## 1. Сфера применения современных информационных и автоматизированных систем.

Информационная система (ИС) — система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию. ИС предназначена для своевременного обеспечения надлежащих людей надлежащей информацией, то есть для удовлетворения конкретных информационных потребностей в рамках определённой предметной области, при этом результатом функционирования информационных систем является информационная продукция — документы, информационные массивы, базы данных и информационные услуги.

АСУ (Автоматизированные системы управления) активно применяются в самых разных сферах жизни и современной промышленности. В частности, они используются в системах освещения, дорожного движения, в системах информации и во всех сферах промышленного хозяйства.

Основной целью применения и использования АСУ выступает повышение эффективности и использования возможностей каждого объекта. Такие системы позволяют быстро и эффективно проводить анализ работы объекта, на основе полученных данных специалисты могут принять определенные решения и наладить производственный процесс.

## 2. Применение современного программного и аппаратного обеспечения информационных систем для решения прикладных задач.

В зависимости от функций, выполняемых программным обеспечением, его можно разделить на 2 группы: системное программное обеспечение и прикладное программное обеспечение.

Системное ПО организует процесс обработки информации на компьютере и обеспечивает нормальную рабочую среду для прикладных программ. Системное ПО настолько тесно связано с аппаратными средствами, что его иногда считают частью компьютера.

В состав системного ПО входят:

- операционные системы;

- сервисные программы;

- трансляторы языков программирования;

- программы технического обслуживания.

Операционная система (ОС) - это совокупность программ, управляющая аппаратной частью компьютера, его ресурсами (оперативной памятью, местом на дисках), обеспечивающая запуск и выполнение прикладных программ, автоматизацию процессов ввода/вывода. Без операционной системы компьютер мертв. ОС загружается при включении компьютера.

Прикладное ПО предназначено для решения конкретных задач пользователя и организации вычислительного процесса информационной системы в целом.

Прикладное ПО позволяет разрабатывать и выполнять задачи (приложения) пользователя по бухгалтерскому учету, управлению персоналом и т.п.

Прикладное программное обеспечение работает под управлением системного ПО, в частности операционных систем. В состав прикладного ПО входят:

- пакеты прикладных программ (ППП) общего назначения;

- пакеты прикладных программ функционального назначения.

ППП общего назначения - это универсальные программные продукты, предназначенные для автоматизации разработки и эксплуатации функциональных задач пользователя и информационных систем в целом.

К этому классу ППП относятся:

- редакторы текстовые (текстовые процессоры) и графические;

- электронные таблицы;

- системы управления базами данных (СУБД);

- интегрированные пакеты;

- Case-технологии;

- оболочки экспертных систем и систем искусственного интеллекта.

## 3. Решение задач моделирования при помощи информационных систем.

**Модель** (лат. “modulus” – мера) – объект-заместитель объекта-оригинала, обеспечивающий изучение некоторых свойств последнего; упрощенное представление системы для её анализа и предсказания, а также получения качественных и количественных результатов, необходимых для принятия правильного управленческого решения.

При решении конкретной задачи, когда необходимо выявить определённое свойство изучаемого объекта, модель оказывается не только полезным, но и порой единственным инструментом исследования. Один и тот же объект может иметь множество моделей, а разные объекты могут описываться одной моделью.

Единая классификация видов моделей затруднительна в силу многозначности понятия “модель” в науке и технике. Её можно проводить по различным основаниям: по характеру моделей и моделируемых объектов; по сферам приложения и др.

**Моделирование** – представление объекта моделью для получения информации о нём путём проведения экспериментов с его моделью.

Под термином “*моделирование*” обычно понимают процесс создания точного описания системы; метод познания, состоящий в создании и исследовании моделей.

Моделирование облегчает изучение объекта с целью его создания, дальнейшего преобразования и развития. Оно используется для исследования существующей системы, когда реальный эксперимент проводить нецелесообразно из-за значительных финансовых и трудовых затрат, а также при необходимости проведения анализа проектируемой системы, т.е. которая ещё физически не существует в данной организации.

Для формирования модели используются:

·-структурная схема объекта;

·-структурно-функциональная схема объекта;

·-алгоритмы функционирования системы;

·-схема расположения технических средств на объекте;

·-схема связи и др.

Все *модели* можно разбить на два больших класса: *предметные* (материальные) и *знаковые* (информационные).

Для проектирования ИС используют *информационные модели*, представляющие объекты и процессы в форме рисунков, схем, чертежей, таблиц, формул, текстов и т.п.

***Информационная модель*** – это модель объекта, процесса или явления, в которой представлены информационные аспекты моделируемого объекта, процесса или явления.

Она является основой разработки моделей ИС.

Для создания описательных *текстовых информационных моделей* обычно используют *естественные языки*.

Наряду с естественными языками (русский, английский и т.д.) разработаны и используются *формальные языки*: системы счисления, алгебра высказываний, языки программирования и др.

Основное отличие формальных языков от естественных состоит в наличие у формальных языков не только жёстко зафиксированного алфавита, но и строгих правил грамматики и синтаксиса.

С помощью формальных языков строят информационные модели определённого типа – формально-логические модели.

При изучении нового объекта сначала обычно строится его описательная модель, затем она формализуется, те есть выражается с использованием математических формул, геометрических объектов и т.д.

Процесс построения информационных моделей с помощью формальных языков называют ***формализацией***.

Модели, построенные с использованием математических понятий и формул, называют *математическими моделями*.

Под ***моделью информационной системы*** в общем случае понимается формализованное описание системы на определенном уровне абстракции. Каждая модель определяет конкретный аспект системы, использует набор диаграмм и документов заданного формата, а также отражает точку зрения и является объектом деятельности различных людей с конкретными интересами, ролями или задачами.

Под термином ***«моделирование»*** понимается процесс создания формализованного описания системы в виде совокупности моделей. Особенно трудным оказывается описание систем средней сложности, к которым относятся информационные системы управления предприятием. С точки зрения человека, эти системы описать достаточно трудно, потому что они настолько велики, что практически невозможно перечислить все их компоненты со своими взаимосвязями, и в то же время недостаточно велики для применения общих упрощающих предположений (как это принято в физике). Неспособность дать простое описание, а, следовательно, и обеспечить понимание таких систем делает их проектирование и создание трудоемким и дорогостоящим процессом и повышает степень их ненадежности. С ростом технического прогресса адекватное описание систем становится все более актуальной проблемой.

Модель должна давать полное, точное и адекватное описание системы, имеющее конкретное назначение. Это назначение, называемое ***целью модели***, вытекает из формального определения модели: М есть модель системы S, если М может быть использована для получения ответов на вопросы относительно S с точностью А.

Целью модели является получение ответов на некоторую совокупность вопросов. Эти вопросы неявно присутствуют (подразумеваются) в процессе анализа и, следовательно, они руководят созданием модели и направляют его. Это означает, что сама модель должна будет дать ответы на эти вопросы с заданной степенью точности. Если модель отвечает не на все вопросы или ее ответы недостаточно точны, то говорят, что модель не достигла своей цели.

Цель моделирования - понять и изучить качественную и количественную природу явления, отразить существенные для исследования черты явления в пригодной для использования в практической деятельности форме. При моделировании важно следить за адекватностью отображения свойств системы на построенную модель.

Для информационной системы строится 2 вида моделей:

модели «AS-IS» («как есть»), отражающие существующее на момент обследования положение дел в организации и позволяющие понять, каким образом функционирует данная организация, а также выявить узкие места и сформулировать предложения по улучшению ситуации;

модели «AS-TO-BE» («как должно быть»), отражающие представление о новых процессах и технологиях работы организации.

Сложные системы характеризуются выполняемыми процессами (функциями), структурой и поведением во времени. Для адекватного моделирования этих аспектов в автоматизированных информационных системах различают функциональные, информационные и поведенческие модели, пересекающиеся друг с другом.

**Функциональная** модель системы описывает совокупность выполняемых системой функций, характеризует, морфологию системы (ее построение) — состав функциональных подсистем, их взаимосвязи.

**Информационная** модель отражает отношения между элементами системы в виде структур данных (состав и взаимосвязи).

**Поведенческая (событийная)** модель описывает информационные процессы (динамику функционирования), в ней фигурируют такие категории, как состояние системы, событие, переход из одного состояния в другое, условия перехода, последовательность событий.

