируется - пользователь работает в бекофисе и будет работать с тем, что ему дали (ыхыхых)) (говорит что когда crm разрабатывали - чё-то там делали и выручка увеличивалась;
* Стабильность и надежность работы - sla999, sla9999 - насколько система бессбойно функционирует, в какой промежуток времени доступна и не подвисает ли вся система; (измеримый показатель по времени-доступности, по требованиям тз )
* Скорость работы - скорость отклика интерфейса системы и то насколько система быстро выполняет требуемые от неё операции;
* Масштабируемость - многонодовость (Нода — это любой компьютер, подключенный к блокчейн-сети [распределенной сети]) либо шардирование (расширяет аппаратные мощности или при добавлении новых нод), позволяет системе расти не только за счет наращивания аппаратных мощностей но и при добавлении новых нод; Масштабирование можно добиться за счет грамотной архитектуры (Cloud Ready / Cloud Native - докерские контейнеры работающие параллельно). Масштабироваться должна как часть прикладной логики, так и база данных, потому что при наличии большого количества фронтов база данных продолжала обрабатывать запросы.
* Интегрируемость, расширяемость - на сколько данную систему по функциональности можно расширить. Если система не расширяема, то это сильно затрудняет добавление новых функций. Интегрируемость - возможность работы с другой системой, передача данных со смежной системой, наличие внешнего api;
* Затраты на модернизацию и сопровождение - чем система более уникальна, требует редких специалистов, то её трудно поддерживать. SAP например трудно поддерживать, а вот 1С чела для сопровождения системы можно на изи найти. Облачные варианты - там проблема с сопровождения и модернизации находится на поставщике облачных услуг. Если не модернизируется, или очень много затрат, то лучше просто заменить систему на более успешных конкурентов.

В нефункциональных требованиях прописаны ограничения и условия её работы.

В функциональных - опредяют как система работает, что она реализует.

*Критерии оценки качества.*

Измеряемые (объективные)

* численные показатели

например: скорость открытия страницы в секундах, либо доступность (SLA)

Экспертные (субъективные)

* заключение эксперта
* анкетирование пользователей

юзабилити например не можем измерить объективно

### Риски снижения качества и методы его повышения

Качеством желательно заниматься до этапа тестирования.

Риски снижения качества:

* Низкое качество требований;
* Ошибка в планировании ресурсов/сроков - берем итоговые требования, детализируем, описываем, декомпозируем до конкретных задач.   
  Планирование осуществляется до того момента как подготовлено итоговое тз детальное или расписаны задачи, поэтому возникают ошибки. Для реализации системы потребовалось слишком много времени напрмер. заказ в сжатые сроки, и тупо надеямся что успеем. Но по итогу задача делается ровно столько, сколько запланировали +20% ага.
* Некорректная архитектура ИС/неправильно выбранная технология -   
  Этот пункт не сможете исправить на этапе тестирования, поэтому надо задумываться заранее  
  Пусть запланировали что наша система работает по интернету, а вот у заказчика очень плохое интернет соединение, да ещё и не всегда, поэтому надо чтобы оффлайн работало. Отсюда придется корректировать с Тонкого клиента на Толстый.  
  технологии - использование антипаттернов (вынесение логики в базу данных), неправильных технологий для реализации систем. Реализация системы Java дольше и тяжелее чем Php например.
* низкое качество тестирования - встречаются кампании где тестирование проходит вручную. Приводит к снижению лояльности заказчика. Можно перетестировать, написать автотесты...;
* Изменение требований в процессе разработки ИС/в процессе сдачи ИС/в процессе эксплуатации - про несогласованность доработок - либо это не документируем (тогда получаем изменение логики работы системы, о которой никто не знает),   
  либо если это было документировано, возможно не были учтены все элементы на которое это всё влияет;
* Низкое качество сопровождения - любая система развивается, обновляется, ей требуются какие-то регламентные работы (чистка БД, удаление временных файлов). Если этого нету, или специалисты неквалифицированные, то система будет постепенно деградировать. И качество системы на старте своей жизни придет к тому что она будет неподдерживаемая.;
* Несогласованные доработки/интеграции (видимо тоже самое как и в изменении требований)

