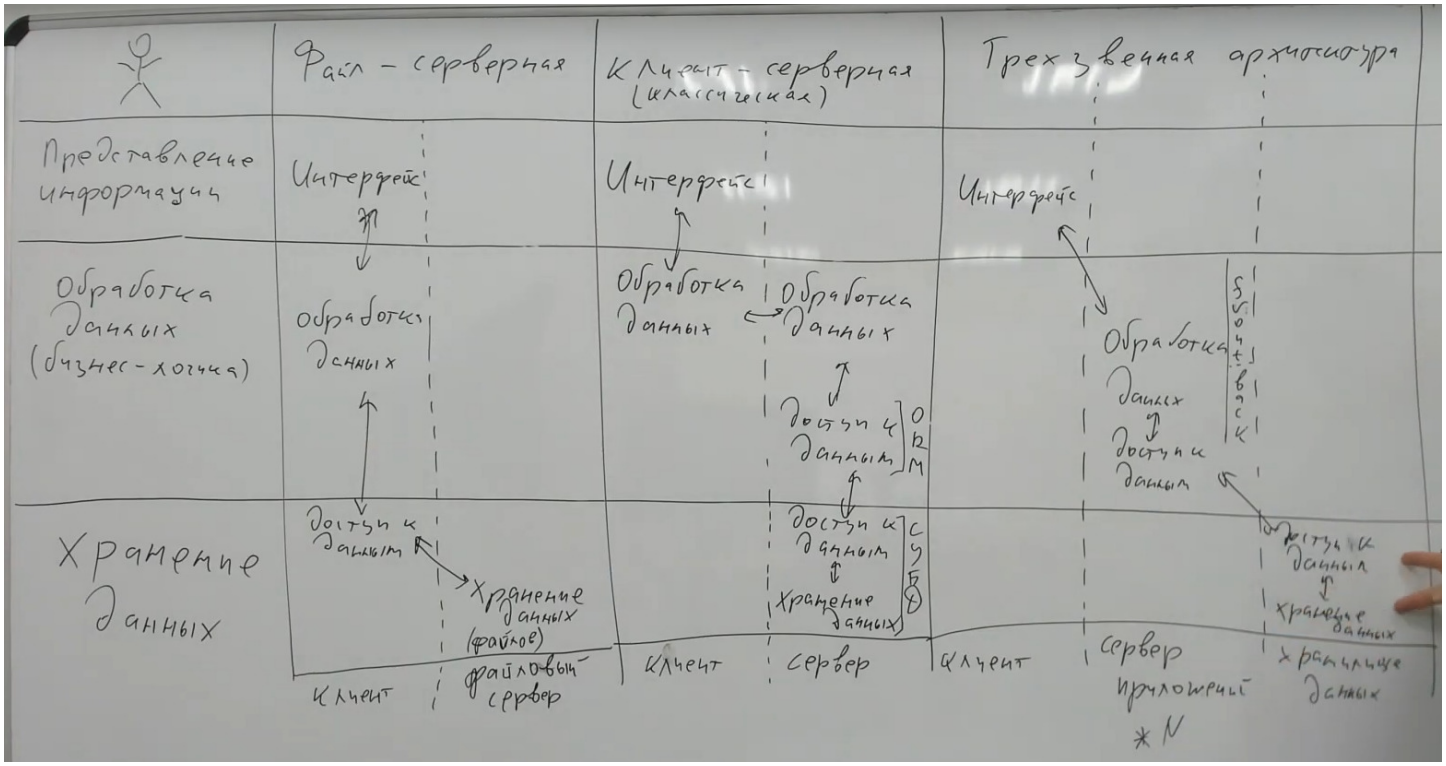
Перейти к описанию и содержанию конспекта.

Ссылка на лекцию



Функциональная архитектура — выделяет функциональные компоненты, функциональны компоненты, как они реализованы, как сгруппированы те или иные функции, кому какой функционал предоставляется и т.д.

Информационная архитектура — говорит от тех информационных объектах и образованных ими информационных потоках, которые определяют решение основной задачи информационной системы — автоматизация процесса сбора, обработки, хранения и передачи информации. *Информационная система направлена автоматизацию задач управления*

На уровнях функциональной и информационной архитектур мы абстрагируемся от технической реализации информационной системы, т. е. от технической реализации ИС.

Не функциональные требования обычно определяют триадой: производительность, надежность, безопасность.

Производительность может измеряться или оцениваться различными характеристиками: время отклика, кол-во операций в единицу времени, эти характеристики могут задаваться как усредненные или предельные, объемы обрабатываемых данных за единицу времени — зависит от задачи, которая ставится (что важно там).