**Графические (визуальные) модели** представляют собой средства для визуализации, описания функциональной структуры системы, последовательности выполняемых действий, передачи информации между функциональными процессами, выявления отношений между данными.

## 4. Решение задач интеллектуальной обработки данных при помощи информационных и автоматизированных систем.

Интеллектуальная обработка информации − использование цифровых технологий для решения задач, возникающих на различных этапах управления данными в организации: распознавание документов, преобразование в структурированный вид, классификация, анализ информации и ее поиск в различных источниках.

Информационная обработка(вроде неплохая статья)

https://sudexpa.ru/articles/perspektivy-sistem-intellektualnoi-obrabotki-dannykh-i-oblachnye-tekhnologii/

Автоматизированная обработка

еще статью кину

https://cyberleninka.ru/article/n/intellektualnye-avtomatizirovannye-sistemy-obrabotki-informatsii-i-upravleniya-problemy-razrabotki-i-issledovaniya/viewer

Назначение и область применения

Интеллектуальный анализ измерительной информации используется для выявления скрытых закономерностей или зависимостей между переменными, характеризующими результаты измерений каких-либо процессов в технических или естественных системах, в том числе, когда измерения представлены большими массивами данных.

Системы интеллектуального анализа позволяют определить состояние анализируемого технического объекта, сделать заключение о качестве его функционирования, дать рекомендации по поиску и устранению неисправностей его технических подсистем или определить состояние анализируемой естественной системы.

Системы интеллектуального анализа и обработки измерительной информации применяются для определения состояний и анализа функционирования ракетно-космических и авиационных комплексов, сложных технических систем, в том числе автоматизированных систем управления технологическими процессами, а также для мониторинга окружающей среды, оценки сейсмичности и т.п.

#### **Функциональные возможности:**

1. Планирование измерительного эксперимента:
   * планирование измерительного эксперимента для станций приема телеметрической информации (ТМИ);
   * планирование измерительного эксперимента для радиолокационных станций (РЛС);
   * планирование измерительного эксперимента для оптических измерительных средств;
   * планирование измерительного эксперимента для оптимального получения траектории движения объекта по информации НАП.
2. Сбор информации с измерительных средств (ИС):
   * сбор и регистрация измерительной информации от ИС;
   * оптико-электронных станций траекторного измерительного комплекса (ОЭС ТИК);
   * сейсмоакустических станций (САС);
   * радиолокационных станций;
   * ММКОС «Сажень-ТМ»;
   * сбор телеметрической информации (ТМИ).
3. Обработка измерительной информации (ИИ) навигационной аппаратуры потребителя (НАП):
   * точечное решение навигационной задачи в различных режимах;
   * расчет ионосферных и тропосферных поправок к измеряемым параметрам;
   * расчет параметров движения навигационных космических аппаратов (НКА);
   * первичный анализ и отбраковка аномальных измерений НАП;
   * исключение недостоверных измерений по отдельным навигационным космическим аппаратам (НКА);
   * пересчет траекторий в заданные системы координат и временные шкалы;
   * построение сглаженной траектории и расчет изохронных вариаций.
4. Обработка измерительной информации (ИИ) в реальном масштабе времени:
   * построение активного и пассивного участка траектории методом динамической фильтрации;
   * расчет траекторий отделяющихся частей с использованием аэромассгабаритных характеристик;
   * прогноз точек падения отделяющихся частей;
   * расчет аварийных трасс;
   * расчет параметров орбиты выводимого объекта;
   * получение и расшифровка измерительной информации НАП;
   * прием и обработка команд системы управления при построении опытной траектории;
   * совместная обработка измерений РЛС и НАП на активном и пассивном участках траектории;
   * независимое сглаживание параметров траектории при наличии разрывов;
   * уточнение начальных условий по измерениям пассивного участка методом наименьших квадратов;
   * уточнение начальных условий и параметров орбиты по измерениям, полученных на витках;
   * прогнозирование эллипсов рассеивания отделяющихся частей.
5. Совместная послесеансная обработка измерительной информации (ИИ):
   * приведение измерений различных измерительных средств на заданные моменты времени;
   * статистический анализ ИИ и отбраковка аномальных измерений;
   * расчет кинематических параметров траектории движения объекта с привлечением всей доступной ИИ;
   * построение аварийной траектории полета в случае аварийного отключения двигателя на активном участке траектории;
   * расчет среднеквадратического отклонения (СКО) кинематических параметров движения объектов;
   * определение отклонений рассчитанных координат траектории объекта от опорной траектории;
   * расчет дополнительных параметров движения объектов (углов наклона вектора скорости, коэффициентов перегрузок, скоростного напора и других) и оценки их СКО;
   * задачи расчета оценок возмущающих факторов баллистического полета;
   * определение начальных условий движения и их СКО на заданную высоту и на заданное время;
6. Баллистический анализ траекторий движения объекта наблюдения:
   * определение расчётных допустимых отклонений параметров траектории;
   * определение отклонений опытных КПТ;
   * анализ кинематических параметров траектории;
   * расчет кинематических параметров траектории на ПУТ;
   * расчет аварийной трассы для точек траектории
   * выбор аварийной трассы;
   * ввод опытных параметров аварийной траектории.

#### **Предлагаемые услуги:**

* разработка и поставка программно-аппаратных комплексов автоматизированной статистической и интеллектуальной обработки различных типов данных большого объема;
* разработка программных комплексов и систем обработки данных с целью очистки измерительной информации от шума;
* разработка программных комплексов и систем анализа измерительной информации для выявления скрытых закономерностей или зависимостей контролируемых процессов в технических или естественных системах;
* разработка программных комплексов и систем обработки траекторной информации, поступающей от различных видов измерительных средств для широкого спектра объектов наблюдения;
* планирование измерительных экспериментов для различных условий с учетом особенностей работы используемых измерительных средств;
* разработка программных комплексов и систем обработки «сырой» информации от навигационной аппаратуры потребителя в реальном масштабе времени путем решения навигационной задачи;
* разработка программных комплексов и систем совместной обработки информации от нескольких измерительных средств в реальном масштабе времени и построение траектории методом динамической фильтрации;
* разработка программных комплексов и систем полной послесеансной обработки измерительной информации с вычислением и прогнозированием параметров движения объекта на различных участках траектории;
* поставка распределенных автоматизированных систем различной степени сложности с возможностью интеграции с существующими системами;
* авторское сопровождение, технический и гарантийный надзор, регламентное обслуживание и ремонт, в том числе, в послегарантийный период;
* обучение и консультирование специалистов.

## 5. Методы и инструменты сопряжение программного и аппаратного обеспечения ИС и АВ.

https://habr.com/ru/post/281887/

есть супер большая статья, хуй знает, поможет или нет. копировать ее не буду, сюда приведу другое. вопрос уебанский

Современные информационные технологии обычно реализуются в рамках сложных эргатических систем. В широком смысле в них можно выделить следующие основные элементы [5, 6]:

1) человеческий (социальный) компонент (разработчики, консультанты, аналитики, программисты, пользователи, обслуживающий персонал);

2) математическое и алгоритмическое обеспечения, в основном, это – используемые математические методы, модели и алгоритмы;

3) лингвистическое обеспечение, т.е. совокупность используемых языков программирования;

4) программное обеспечение, состоящее из системного (общего) и прикладного программного

обеспечения;

5) технические (аппаратные) средства, т.е. средства вычислительной, коммуникационной и организационной техники;

6) информационное обеспечение, т.е. файлы с данными об объектах, базы данных и т.п.;

7) организационное, правовое и методическое обеспечения, включающие инструктивные и нормативно-методические материалы по организации работы управленческого и технического персонала в рамках конкретной информационной технологии.

Часто в качестве компонентов ИС рассматриваются платформа, интерфейс, информационно- командная среда, база знаний, модели предметной области и др. Платформа определяется компьютерной и операционной системами, на которых можно установить конкретную ИТ. Наиболее распространенные компьютерные системы: ПК, рабочие станции, серверы, миникомпьютеры, ноутбуки, карманные ПК, тонкие клиенты (сетевые компьютеры), мэйнфреймы и др. Широко используются операционные системы: DOS, Windows 3.xx, Windows 9x Windows NT\*, OS/2\*, разновидности UNIX\*, Linux, MacOS, разновидности LAN\*, Novell Net Ware\* (здесь знаком "\*" выделены сетевые операционные системы).