Методы повышения качества:

* Регулярное взаимодействие с заказчиком - не только компания, но и отдел продаж, отдел маркетинга. Требования могут меняться, поэтому надо общаться с заказчиком. Если нет взаимодействия, то это либо госзаказ, либо придется всё менять на релизе, и вообще заказчик ожидал другое;
* Письменная фиксация изменений в требованиях - если что-то меняется в требованиях, то это надо фиксировать, иначе будет что система работает одним способом, а заказчик ожидал другое и в документациях написано другое (да и ответственной лицо, с кем договаривались, может измениться);
* Формализация требований к тестированию на этапе проектирования - TDD, BDD - сначала формируем тесты, потом делаем систему (на этапе проектирования знаем что и как будем тестировать (заложили что будет SLA9995, то и думать об этом надо в самом начале и прописывать на этапе проектирования));
* Составление программы и методики испытаний, а также регламентов сопровождения;   
  - методики испытаний - документ, который позволит пройти все необходимые шаги для сдачи системы. Они завершаются либо успешно, либо нет. Без формального описания нельзя систему сдать.

Регламент сопровождения - специалисты могут не знать как сопровождать, потому что например те кто делал могут уже работать в других кампаниях. Главное просто иметь описание как систему сопровождаем.

* Планирование этапа тестирования в рамках общего цикла разработки;   
  когда планируется реализация проекта - закладываете время на первичное описание требований требования, время на разработку и тестирование произойдет где-то там. В итоге можем не уложиться в сроки. Если предлагают сократить тестирование, то вы сокращаете разработку, потому что иначе получите некачественный продукт.
* Применение на регулярной основе методов автоматизированного и ручного тестирования;   
  автоматизированное тестирование - может тестировать интерфейс (через Selenium например). сравнение скринтшотов (есть утилиты до/после).   
  всё взаимодействие происходит по API, дергаем по Mock’ам.
* Проведение регрессионного тестирования и тестирования по подсистемам  
  есть регресс который проводит тестирование всего объема системы, того что надо для выпуска в прод. И есть подсистемы, можно делать тестирования каждый раз как внесли изменения в подсистему, особенно если у нас микросервисы.

### Способы и виды тестирования ИС

**Способы тестирования**

Ручное тестирование:

* Тестирование по требованиям; когда есть требования, смотрим как оно описано и по нему проходимся.
* Тестирование по ПиМИ; по программе и методике испытаний, когда мы   
  создали отдельный документ с программой и методикой испытаний, внесли каких-то тестовых пользователей, описали предусловия, выполняемые шаги (достаточно однозначные).  
  Типа откройте приложуху, нажмите туда-то туда-то.
* Регрессионное тестирование - некое полное тестирование перед выпуском системы

Автоматизированное тестирование:

* Автоматизированное тестирование через пользовательский интерфейс (скрипты или запись действий специалиста по тестированию);

(Selenium, Selenium web driver), либо описание тестов на Java. То есть они проводятся каждую ночь и мы всегда знаем что всё работает.

* Тестирование программных модулей при помощи API

Тестирование по методу белого и черного ящика. Просто отправляем Rest запрос, проверяем выполнение, получаем какой-то результат, артефакты. И тоже имею спустя несколько минут результаты выполнения, если тестов много

Чем выше покрытие автоматических тестов, тем меньше требуется тестировщику для проверки продукта, меньше ручных тестов

**Виды тестирования**

по объектам:

* модульное тестирование (unit тесты)
* интеграционное тестирование - проверяем взаимодействие модулей между собой. Вызываем последовательность сервисов, и можем узнать где ломается тестирование, где что некорректно работает.  
  а вот всё в микросервисах всё в облаках работает, возникает какая-то ошибка и приходиться трейсить по логам - это занимает время.
* системное тестирование - тестирование всей системы целиком. То что видит заказчик.