Показатели надежности говорит о том, что мы можем продолжать эксплуатировать систему на протяжении какого-то времени без отказов в работе системы. Показатели могут быть разные: наработка на отказ — количество часов

которых мы ожидаем, что система будет работать с вероятность отказа не превышающий определенные пороговые значения, это может быть время восстановления после отказа, какие-то характеристики с надежностью хранения данных — допустимый процент потерь данных, время за которое мы готовы потерять данные и другие характеристики

Характеристики безопасности: невозможность несанкционированного доступа к данным, несанкционированные изменения, сама система, которая имеет аппаратную составляющую была безопасна для обслуживающего персонала

Кроме триады есть еще задачи: *масштабирование*

Информационная система — по умолчанию, по определению консервативное решение, потому что внедрение очень дорого (не в плане дорого стоит программное обеспечение, работы интеграторов, железа) с точки зрения влияния на бизнес процессы — для того чтобы внедрить систему мы должны подвергнуть большому стрессу всю нашу организационную структуру, весь персонал, подвергнуть риску существующие бизнес процессы. Пока это системы начнет приносить прибыль, какой-то профит с точки зрения снижения издержек пройдет достаточно сложный стрессовый период, когда её внедрения начнет ломать, менять

процессы, поэтом пройдя один раз этот путь и выведя эту систему на стационарный режим, когда она будет дальше работать так, что все ее участники вовлечены в использование ИС, не хотелось бы опять подвергать стрессу, рискам, процессам внедрением еще ИС. Поэтому хотелось бы на протяжении длительного периода времени (годы, пятилетки, десятилетия) не трогать и продолжать использовать. Особенно в таких областях как банковская сфера, управление производствами и т.д.

ИС всегда работает в открытом контексте — внешние окружение влияет на её работу (регаляторы и т.д.). Поэтому хотелось бы очень просто масштабировать, изменять ИС, но не менять саму ИС. Поэтому возникают требования к универсальности.

Масштабирование мешает триаде, т. к. возможны снижения производительности, безопасности, надежности.

Совместимость программная и аппаратная. Например, совместимость с бд, совместимость с железом, совместимость с другими системами и т. д. Удовлетворение этого требования будет мешать всем остальным требованиям.

Могут быть и другие требования: *соответствие отраслевым стандартам и др.*

Т.е. ищем компромиссное решения, т. к. решение, которое удовлетворяло бы всем аспектам невозможно.

Именно здесь и появляются типовые архитектуры. Если абстрагироваться от «внешнего вида», то можно заметить, что многие системы внутри концептуально

похожи. Например, дома: как устроен каркас, как устроен фундамент, как устроено перекрытие, как устроены окна, как устроены проемы и т. д. Найденные компромиссы в решениях фиксировались и отражались в различных учебниках, программах, передача от мастера к подмастерью, т. е. эволюционным образом приходили к некой типовому варианту архитектуры

Еще одно не функциональное, но важное требование — *конечная стоимость решения*

Затраты на персонал, которые будет разрабатывать, поддерживать. Уникальное решение потребует высококвалифицированных специалистов, которые способны разобраться в этом решении и дальше развивать или специальная подготовка кадров нужна ради одного проекта.

См.пример Барсилона собор Саграда Фамилия. Его строят уже больше ста лет

Если будут типовые решения, то дешево подготовить персонал по сопровождению, внедрению, переход пользователей и т. д.

Идеальной архитектуры нет!

У каждой архитектуры есть своя ниша. Решения нужно сравнивать по каким-то принципам. ИС реализует информационную технологию и реализует пять основных функций

сбор, обработка, хранение, передача и представление информации

но раскладывать по 5 процесса не всегда бывает удобно, поэтому часто принимают восприятие информационной системы как три слоя (см. картинку выше):

— Человек

— Слой представления информации

— Слой обработки данных

— Слой хранения данных

По слоям видно, что в какой-то момент происходит переход от слова «данные» к слову «информация». Это связано с тем, что когда мы говорим про данные, то информация так или иначе уже формализована (происходить опись данных, принципы кодирования и т.д.).

На уровне представления мы переходим к человеку, т. к. выше находится человек-пользователь и этот пользователь работает с информацией. Конечному пользователю на конечные протоколы, структуры данных, способы хранения (например, реляционные БД или нет) и т. д. — всё равно, ему не очень важно это всё. Т.к. этот пользователь должен выполнять свою бизнес-функцию, решать какую-то прикладную задачу и исходя из этого пользователь работает с

информацией и ему важно как данные представлены.

Уровень представление инкапсулирование не только процессы представления, но и ввод, т. к. по сути этот уровень является интерфейсом т. к. обеспечивает взаимодействие естественного интеллекта пользователя с программной логикой нашей ИС, поэтому на уровне преставления говорим и про операции ввода, и про вывод.

Передача между уровней и отдельно, часто не выделяется в отдельный слой

Если мы хотим объединить все слои в монолит, то получаем систему формата один пользователь-одна система (standalone). Возникает проблема, что множество пользователей с трудом могут использовать такую систему.