В узком смысле в информационных системах выделяют две основные части – технические и программные средства. Технические средства (Hardware), включающие вычислительную, периферийную и коммуникационную технику, обеспечивают прием и передачу трех основных видов информации (речь, печатный текст, графика) в статике и динамике с использованием трех чувств восприятия человека (слух осязание, зрение). Следует заметить, что в последние годы ведутся работы по расширению возможностей ИТ в направлении передачи информации, например, в виде запаха с применением обоняния.

Программные, информационные и сервисные средства (Software) обеспечивают обработку данных и, как уже отмечалось, состоят из общего и прикладного программного обеспечения (ПО), а также программных документов, необходимых для эксплуатации этих программ. Общее программное обеспечение включает операционные системы (ОС), системы программирования и программы технического обеспечения, которые представляют сервис для эксплуатации компьютера, выявления ошибок при сбоях, восстановления испорченных программ и данных.

Операционные системы подразделяют на однопрограммные (MS DOS, SKP), они поддерживают пакетный или диалоговый режимы обработки информации, многопрограммные (DOS 7.0, WINDOWS, UNIX, OS/2), которые позволяют совмещать диалоговую и пакетную технологии обработки информации, и многопользовательские или сетевые ОС (INTERNET, NOVEL, ORACLE, NetWare), которые осуществляют удаленную обработку в сетях, а также диалоговую и пакетную технологию на рабочем месте.

Примерами прикладного ПО являются: программы САПР (CAD/CAM/CAE), ПО для управления производственными (технологическими) процессами, бухгалтерские и складские программы, офисные приложения, графические системы, системы управления проектами, ПО для архивирования, СУБД средства разработки ПО, программы электронной почты, браузеры Internet и др. Прикладное ПО состоит из отдельных прикладных программ или пакетов, называемых приложениями.

При выборе и совершенствовании прикладного ПО рекомендуется использовать концепцию открытых систем для облегчения обмена и совместного использования информации, обеспечивать максимально возможный уровень интеграции между системами как внутри организации, так и с внешними участниками, применять коммерческие программные продукты, позволяющие предоставлять результаты работ в стандартном виде. Прикладное ПО должно обеспечивать решения всего необходимого комплекса задач, в том числе проектно-конструк-торских, производственных, управления данными, обслуживания и т.д.

Применительно к программным продуктам иногда говорят о конфигурации системы, включающей различные функциональные блоки, например: бухгалтерский учет и отчетность, складская логистика, кадры и зарплата, учет движения финансовых средств и т.д.

Сетевые ИС и ИТ в качестве компонентов включают сети различных уровней, их компонентами

являются:

1) аппаратные средства (сетевые адаптеры, маршрутизаторы, средства телекоммуникации и т.д.);

2) системное программное обеспечение (ОС, СУБД и т.п.);

3) инструментальное программное обеспечение (алгоритмические языки, системы программирования, языки спецификаций, технология программирования);

4) комплектация узлов хранения и переработки информации.

В корпоративных информационных системах выделяют две относительно независимые части: компьютерную инфраструктуру организации (корпоративную сеть) и комплекс взаимосвязанных функциональных подсистем, обеспечивающих решение задач организации и достижение ее целей. В свою очередь корпоративная сеть в качестве составляющих включает следующие инфраструктуры: сетевую, телекоммуникационную, программную, информационную и организационную [9]. Важную роль в использовании ИТ играет интерфейс, т.е. технология общения пользователя с компьютером и взаимодействия частей компьютера. Информационно-командная среда представляет собой совокупность программного и информационного обеспечения и определенного стандарта интерфейса. Важнейшим компонентом интеллектуальных ИС и, в частности, экспертных систем, является база знаний. База знаний (БЗ) есть совокупность знаний, хранящихся в памяти ЭВМ. В БЗ выделяют интенсиональную (знания о чем–то "вообще") и экстенсиональную (знания о чем–то "конкретно", наполненные оболочки, т.е. базы данных) части. Другими словами, БЗ представляет отображение предметной области и включает в себя базу данных с директивной информацией (плановые задания, режимы работы, научно-техническую информацию и т.д.). С помощью моделей предметной области в виде совокупности описаний обеспечивается взаимопонимание между пользователями, т.е. специалистами предприятия и разработчиками и др.

Распространенный термин – архитектура ИС не имеет четкого определения. Обычно под архитектурой автоматизированной системы понимают ее описание на некотором общем уровне, включающее следующие сведения: принцип действия, диапазон возможностей, конфигурация и взаимное соединение основных узлов, пользовательские возможности программирования, средства пользовательского интерфейса, организация памяти, операций ввода-вывода и управления, подробная структурная (или принципиальная) схема [10]. Архитектура информационной системы отражает концепцию взаимосвязи ее элементов, спецификацию сопряжения системы с пользователями и внутренних ее компонентов между собой, она включает компоненты логической, физической и программной структур [2]. Более четко понятие архитектуры формулируется применительно к конкретным технологиям и системам.

При описании архитектуры программного продукта класса конструктор (трансформер) ука

## 6. Методы комплексирование программного и аппаратного обеспечения ИС и АВ.

Цель процесса комплексирования программных средств заключается в объединении программных блоков и программных компонентов, создании интегрированных программных элементов, согласованных с проектом программных средств, которые демонстрируют, что функциональные и нефункциональные требования к программным средствам удовлетворяются на полностью укомплектованной или эквивалентной ей операционной платформе.

В результате успешного осуществления процесса комплексирования программных средств:

a. разрабатывается стратегия комплексирования для программных блоков, согласованная с программным проектом и с расположенными по приоритетам требованиями к программным средствам;

b. разрабатываются критерии верификации для программных составных частей, которые гарантируют соответствие с требованиями к программным средствам, связанными с этими составными частями;

c. программные составные части верифицируются с использованием определенных критериев;

d. программные составные части, определенные стратегией комплексирования, изготавливаются;

e. регистрируются результаты комплексного тестирования;

f. устанавливается согласованность и прослеживаемость между программным проектом и программными составными частями;

g. разрабатывается и применяется стратегия регрессии для повторной верификации программных составных частей при возникновении изменений в программных блоках (в том числе, в соответствующих требованиях, проекте и кодах).

При реализации проекта необходимо осуществлять следующие виды деятельности и задачи в соответствии с принятыми в организации политиками и процедурами в отношении процесса комплексирования программных средств.

Исполнитель должен разработать план комплексирования для объединения программных блоков и программных компонентов в программную составную часть. План должен включать требования к тестированию, процедуры, данные, обязанности и графики работ. План должен быть документально оформлен.

Исполнитель должен объединить программные блоки, программные компоненты и тесты, поскольку они разрабатываются в соответствии с планом комплексирования. Должны быть гарантии, что каждая такое объединение удовлетворяет требованиям к программной составной части и что она комплексируется при завершении этого действия. Результаты комплексирования и тестирования должны быть документально оформлены.

Должна быть разработана стратегия регрессии для применения повторной верификации программных элементов в случае, когда изменения проводятся в программных блоках, включая соответствующие требования, проект и коды.

Исполнитель должен обновлять пользовательскую документацию по мере необходимости.

Исполнитель должен разработать и документально оформить для каждого квалификационного требования к программной составной части комплект тестов, тестовых примеров (входов, результатов, критериев тестирования) и процедур тестирования для проведения квалификационного тестирования программных средств. Разработчик должен гарантировать, что после комплексирования программная составная часть готова к квалификационному тестированию.

Исполнитель должен оценить план комплексирования, проект, код, тесты, результаты тестирования и пользовательскую документацию, учитывая следующие критерии:

a. прослеживаемость к системным требованиям;

b. внешняя согласованность с системными требованиями;

c. внутренняя согласованность;

d. тестовое покрытие требований к программной составной части;

e. приспособленность используемых методов и стандартов тестирования;

f. соответствие с ожидаемыми результатами;

g. осуществимость квалификационного тестирования программных средств;

h. осуществимость функционирования и сопровождения.

## 7. Методики проектирования программного обеспечения ИС и АС.

Проектирование

Этап проектирования дает ответ на вопрос "Как (каким образом) система будет

удовлетворять предъявленным к ней требованиям?". Задачей этого этапа является

исследование структуры системы и логических взаимосвязей ее элементов, причем здесь

не рассматриваются вопросы, связанные с реализацией на конкретной платформе.