### Дефекты ИС и принципы работы с ними

**Дефект** - производственный брак.

**Ошибка, bug** - понимание дефекта для сферы IT.

Базовые принципы работы с дефектами.

* Атомарность; (не надо описывать в одном дефекте сразу несколько проблем)
* Четкое описание; (четкая последовательность действий чтобы дефект воспроизвести)
* Верифицируемость; (чем более непонятное описание, тем большая вероятность что его никогда нельзя будет воспроизвести, перепроверить -> дефект будет жить с пользователем вечно)
* Классификация; (относится к логике системы, к документации, к инфраструктурным проблемам, к пользовательским интерфейсам)
* Приоритезация; (позволяет определить время исправления дефекта, потому что не все дефекты срочные - блокирующий дефект например важно исправить, а цвет не тот подождет до след релиза.

Но тут важно смотреть ещё и роль пользователя. [если кнопка не работает или не тот цвет у одного пользователя - случайное индивидуальное явление - но этот чел топ-менеджер, то важно исправить в кратчайшие сроки - VIP саппорт так сказать], хоть приоритет и низкий)

* Связь с первичным требованием/задачей для разработки

Дефект сам по себе не существует в вакууме, если нашли - значит возник когда реализовали какую-то фичу, модуль.

Крупный продукт может реализовываться несколькими командами. Значит понимание к какому требованию относится проблема (чего?), поможет понять в какую команду необходимо передать проблему. А если появился дефект без задачи, значит кто-то сделал функционал, который не задокументирован - тоже трабл.

базис цикла работы с дефектами (дефект не важен - отложили, не можем исправить (платформа не та, вендор сам исправит) - говорим не может исправить и всё)



### Понятие релиза и принципы управления релизами ИС

**Релиз** (версия, сборка) - набор новых и/или измененных конфигурационных единиц, в отношении которых осуществлено тестирование и которые рекомендованы для использования.

**Требования к релизу:**

* Сформулирован состав изменений;
* Проведено регрессионное тестирование;
* Сформирован план внедрения/миграции

По релизу есть документированный состав изменений, те таски которые были выполнены. При планировании обновлений обычно не выделяется время на проведение миграции. Если меняет БД, то много времени миграция бд занимает, поэтому по каждому релизу требуется планировать все миграции, кто ответственный, что требуется выполнить помимо выкладки кода.

Чё-то говорит все юзают ITIL’овские стандарты.



### Отраслевые стандарты подготовки документов при проектировании ИС. ГОСТ, РД.

### Понятие требования. Принципы составления требований.

### Типы требований к ИС и видам обеспечения

### Методы повышения отказоустойчивости и надежности ИС

### Состав документов, подготавливаемых при первичной реализации/модернизации ИС

**Random определения**

**Lifehuck by @NShmendyuk:** Функциональные требования - что делает система  
Нефункциональные требования - как работает система

**Электронная подпись (ЭП, устаревшее название - Электронно-цифровая подпись (ЭЦП))** - информация в электронной форме, которая присоединена к другой информации в электронной форме (подписываемой информации) или иным образом связана с такой информацией и которая используется для определения лица, подписывающего информацию.

**Сертификат ключа проверки электронной подписи** - электронный документ или документ на бумажном носителе, выданные удостоверяющим центром либо доверенным лицом удостоверяющего центра и подтверждающие принадлежность ключа проверки электронной подписи владельцу сертификата ключа проверки электронной подписи.