Проблема организации бизнес-процессов на взаимодействии, поэтому одна из основных задач ИС это обеспечить совместную обработку данных, совместную работу пользователей. Это привело к тому, что представление данных и хранение данных нужно разнести, потому что относительно хранения данных мы хотим обеспечить целостность, а относительно представления распределенность. Мы не можем одновременно обеспечить системность, доступность и устойчивость к разделению

Предтеча к клиент-серверной архитектуре является файл-серверная архитектура (см.картинку)

Файл-серверная архитектура

Если много пользователей начнут работать с данными, то они могут привести их в неконсистентное состояние + данные могут быть разделены с точки зрения безопасности, доступа, надежности

хранение данных — это хранение файлов на сервере

доступ уже на стороне клиентского приложения

Преимущества

-можно реализовать многопользовательский режим работы с данными(разные пользователи, работая с разными экземплярами клиента могут работать с одними и теми же данными), но одновременную работу обеспечить не получится (каждое клиентское приложение должно работать с одним и тем же своим файлом)

-дешевизна решения (т. к. на уровне доступа данных к хранению мы можем использовать механизмы самой ФС ОС) как:

1. либо передает файлы на клиент и возвращаем (низкая производительность — передавать весь файл надо. При маленьких файлах сложно обеспечить консистентность)

2. мы используем механизм удаленного доступа к файлу (нужно знать

организацию блоков файла до запроса нужного блока+запрос блокировок нужных частей)

Недостатки:

— Надежность (в случае сбоев или случайных записей можно потерять файл)

— Безопасность (шифрование и т. д.)

— масштабирование (больше клиентов — больше очередей по доступу)

Клиент-серверная архитектура (классическая)

Вопрос доступа переносится на уровень обработки данных (серверу)

(см.рисунок)

Обработка данных раскладывается на клиент и сервер

Доступ раскладывается на клиент и сервер

Хранение на сервере

Что поменялось?

Данные→БД и СУБД

Если нагрузка растет, то иногда переходят на другую субд и используется спец. Решение для работы с субд (ORM, если объектно-ориентированные обработка данных — слой абстракции, который помогает уйти от того как будут данных обрабатываться, т. е. в одну сторону смотрит объектами понятными бизнес-логики, а другой понятной субд)

Почему происходит разделение в обработке данных?

Потому что К-Л арх дальше распадается на два подкласса:

толстый клиент

тонкий клиент

При этом нельзя провести границу где толстый клиент, а где тонкий потому что различие заключается в том, где сконцентрирована обработка данных.

Если концентрирована на стороне сервера, то *тонкий клиент*

Иногда утрировано и некорректно говорят, что на тонком клиенте остается только интерфейс — это не правда, т. к. в большинстве случаев на тонком клиенте большиство обработок производится на стороне клиента (валидации т. д. т. к. создавать решения, где только сервер обрабатывает очень дорого по задержкам)

Толстый клиент это когда всю её обработку или большую её часть переносим на сторону клиента.

Редко можно сказать, когда вся обработка перенесена на сторону клиента, потому что слой взаимодействия с доступом к данным потребует определенной обработки данных (как минимум обеспечить совместимость на уровне программных интерфейсов).

Преимущества

-поддержка полноценной многопользовательской работы с гарантией целостности данных

— повышаем возможности масштабирования

— повышаем возможности по компромиссу целостности данных и пользовательской работы

Недостатки

— стоимость поддержки,

— стоимость разработки

-для толстого клиента: проблема обновлений (нужно обновлять толстые клиенту — вмешивание в работы пользователей), высокие требования к стоимости оборудования клиентов

-для тонкого клиента: высокие требования к стоимости оборудования сервера (вопрос равномерности нагрузок — клиент или сервер ?), проблемы масштабирования (сервер может не выдержать условных 1000 клиентов, т. е. нужно переходить к распределенной архитектуре), клиент чувствительный к каналу связи

-вопрос безопасности на связи между клиентами и сервером

Трехзвенная архитектура

Компромисс в модули противоречий с клиентами и серверами0

(см. рисунок)

Компоненты:

1.клиент

2.сервер приложений

3.хранилище данных

Этим разделением пытаемся достичь:

-производительность (несколько серверов)

-масштабируемость системы (увеличивать серверов обработки данных)

Преимущества:

-разделение на три части обработки

-защита обработки и хранения данных(мб частная сеть)

-масштабирование слоя обработки (но ограничивается хранилищем)

Недостатки:

-стоимость

-дорого поддержки

-проблемы тонкого клиента остаются

Частный случай web-арх.:

клиент — браузер

сервер приложений — публикашион сервере, веб-сервер

сервер -хранение — субд (мб гетерогенная субд)

В такой арх. Часто обработка данных делится на фронт-енд и бэк-енд для того чтобы разделить то, что связано с доступом к данным, их обработки и то, что связано с формирование интерфейсов (браузер). Фронт создает файлы (html/css/js и т. д.) и передает браузеру, т. е. часть фронта всё же на сервере. Бэк реализует вопросы связанны с доступом и вопросы связанных с обработкой данных

Проблемы:

— даже если будет много серверов приложений затык может быть на уровне хранения данных, т.к сервер хранения данных один и тот же

— проблема хранилища — предельные характеристики серверов

— если распределенное хранилище, то упираемся во все известные проблемы хранилища распределенных транзакций

— может перейти от жесткой консистентности к консистентности к конечном случае или

миграция блоков данных по принципам распределенного чтения и со всеми вытекающими последствиями

Далее переход к распределенным архитектурам