Проектирование определяется как "(итерационный) процесс получения логической

модели системы вместе со строго сформулированными целями, поставленными перед

нею, а также написания спецификаций физической системы, удовлетворяющей этим

требованиям". Обычно этот этап разделяют на два подэтапа:

1 Проектирование архитектуры ПО, включающее разработку структуры и

интерфейсов компонент, согласование функций и технических требований к

компонентам, методам и стандартам проектирования, производство отчетных

документов;

2 Детальное проектирование, включающее разработку спецификаций каждой

компоненты, интерфейсов между компонентами, разработку требований к тестам и

плана интеграции компонент.

На этапе проектирования прежде всего формируются модели данных. Проектировщики в

качестве исходной информации получают результаты анализа. Построение логической и

физической моделей данных является основной частью проектирования базы данных.

Полученная в процессе анализа информационная модель сначала преобразуется в логическую,

а затем в физическую модель данных.

Параллельно с проектированием схемы базы данных выполняется проектирование процессов,

чтобы получить спецификации (описания) всех модулей ИС. Оба эти процесса проектирования

тесно связаны, поскольку часть бизнес-логики обычно реализуется в базе данных (ограничения,

триггеры, хранимые процедуры). Главная цель проектирования процессов заключается в

отображении функций, полученных на этапе анализа, в модули информационной системы. При

проектировании модулей определяют интерфейсы программ: разметку меню, вид окон, горячие

клавиши и связанные с ними вызовы.

Конечными продуктами этапа проектирования являются:

схема базы данных (на основании ER-модели, разработанной на этапе анализа);

набор спецификаций модулей системы (они строятся на базе моделей функций).

2 Структурно-функциональное моделирование ИС. Основная идея структурного подхода заключается в декомпозиции, т. е. разбиении системы на функциональные подсистемы. Структурный подход применим при проектировании ИС, где требуется получить представление о технологическом процессе (ТП) производства изделия. Декомпозиция системы до атомарного уровня (элементарной операции) может нарушить принцип абстрагирования, т. е. выделения существенных и отвлечения от несущественных аспектов системы. Для получения необходимой информации о ТП производства рассматриваемого объекта создадим структурно-функциональную модель «как есть» в нотации графического моделирования IDEF0. На верхнем уровне система представляет собой контекстную диаграмму (рис. 2). На вход поступает пластина с кристаллами и корпус будущей микросхемы, на выходе получают готовое изделие или брак. В качестве документа управления выступает маршрутная карта ТП изготовления МКМ, в качестве механизмов — оборудование и оснастка, а также персонал участка сборки.



3 Объектно-ориентированное моделирование ИС. Объектно-ориентированный подход предполагает оперирование «объектом», обладающим некоторыми атрибутами и способным выполнять определённые операции. При этом повышается унификация разработки и ее пригодность для повторного использования. ИС строится на основе стабильных промежуточных описаний, что упрощает внесение изменений [2]. Вначале удобно представить систему в виде модели вариантов использования (рис. 5). В числе прецедентов отразим будущие классы, выявленные в ходе функционального моделирования. В проектируемую ИС входит экспертная система (ЭС), способная заменить технолога-эксперта во время синтеза технологического процесса.

Сюда входят разныее UML диаграммы(диаграммы классов, вариантов использования, взаимодействия, последовательности, развёртывания, состояния и тд.) Потом DFD диграмма, то есть диаграмма потоков данных. IDEF0 и тд диаграммы.

## 8. Методики проектирования аппаратного обеспечения ИС и АС.

Смотри 7 вопрос, ответы очень схожи.

## 9. Фазы жизненного цикла программного обеспечения ИС и АС.

**Жизненный цикл** является моделью создания и использования АИС, которая отражает различные состояния системы с момента возникновения в данном комплексе средств до момента его полного выхода из употребления.

В настоящее время известны и используются следующие модели жизненного цикла:

* **Каскадная модель** ( рис. 2.1) предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе.
* **Поэтапная модель с промежуточным контролем** ( рис. 2.2). Разработка ИС ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах; время жизни каждого из этапов растягивается на весь период разработки.
* **Спиральная модель** ( рис. 2.3). На каждом витке спирали выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка.Особое внимание уделяется начальным этапам разработки - анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования).

Для АИС условно выделяют следующие основные этапы их жизненного цикла:

1. **анализ** — определение того, что должна делать система;

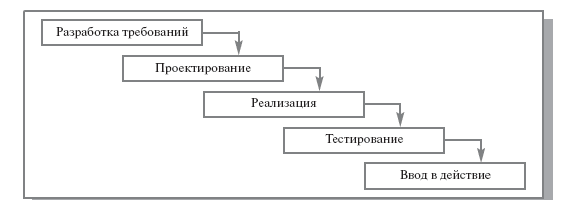
2. **проектирование**— определение того, как система будет функционировать: прежде всего спецификация подсистем, функциональных компонентов и способов их взаимодействия в системе;

3. **разработку** — создание функциональных компонентов и отдельных подсистем, соединение подсистем в единое целое;

4. **тестирование** — проверку функционального и параметрического соответствия системы показателям, определенным на этапе анализа;

5. **внедрение** — установку и ввод системы в действие;

6. **сопровождение** — обеспечение штатного процесса эксплуатации системы на предприятии заказчика.

**Жизненный цикл ИС можно представить как ряд событий, происходящих с системой в процессе ее создания и использования.**

1. **Основные процессы:**
   * **приобретение;**
   * **поставка;**
   * **разработка;**
   * **эксплуатация;**
   * **сопровождение.**
2. **Вспомогательные процессы:**
   * **документирование;**
   * **управление конфигурацией;**
   * **обеспечение качества;**
   * **разрешение проблем;**
   * **аудит;**
   * **аттестация;**
   * **совместная оценка;**
   * **верификация.**
3. **Организационные процессы:**
   * **создание инфраструктуры;**
   * **управление;**
   * **обучение;**
   * **усовершенствование.**

## 10. Фазы жизненного цикла аппаратного обеспечения ИС и АС.

По предыдущему вопросу можно ответить, но типо аппаратное обеспечение живёт дольше, а так у них много схожих моментов, в интернете ничо не нашёл

## ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНЫХ СРЕДСТВ ВИКИ (СЛЕГКА ВИДОИЗМЕНИТЬ)

Жизненный цикл программного обеспечения (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

## 11. Методы обеспечения качества программного обеспечения ИС и АС.

Как контролировать качество системы? Как точно узнать, что программа делает именно то, что нужно, и ничего другого? Как определить, что она достаточно надежна, переносима, удобна в использовании? Ответы на эти вопросы можно получить с помощью процессов верификации и валидации.

***Верификация*** обозначает проверку того, что ПО разработано в соответствии со всеми требованиями к нему, или что результаты очередного этапа разработки соответствуют ограничениям, сформулированным на предшествующих этапах.

***Валидация*** — это проверка того, что сам продукт правилен, т.е. подтверждение того, что он действительно удовлетворяет потребностям и ожиданиям пользователей, заказчиков и других заинтересованных сторон.

Эффективность верификации и валидации, как и эффективность разработки ПО в целом, зависит от полноты и корректности формулировки требований к программному продукту.

Основой любой системы обеспечения качества являются методы его обеспечения и контроля. ***Методы обеспечения качества*** [9] представляют собой техники, гарантирующие достижение определенных показателей качества при их применении. Мы будем рассматривать подобные методы на протяжении всего курса.

***Методы контроля качества*** позволяют убедиться, что определенные характеристики качества ПО достигнуты. Сами по себе они не могут помочь их достижению, они лишь помогают определить, удалось ли получить в результате то, что хотелось, или нет, а также найти ошибки, дефекты и отклонения от требований. Методы контроля качества ПО можно классифицировать следующим образом.

Методы и техники, связанные с выяснением свойств ПО во время его работы. Это, прежде всего, все виды *тестирования*, а также *профилирование* и измерение количественных показателей качества, которые можно определить по результатам работы ПО — эффективности по времени и другим ресурсам, надежности, доступности и пр.

Методы и техники определения показателей качества на основе симуляции работы ПО с помощью моделей разного рода.

К этому виду относятся проверка на моделях (model checking), а также прототипирование (макетирование), используемое для оценки качества принимаемых решений.

Методы и техники, нацеленные на выявление нарушений формализованных правил построения исходного кода ПО, проектных моделей и документации.