**Сертификат ключа электронной подписи:**

* Токен (носитель, флешка) может действовать до 3 лет.
* Сертификат выпускается, в основном, на 12 месяцев. Связано это лишь с возможностью получать прибыль на перевыпуске сертификата каждые 12 месяцев
* Выпускается на сертифицированном носителе либо хранится в реестре
* Содержит
  + Идентификатор
  + ФИО
  + Регион
  + Информациях о средствах ЭП
  + Информация об УЦ
  + Сроки действия
  + СНИЛС
  + ИНН

**Удостоверяющий центр** - юридическое лицо или индивидуальный предприниматель, осуществляющие функции по созданию и выдаче сертификатов ключей проверки электронных подписей, а также иные функции, предусмотренные настоящим Федеральным законом. **(примеры: Контур, Ростелеком, Корус консалтинг)**

**Удостоверяющий центр (всего их несколько сотен):**

* Проходит аккредитацию в МинКомСвязи России
* Имеет лицензию ФСБ на СКЗИ
* Работает по опубликованному регламенту
* Регулярно публикует CRL (список отозванных электронных подписей)

**Владелец сертификата ключа проверки электронной подписи** - лицо, которому в установленном настоящим 63-ФЗ порядке выдан сертификат ключа проверки электронной подписи.

**Ключ электронной подписи** - уникальная последовательность символов, предназначенная для создания электронной подписи.

**Ключ проверки электронной подписи** - уникальная последовательность символов, однозначно связанная с ключом электронной подписи и предназначенная для проверки подлинности электронной подписи.

**Регулирует все это 63 Федеральный закон “Об электронной подписи”**

**Виды электронных подписей**

|  | Простая | Неквалифицированная | Квалифицированная |
| --- | --- | --- | --- |
| Внутренний и внешний документооборот | **+** | **+** | **+** |
| Арбитражный суд | **+** | **+** | **+** |
| Документооборот с физическими лицами | **+** | **+** | **+** |
| Госуслуги | **+** |  | **+** |
| Контролирующие органы (ФНС, ФСС, ПФР) |  |  | **+** |
| Электронные торги |  |  | **+** |

**Простая ЭП** - часто применяется внутри организаций для подписания внутренних документов. Может представлять из себя пару логин|пароль для идентификации лица, подписавшего документ. Кроме хранения истории ничего дополнительно не требуется.

**Неквалифицированная ЭП** - генерируется специальным не аккредитованным удостоверяющим центром. Сертификат ключа проверки не действителен без подписания специального договора между сторонами о действительности сертификата.

**Квалифицированная ЭП** - не требует никаких специальных документов для действительности. По силе равнозначен подписи поставленной собственноручно при помощи ручки и печати. Может быть выдана только квалифицированным аккредитованным удостоверяющим центром, стоит денег. Применяется только при совершении серьезных контрактов на десятки миллионов рублей, например, государственными органами.

**Принцип работы ЭП:** Данные -> Хэшируются (обычный MD5) -> Хэш шифруется закрытым ключом, который хранится вместе с сертификатом, происходит связывание с данными -> На другой стороне хэш дешифруется открытым ключом, а данные хэшируются -> Дешифрованный и полученный из данных хэши сравниваются

**Проверка электронной подписи в момент подписания**. Описанный выше принцип работает хорошо пока не окажется так, что у сертификата ключа ЭП истек срок действия и он больше не валиден. С этим можно бороться путем добавления штампа времени на момент подписания. Из специального сервера штампов времени извлекается штамп времени и добавляется в информацию о подписании документов. Таким образом на момент подписания будет всегда известно действовал или уже истек сертификат ключа у контрагента.