К методам такого рода относится *инспектирование кода*, заключающееся в целенаправленном поиске определенных дефектов и нарушений требований в коде на основе набора шаблонов, автоматизированные методы поиска ошибок в коде, не основанные на его выполнении, методы проверки документации на согласованность и соответствие стандартам.

Методы и техники обычного или формализованного анализа проектной документации и исходного кода для выявления их свойств.

К этой группе относятся многочисленные методы *анализа архитектуры ПО*, о которых пойдет речь в следующей лекции, методы формального доказательства свойств ПО и формального анализа эффективности применяемых алгоритмов.

Далее мы несколько подробнее рассмотрим тестирование и проверку на моделях как примеры методов контроля качества.

## 12. Методы обеспечения качества аппаратного обеспечения ИС и АС.

**P.S. По-русски ничего нет, это перевод.**

Аппаратное обеспечение также очень важно, как и программное. Отсюда необходимы следующие меры для обеспечения качества аппаратного обеспечения:

**Проверка оборудования**

Для очень важной системы, например, системы электронного голосования, целесообразно использовать независимый центр тестирования для выполнения проверочных тестов системы. Для менее важных систем проверка системы может быть проведена собственными силами.

Проверочные тесты оборудования (также известные как квалификационные тесты) могут включать:

·-тестирование оборудования в условиях, имитирующих ожидаемые условия реальной жизни, включая условия хранения, транспортировки, эксплуатации и обслуживания

·-обеспечение соответствия оборудования местным экологическим требованиям, включая укрытие, пространство, принадлежности, электроснабжение и соответствующие экстремальные температуры, влажность и загрязнение

·-обеспечение адекватности и полноты соответствующей документации

·-проверка того, что оборудование способно работать в ожидаемых нормальных условиях и возможных ненормальных условиях

·-обеспечение наличия соответствующих мер безопасности и их соответствия соответствующим стандартам

·-обеспечение наличия соответствующих мер по обеспечению качества

**Аппаратное тестирование**

Тестирование оборудования обычно более детальное и тщательное, чем проверка. Тестирование необходимо, чтобы убедиться, что каждый компонент системы работает должным образом, и что система работает в точном соответствии с конкретными требованиями.

Комплексная структурированная программа тестирования - программа, обеспечивающая тестирование всех аспектов системы. Это особенно важно для таких ключевых систем, как системы электронного голосования. Меры тестирования, которым можно следовать, включают:

·-разработка набора критериев тестирования

·-применение «нерабочих» тестов, чтобы гарантировать, что оборудование может выдерживать ожидаемые уровни физического обращения, например, испытания на падение при транспортировке

·-проверка, если это необходимо, любого кода, аппаратно встроенного в аппаратное обеспечение (этот код иногда называют прошивкой), чтобы убедиться в его логической корректности и соблюдении соответствующих стандартов

·-применение функциональных тестов для определения того, были ли выполнены критерии тестирования

·-применение качественных оценок для определения соответствия критериям тестирования

·-проведение тестов как в «лабораторных» условиях, так и в различных «реальных» условиях

·-проводить испытания в течение длительного периода времени, чтобы гарантировать стабильную работу систем

·-проведение «нагрузочных тестов», моделирование, насколько это возможно, разнообразных «реальных» условий и использование или превышение объемов данных, которые можно было бы ожидать в реальной ситуации

·-проверка того, что «то, что входит», - это «то, что выходит», путем ввода известных данных и проверки того, что результат согласуется с вводом

**Техническое обслуживание оборудования**

После того, как оборудование было проверено, протестировано и внедрено, его необходимо поддерживать. Процедуры технического обслуживания будут различаться в зависимости от типа и сложности технологии. Многие элементы оборудования поставляются с графиком обслуживания или программой, рекомендованной производителем или поставщиком. Техническое обслуживание также может осуществляться производителем или поставщиком в рамках договора купли-продажи.

Системы должны поддерживаться, чтобы гарантировать, что они продолжают работать на уровне, продемонстрированном на стадии тестирования. Возможно, потребуется разработать планы постоянного мониторинга или тестирования, чтобы гарантировать, что потребности в техническом обслуживании выявляются и удовлетворяются при необходимости. Если системы используются в течение длительного времени, можно использовать механизм для отслеживания обратной связи от пользователей в качестве еще одного средства для определения необходимости обслуживания и модификации.

Если в аппаратное обеспечение вносятся изменения в результате обслуживания или обновлений системы, может возникнуть необходимость инициировать дальнейшие раунды проверки и тестирования системы, чтобы убедиться, что модифицированная система по-прежнему соответствует стандартам.

## 13. Понятие технического задания на разработку ИС и АС.

Техническое задание — исходный документ на проектирование технического объекта (изделия). ТЗ устанавливает основное назначение разрабатываемого объекта, его технические характеристики, показатели качества и технико-экономические требования, предписание по выполнению необходимых стадий создания документации (конструкторской, технологической, программной и т. д.) и её состав, а также специальные требования. Техническое задание является юридическим документом — как приложение включается в договор между заказчиком и исполнителем на проведение проектных работ и является его основой: определяет порядок и условия работ, в том числе цель, задачи, принципы, ожидаемые результаты и сроки выполнения. То есть должны быть объективные критерии, по которым можно определить, сделан ли тот или иной пункт работ или нет. Все изменения, дополнения и уточнения формулировок ТЗ обязательно согласуются с заказчиком и им утверждаются. Это необходимо и потому, что в случае обнаружения в процессе решения проектной задачи неточностей или ошибочности исходных данных возникает необходимость определения степени вины каждой из сторон-участниц разработки, распределения понесенных в связи с этим убытков. Техническое задание, как термин в области информационных технологий – это юридически значимый документ, содержащий исчерпывающую информацию, необходимую для постановки задач исполнителям на разработку, внедрение или интеграцию программного продукта, информационной системы, сайта, портала либо прочего ИТ сервиса.

Техническое задание должно содержать следующие разделы:

· введение;

· основания для разработки;

· назначение разработки;

· требования к программе или программному изделию;

· требования к программной документации;

· технико-экономические показатели;

· стадии и этапы разработки;

· порядок контроля и приемки;

· в техническое задание допускается включать приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

## 14. Базовые структуры и архитектуры программного обеспечения ИС и АС.

Архитектура программного обеспечения (англ. software architecture) — совокупность важнейших решений об организации программной системы. Архитектура включает:

·-выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;

·-соединение выбранных элементов структуры и поведения во всё более крупные системы;

·-архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение[1][2].

Документирование архитектуры программного обеспечения (ПО) упрощает процесс коммуникации между разработчиками, позволяет зафиксировать принятые проектные решения и предоставить информацию о них эксплуатационному персоналу системы[3], повторно использовать компоненты и шаблоны проекта в других.

Архитектура ПО обычно содержит несколько видов, которые аналогичны различным типам чертежей в строительстве зданий. В онтологии, установленной ANSI / IEEE 1471—2000, виды являются экземплярами точки зрения, где точка зрения существует для описания архитектуры с точки зрения заданного множества заинтересованных лиц.

Архитектурный вид состоит из 2 компонентов:

·-Элементы

·-Отношения между элементами

Архитектурные виды можно поделить на 3 основных типа[10]:

1. Модульные виды (англ. module views) — показывают систему как структуру из различных программных блоков.

2. Компоненты-и-коннекторы (англ. component-and-connector views) — показывают систему как структуру из параллельно запущенных элементов (компонентов) и способов их взаимодействия (коннекторов).

3. Размещение (англ. allocation views) — показывает размещение элементов системы во внешних средах.

Примеры модульных видов:

·-Декомпозиция (англ. decomposition view) — состоит из модулей в контексте отношения «является подмодулем»

·-Использование (англ. uses view) — состоит из модулей в контексте отношения «использует» (т.е. один модуль использует сервисы другого модуля)

·-Вид уровней (англ. layered view) — показывает структуру, в которой связанные по функциональности модули объединены в группы (уровни)

·-Вид классов/обобщений (англ. class/generalization view) — состоит из классов, связанные через отношения «наследуется от» и «является экземпляром»

Примеры видов компонентов-и-коннекторов:

·-Процессный вид (англ. process view) — состоит из процессов, соединённых операциями коммуникации, синхронизации и/или исключения

·-Параллельный вид (англ. concurrency view) — состоит из компонентов и коннекторов, где коннекторы представляют собой «логические потоки»

·-Вид обмена данными (англ. shared-data (repository) view) — состоит из компонентов и коннекторов, которые создают, сохраняют и получают постоянные данные

·-Вид клиент-сервер (англ. client-server view) — состоит из взаимодействующих клиентов и серверов, а также коннекторов между ними (например, протоколов и общих сообщений)

Примеры видов размещения:

·-Развертывание (англ. deployment view) — состоит из программных элементов, их размещения на физических носителях и коммуникационных элементов

·-Внедрение (англ. implementation view) — состоит из программных элементов и их соответствия файловым структурам в различных средах (разработческой, интеграционной и т.д.)