**Принцип работы криптопровайдера (CSP)**

* Использование ГОСТ Р 34.10-2001
* Взаимодействие через Microsoft CryptoAPI (в случае Windows)
* Работа со стандартными контейнерами PCS#11, PCS#7 (унифицированный стандарт, благодаря которому контейнер, созданный в одной ОС, можно пересылать куда угодно и как угодно, далее он может быть успешно расшифрован)
* Работа с сертифицированными носителями (USB-флешка со специальным драйвером)
* Обеспечение подписания документов присоединенной или отсоединенной подписью (**отсоединенная подпись** - файл подписи идет отдельно от самого документа, **присоединенная подпись** - ЭП включена в содержимое самого файла с данными, например, в Word-файл. Не ко всем типам документов можно присоединить подпись, в таких случаях используется отсоединенная подпись)
* Работа с форматами сертификата Х.509 и ГОСТ

**Принцип работы криптопровайдера (CSP):**

1. Приложение при подписании документа отправляет документ программной части CSP
2. CSP по специальному протоколу EKE передает хэш документа на аппаратную часть токена
3. На аппаратной части токена происходит шифрование, где сам сертификат никогда не покидает его пределы, а возвращается готовая подпись (с некоторых типов токена сертификат можно извлечь)
4. Программная часть CSP возвращает документ с подписью

**Трансграничное взаимодействие**

1. Вступает во взаимодействие третья сторона. Это может быть УЦ, которому доверяют обе стороны
2. Документ, подписанный в одной стране, пересылается на проверку этому УЦ, который подтверждает, что документ подписан в соответствии со стандартами той страны, куда направляется документ. Это все происходит вручную

**Решения для защиты канала**

* SSL/OpenSSL (подключение происходит по HTTP(S), а дальше шифруется SSL)
* ГОСТ:
  + VipNet
  + Континент

**ГОСТ-решения**: приобретается координатор (обычная одноюнитовая железка, установленная в ЦОДе), к которой можно подключаться по зашифрованному каналу с любого компьютера при наличии специального клиента. Образуется защищенная сеть между координатором и компьютерами, подключенными к нему. При этом возможно отключение компьютера от сети Интернет для большей безопасности. Возможна настройка межсетевого взаимодействия, когда два сегмента сети соединены через координаторы, между которыми установлено доверие. **На VipNet координаторах построена сеть МинЗдрава (каждая больница интегрирована в сеть в рамках региона, дальше через VipNet идет соединение с головным центром МинЗдрава).**

**Требования к оптимизации:**

* Максимальное использование
* Применение архитектурных особенностей
* Учет запаса по ресурсам

Если ПО от вендора, то просто берутся требования к аппаратному обеспечению от вендора. Если же используется какой-то кастомизированный программный продукт, то проводится нагрузочное тестирование и под максимальной нагрузкой определяются аппаратные требования с учетом некоторого запаса (в среднем ~30%). Также следует учесть сезонные пики и либо организовывать CDN (Content Delivery Network, географически распределенная сеть), либо на этот период арендовывать облачные ресурсы.

**Требования к программному обеспечению:**

* **Операционные системы** (десктопы - Windows, Linux, MacOS; мобильные - Android, iOS; серверы - UNIX-подобные ОС, Windows Server, macOS Server (очень редко). При проектировании ИС стоит учитывать лицензионные отчисления, не только на ОС, но и на бд и прочий софт. Linux тоже может быть платным, по подписке, предоставляющей технический саппорт.)
* **Платформы** (в первую очередь стоит рассмотреть готовые платформы при проектировании и под эту платформу уже писать свое решение, используя что-то готовые, добавляя нужные модули и прочее. Потому что самописное решение всегда дорого не только в плане разработки, но и будущей поддержки решения)
* **Контейнеризация** (Docker/k8s. Для пользователя ничего не меняется, но при запросе kubernates как регистратор контейнеров определит какой из контейнеров с аналогичным запущенным сервисом наименее загружен и передаст через контроллер запрос на соответствующую ноду. Нода, обработав запрос вернет ответ пользователю. Еще одно преимущество помимо распределения нагрузки это возможность снизить даунтайм путем, например, поочередного обновления контейнеров. Также кубернейтс при пиках нагрузки способен производить скейлинг и наращивать количество контейнеров на время высокой нагрузки)
* **Виртуализация**