·-Распределение работы (англ. work assignment view) — состоит из модулей и описания того, кто ответственен за внедрение каждого из них

Хотя было разработано несколько языков для описания архитектуры программного обеспечения, в настоящий момент нет согласия по поводу того, какой набор видов должен быть принят в качестве эталона. В качестве стандарта «для моделирования программных систем (и не только)» был создан язык UML.

Для удовлетворения проектируемой системы различным атрибутам качества применяются различные архитектурные шаблоны (паттерны). Каждый шаблон имеет свои задачи и свои недостатки.

Примеры архитектурных шаблонов:

·-Многоуровневый шаблон (Layered pattern). Система разбивается на уровни, которые на диаграмме изображаются один над другим. Каждый уровень может вызывать только уровень на 1 ниже него. Таким образом разработку каждого уровня можно вести относительно независимо, что повышает модифицируемость системы. Недостатками данного подхода являются усложнение системы и снижение производительности.

·-Шаблон посредника (Broker pattern). Когда в системе присутствует большое количество модулей, их прямое взаимодействие друг с другом становится слишком сложным. Для решения проблемы вводится посредник (например, шина данных), по которой модули общаются друг с другом. Таким образом, повышается функциональная совместимость модулей системы. Все недостатки вытекают из наличия посредника: он понижает производительность, его недоступность может сделать недоступной всю систему, он может стать объектом атак и узким местом системы.

·-Шаблон «Модель-Представление-Контроллер» (Model-View-Controller pattern). Т.к. требования к интерфейсу меняются чаще всего, то возникает потребность часто его модифицировать, при этом сохраняя корректное взаимодействие с данными (чтение, сохранение). Для этого в шаблоне Model-View-Controller (MVC) интерфейс отделён от данных. Это позволяет менять интерфейсы, равно как и создавать их разные варианты. В MVC система разделена на:

o Модель, хранящую данные

o Представление, отображающее часть данных и взаимодействующее с пользователем

o Контроллер, являющийся посредником между видами и моделью

Однако, концепция MVC имеет и свои недостатки. В частности, из-за усложнения взаимодействия падает скорость работы системы.

·-Клиент-серверный шаблон (Client-Server pattern). Если есть ограниченное число ресурсов, к которым требуется ограниченный правами доступ большого числа потребителей, то удобно реализовать клиент-серверную архитектуру. Такой подход повышает масштабируемость и доступность системы. Но при этом сервер может стать узким местом системы, при его недоступности становится недоступна вся система.

## 15. Основные современные технологии разработки программного обеспечения ИС и АС.

**«Waterfall Model» (каскадная модель или «водопад»)**

****

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 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.

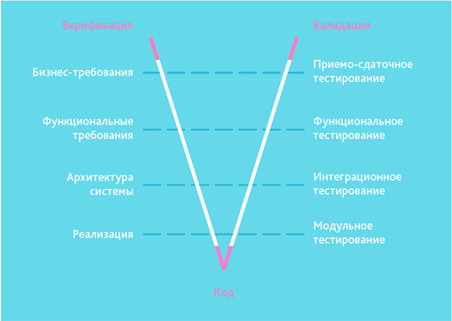
**Когда использовать каскадную методологию?**

· Только тогда, когда требования известны, понятны и зафиксированы. Противоречивых требований не имеется.

·-Нет проблем с доступностью программистов нужной квалификации.

·-В относительно небольших проектах.

**«V-Model»**

****

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, находящегося уже на первоначальных стадиях проектирования. Стадия тестирования проводится одновременно с соответствующей стадией разработки, например, во время кодирования пишутся модульные тесты.

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.

**«Incremental Model» (инкрементная модель)**

****

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.

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.

**«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.

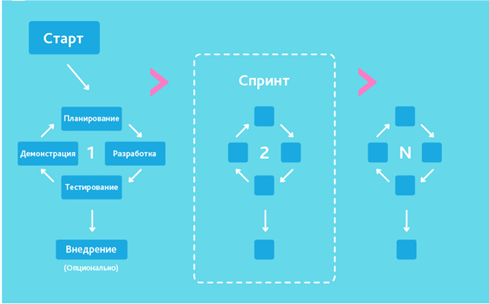
· Сборка приложения: используются средства автоматической сборки для преобразования моделей системы автоматического проектирования в код.

· 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.

**«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.

**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.

**«Iterative Model» (итеративная или итерационная модель)**

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.

**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.

**«Spiral Model» (спиральная модель)**

****

«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.

## 16. Программные интерфейсы информационных и автоматизированных систем.

**Интерфейсы информационных систем** условно можно разделить на 3 группы:

1) текстовые (текст-ориентированные),

2) смешанные (псевдографические),

3) графические.

В качестве примера текстовых (текст-ориентированных) интерфейсов, приведём интерфейс командной строки DOS или shell-интерпретатор UNIX. Пользователь взаимодействует с ВС с помощью клавиатуры, набирая специальные команды. Для задания различных опций служат параметры. Система как ответ на действия пользователя выдает сообщения, или результат выполнения введенной команды в текстовом виде. Курсор может иметь вид мигающего прямоугольника или черточки, обозначающей место ввода. В таком режиме можно одновременно взаимодействовать лишь с одной программой. Управлять взаимодействием программ можно только с командной строки, причём проверить результат - по окончании работы.

В смешанных (псевдографических) интерфейсах различают понятия "оконный" и "графический" интерфейсы. Первый базируется на принципе разделения реального окна монитора (или виртуального desktop'а намного большего размера, чем физический дисплей) на прямоугольные области, внутри каждой из которых определенная программа направляет свой вывод и откуда получает команды.

Термин "графический" означает использование оконного графического интерфейса (каждое окно отображает графический интерфейс); полноэкранного режима (выполняется только одна программа, осуществляющая вывод в графическом режиме). То есть: оконный не обязательно графический, а графический не всегда оконный.

Псевдографическими обозначают интерфейсы, с графическими интерфейсными элементами, например кнопками, индикаторами процесса выполнения, реализуемыми с помощью псевдографики набора ANSI, например, оболочка FAR. Псевдографический интерфейс можно отнести к промежуточному между чисто командным интерфейсом и графическим.

К графическим интерфейсам относят все оконные графические системы Windows, оболочки для UNIX - KDE, GNOM, CDE, X-Window, Photon из ОС QNX, Aqua из MacOS X. Графическими они называются потому, что все элементы пользовательского интерфейса, как и сами данные в окнах, отображаются в графическом режиме, с помощью 256, 16-битной или 32-битной глубины цветового буфера. Это позволяет сформировать привлекательные с точки зрения пользователя окна, кнопки, пиктограммы, ползунки, индикаторы.

Понятие окна - общее для всех этих систем. Окно - прямоугольная область экрана, куда программа выводит свои данные и откуда получает команды.

Интерфейсы **АИС** являются, своего рода, "проекцией" интерфейсов ОС пользователя.

Интерфейс должен обеспечивать:

·-наглядность отображения информации; приближенность к естественному языку, естественным знаковым системам;

·-возможность отображения различной - как фактографической, так и документальной информации (т.е. текста и мультимедиа содержимого);

·-возможность работы с максимально доступным множеством источников данных без потери гибкости, причём с условием регулярности элементов управления; независимость от архитектуры системы и организации сетевых ресурсов.

Пользователи или потребители информации - это животный и растительный мир, люди и технические устройства.

При этом конечный пользователь (англ. "End user") - это пользователь, не работающий непосредственно с системой, но применяющий результат её функционирования.

Интерфейс пользователя - это совокупность правил, методов и программно-аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Пользовательский интерфейс часто понимают только как внешний вид программы. В действительности интерфейс пользователя включает в себя все аспекты, оказывающие влияние на взаимодействие пользователя и системы.

## 17. Аппаратные интерфейсы связи информационных и автоматизированных систем.

**Аппаратный интерфейс** – совокупность алгоритмов обмена и технических средств, обеспечивающих обмен между устройствами. В семиуровневой сетевой модели OSI аппаратный интерфейс соответствует физическому и частично канальному уровню, которые определяют физическую и логическую организацию аппаратного интерфейса.

К основным характеристикам аппаратных интерфейсов относятся:

1. Скорость передачи (пропускная способность, производительность). Производительность оценивается количеством информации (полезной), передаваемой в секунду. Избыточная информация может достигать 90%. Производительность связана с понятием тактовой частоты. Также на неё влияет разрядность шины данных.

2. Протяжённость. Протяжённость связана и влияет на производительность интерфейса, определяется типом сопрягаемых устройств вычислительной системы.

3. Тип сопрягаемых устройств вычислительной системы.

4. Топология. По топологии выделяют: Радиальные интерфейсы. Шинные интерфейсы (моноканал). Цепочечные интерфейсы. Интерфейсы со сложной топологией (каждый с каждым, произвольная топология, гиперкуб и т.д.).

5. Разрядность слова данных (последовательный или параллельный интерфейс).

6. Синхронный или асинхронный интерфейс. Важнейшим моментом в работе аппаратных интерфейсов является синхронизация передачи информации. Синхронизация – это согласование процессов взаимодействия между устройствами, заключающееся в передаче информации источником и ее приема приемником (одним или несколькими). Существуют два основных режима синхронизации: синхронный и асинхронный.

7. Симплексный, полудуплексный, дуплексный обмен.

Примеры аппаратных интерфейсов:

**Шина USB** (Universal Serial Bus — универсальная последовательная шина) обеспечивает подключение к компьютеру большое количество разнообразных периферийных устройств, в том числе мобильные телефоны и бытовую электронику.

**Инфракрасный интерфейс** IrDA позволяет осуществлять беспроводную связь между двумя устройствами на расстоянии до 1 метра. Инфракрасная связь — IR (Infra Red) Connection — безопасна для здоровья, не создает помех в радиочастотном диапазоне и обеспечивает конфиденциальность передачи. ИК-лучи не проходят через стены, поэтому зона приема ограничивается небольшим, легко контролируемым пространством.

**Bluetooth** (синий зуб) — радиоинтерфейс с низким энергопотреблением (мощность передатчика всего порядка 1 мВт) для организации персональных сетей, обеспечивающий передачу данных в режиме реального времени на небольшие расстояния. Каждое устройство Bluetooth имеет радиопередатчик и приемник, работающие в диа¬пазоне частот 2,4 ГГц. Дальность действия радиоинтерфейса составляется около 100 м — для покрытия стандартного дома.

**Ethernet** (ether — эфир) — технология передачи данных, используемая в большинстве локальных компьютерных сетей. Этот интерфейс базируется на стандарте IEE 802.3. Если интерфейс RS-485 можно рассматривать по принципу «один ко многим», то Ethernet работает по принципу «многие ко многим».

## 18. Сетевые интерфейсы ИС и АС. Локальные и глобальные сети.

**Сетевой интерфейс** — физическое или виртуальное устройство, предназначенное для передачи данных между программами через компьютерную сеть.

Примеры сетевых интерфейсов:

·-Физические интерфейсы сетевых карт и телекоммуникационных устройств (коммутаторов, маршрутизаторов и так далее)

·-Петлевые интерфейсы для обмена данными между процессами на одном компьютере или управляемом сетевом устройстве. Для них выделена специальная подсеть 127.0.0.0/8

·-Туннели — для инкапсуляции протокола того же или более низкого уровня в другой протокол

·-Интерфейсы виртуальных сетей (VLAN).

Каждый интерфейс в сети может быть однозначно идентифицирован по его адресу. Разные сетевые протоколы используют разные системы адресации, например MAC-адреса в Ethernet или IP-адреса в IP.

Настройка сетевых интерфейсов в UNIX/Linux-системах традиционно выполняется с помощью команды ifconfig, а в Linux ещё и при помощи команды ip.

**IP-адрес:** Адрес IP, соответствующий данному сетевому интерфейсу. Пакеты, отправленные по этому адресу, поступят на соответствующий интерфейс

**Маска подсети:** Битовая маска, необходимая для вычисления маршрута передачи IP-пакета

**Широковещательный адрес:** Адрес, используемый при широковещательной рассылке пакетов через интерфейс.

**Метрика:** Условная характеристика интерфейса соответствующая уровню затрат при передаче информации через него. Используется при маршрутизации пакетов, для выбора оптимального маршрута.

**MTU:** Maximum Transfer Unit. Максимальный размер блока данных обрабатываемого интерфейсом. Наибольшее значение MTU определяется типом интерфейса (например, для Ethernet MTU=1500), но может быть искусственно снижено.

**MAC-адрес:** Аппаратный адрес сетевого устройства, соответствующего интерфейсу (для которых это имеет смысл).

К локальным сетям - Local Area Networks (LAN) - относят сети компьютеров, сосредоточенные на небольшой территории (обычно в радиусе не более 1-2 км). В общем случае локальная сеть представляет собой коммуникационную систему, принадлежащую одной организации. Из-за коротких расстояний в локальных сетях имеется возможность использования относительно дорогих высококачественных линий связи, которые позволяют, применяя простые методы передачи данных, достигать высоких скоростей обмена данными порядка 100 Мбит/с. В связи с этим услуги, предоставляемые локальными сетями, отличаются широким разнообразием и обычно предусматривают реализацию в режиме on-line.

Глобальные сети - Wide Area Networks (WAN) - объединяют территориально рассредоточенные компьютеры, которые могут находиться в различных городах и странах. Так как прокладка высококачественных линий связи на большие расстояния обходится очень дорого, в глобальных сетях часто используются уже существующие линии связи, изначально предназначенные совсем для других целей. Например, многие глобальные сети строятся на основе телефонных и телеграфных каналов общего назначения. Из-за низких скоростей таких линий связи в глобальных сетях (десятки килобит в секунду) набор предоставляемых услуг обычно ограничивается передачей файлов, преимущественно не в оперативном, а в фоновом режиме, с использованием электронной почты. Для устойчивой передачи дискретных данных по некачественным линиям связи применяются методы и оборудование, существенно отличающиеся от методов и оборудования, характерных для локальных сетей. Как правило, здесь применяются сложные процедуры контроля и восстановления данных, так как наиболее типичный режим передачи данных по территориальному каналу связи связан со значительными искажениями сигналов.

Рассмотрим основные отличия локальных сетей от глобальных более детально. Так как в последнее время эти отличия становятся все менее заметными, то будем считать, что в данном разделе мы рассматриваем сети конца 80-х годов, когда эти отличия проявлялись весьма отчетливо, а современные тенденции сближения технологий локальных и глобальных сетей будут рассмотрены в следующем разделе.

1. Протяженность, качество и способ прокладки линий связи. Класс локальных вычислительных сетей по определению отличается от класса глобальных сетей небольшим расстоянием между узлами сети. Это в принципе делает возможным использование в локальных сетях качественных линий связи: коаксиального кабеля, витой пары, оптоволоконного кабеля, которые не всегда доступны (из-за экономических ограничений) на больших расстояниях, свойственных глобальным сетям, В глобальных сетях часто применяются уже существующие линии связи (телеграфные или телефонные), а в локальных сетях они прокладываются заново.

2. Сложность методов передачи и оборудования. В условиях низкой надежности физических каналов в глобальных сетях требуются более сложные, чем в локальных сетях, методы передачи данных и соответствующее оборудование. Так, в глобальных сетях широко применяются модуляция, асинхронные методы, сложные методы контрольного суммирования, квитирование и повторные передачи искаженных кадров. С другой стороны, качественные линии связи в локальных сетях позволили упростить процедуры передачи данных за счет применения немодулированных сигналов и отказа от обязательного подтверждения получения пакета.

3. Скорость обмена данными. Одним из главных отличий локальных сетей от глобальных является наличие высокоскоростных каналов обмена данными между компьютерами, скорость которых (10,16и100 Мбит/с) сравнима со скоростями работы устройств и узлов компьютера - дисков, внутренних шин обмена данными и т. п. За счет этого у пользователя локальной сети, подключенного к удаленному разделяемому ресурсу (например, диску сервера), складывается впечатление, что он пользуется этим диском, как «своим». Для глобальных сетей типичны гораздо более низкие скорости передачи данных - 2400,9600,28800,33600 бит/с, 56 и 64 Кбит/с и только на магистральных каналах - до 2 Мбит/с.

4. Разнообразие услуг. Локальные сети предоставляют, как правило, широкий набор услуг - это различные виды услуг файловой службы, услуги печати, услуги службы передачи факсимильных сообщений, услуги баз данных, электронная почта и другие, в то время как глобальные сети в основном предоставляют почтовые услуги и иногда файловые услуги с ограниченными возможностями - передачу файлов из публичных архивов удаленных серверов без предварительного просмотра их содержания.

5. Оперативность выполнения запросов. Время прохождения пакета через локальную сеть обычно составляет несколько миллисекунд, время же его передачи через глобальную сеть может достигать нескольких секунд. Низкая скорость передачи данных в глобальных сетях затрудняет реализацию служб для режима on-line, который является обычным для локальных сетей.

6. Разделение каналов. В локальных сетях каналы связи используются, как правило, совместно сразу несколькими узлами сети, а в глобальных сетях - индивидуально.

7. Использование метода коммутации пакетов. Важной особенностью локальных сетей является неравномерное распределение нагрузки. Отношение пиковой нагрузки к средней может составлять 100:1 и даже выше. Такой трафик обычно называют пульсирующим. Из-за этой особенности трафика в локальных сетях для связи узлов применяется метод коммутации пакетов, который для пульсирующего трафика оказывается гораздо более эффективным, чем традиционный для глобальных сетей метод коммутации каналов. Эффективность метода коммутации пакетов состоит в том, что сеть в целом передает в единицу времени больше данных своих абонентов. В глобальных сетях метод коммутации пакетов также используется, но наряду с ним часто применяется и метод коммутации каналов, а также некоммутируемые каналы - как унаследованные технологии некомпьютерных сетей.

8. Масштабируемость. «Классические» локальные сети обладают плохой масштабируемостью из-за жесткости базовых топологий, определяющих способ подключения станций и длину линии. При использовании многих базовых топологий характеристики сети резко ухудшаются при достижении определенного предела по количеству узлов или протяженности линий связи. Глобальным же сетям присуща хорошая масштабируемость, так как они изначально разрабатывались в расчете на работу с произвольными топологиями.

## 19. Применение клиент-серверных технологий для построения ИС и АС.

Как видно из названия, главные «действующие лица»:

·-клиент – компьютерное устройство, которое отсылает запросы серверу, касающиеся выполнения определенных задач или предоставления конкретной информации.

·-сервер – компьютерное устройство, гораздо мощнее обычного ПК.

Система работает по следующему принципу:

1. Клиент отправляет запрос серверной машине.

2. Сервер принимает обращение с требованием выполнить определенное действие и выполняет поставленную задачу.

3. Программно-аппаратный комплекс отправляет клиенту результат выполненной работы, обработанного запроса.

Использование архитектуры клиент-сервер позволило создавать надежные (в смысле целостности данных) многопользовательские ИС и АС с централизованной базой данных, независимые от аппаратной (а часто и программной) части сервера БД и поддерживающие графический интерфейс пользователя (ГИП) на клиентских станциях, связанных локальной сетью. Причем издержки на разработку приложений существенно сокращались

Основные особенности:

·-Клиентская программа работает с данными через запросы к серверному ПО.

·-Базовые функции приложения разделены между клиентом и сервером.

Плюсы:

·-Полная поддержка многопользовательской работы

·-Гарантия целостности данных

Минусы:

·-Бизнес логика приложений осталась в клиентском ПО.

·-При любом изменении алгоритмов, надо обновлять пользовательское ПО на каждом клиенте.

·-Высокие требования к пропускной способности коммуникационных каналов с сервером, что препятствует использование клиентских станций иначе как в локальной сети.

·-Слабая защита данных от взлома, в особенности от недобросовестных пользователей системы.

·-Высокая сложность администрирования и настройки рабочих мест пользователей системы.

·-Необходимость использовать мощные ПК на клиентских местах.

·-Высокая сложность разработки системы из-за необходимости выполнять бизнеслогику и обеспечивать пользовательский интерфейс в одной программе.

Нетрудно заметить, что большинство недостатков классической или 2-х слойной архитектуры клиент-сервер проистекают от использования клиентской станции в качестве исполнителя бизнес-логики ИС. Поэтому очевидным шагом дальнейшей эволюции архитектур ИС явилась идея "тонкого клиента", то есть разбиения алгоритмов обработки данных на части связанные с выполнением бизнес-функций и связанные с отображением информации в удобном для человека представлении. При этом на клиентской машине оставляют лишь вторую часть, связанную с первичной проверкой и отображением информации, перенося всю реальную функциональность системы на серверную часть.

## 20. Понятие программного интерфейса (API).

**API Application Programming Interface** (программный интерфейс приложения) — это интерфейс, позволяющий двум независимым компонентам программного обеспечения обмениваться информацией. API играет роль посредника между внутренними и внешними программными функциями, обеспечивая настолько эффективный обмен информацией, что конечные пользователи обычно его просто не замечают.

Говоря по-простому, API действует как виртуальный посредник и передает информацию из одного интерфейса, например мобильного приложения, в другой. API связывает различные части программной платформы, чтобы передаваемая информация дошла до места назначения.

Эти связующие узлы не только выполняют роль внутренних каналов связи, но и позволяют внешним инструментам получать доступ к этой же информации. Таким образом API-интерфейсы могут относиться к одной из двух категорий:

·-Внутренние/частные API

·-Внешние/открытые API

Частные API доступны только разработчикам и пользователям из числа сотрудников организации. Такие API обычно связывают внутренние процессы для уменьшения разрозненности рабочих данных и оптимизации совместной работы.

Открытые API, в свою очередь, позволяют внешним разработчикам получать доступ к информации и интегрировать информацию, которая передается из одного программного инструмента в другой. Открытые или частные API экономят время разработчиков, позволяя им объединять платформы с имеющимися инструментами и устраняя необходимость в создании нового функционала с нуля.

На практике API могут использоваться для связи практически любых процессов. Вот несколько распространенных примеров использования API:

·-Обмен информацией о рейсах между авиакомпаниями и туристическими сайтами

·-Использование Google Maps в приложении для совместных поездок (райдшеринга)

·-Создание виртуальных собеседников в службе обмена сообщениями

·-Встраивание видеоклипов с YouTube на веб-странице

·-Автоматизация рабочих процессов в программных инструментах для B2B-сектора.

## 21. Облачные системы для комплексирования ИС и АС. (как найду что добавить доделаю)

При разработке новой сложной системы испытания на уровне системы невозможно начать, пока система полностью не собрана и не продемонстрирована ее работоспособность. Успех и быстрота комплексирования сложной системы зависят от того, насколько удачно она была разбита на подсистемы, взаимодействия между которыми просты, а подсистемы, в свою очередь, - на хорошо определенные компоненты. Обычно процесс комплексирования включает две стадии: 1) сборка отдельных подсистем из соответствующих компонентов; 2) монтаж подсистем и их агрегирование в систему в целом.

## 22. Методы удаленного доступа к ресурсам ИС и АС.

Существует три метода подключения удаленного пользователя или отделения фирмы к локальной вычислительной сети (ЛВС) компании

***Эмуляция терминала*** — это метод, при котором пользователь удаленного терминала с помощью специального программного обеспечения подключается по глобальной сети к другому компьютеру, как локальный узел. Этот способ часто используется на мейнфреймах и мини-компьютерах, но мало распространен в ЛВС.

***Удаленное управление (remote control)*** - это метод, который позволяет удаленному пользователю получить контроль над локальными ПК в ЛВС корпорации (т. е. управлять одним из ПК в ЛВС). Скорость проведения сеанса и его возможности зависят от характеристик управляемого ПК, т. к. именно на нем выполняется обработка всех сетевых команд. Коды клавиш, нажимаемых на удаленном ПК, посылаются в управляемый ПК, а все изменения на экране управляемого выводятся на экран удаленного ПК

Сервер удаленного доступа может быть реализован: -в виде модема со встроенным специальным ПО; -либо быть сервером ЛВС, на котором выполняются программы удаленного узла.

ИЛИ

Удаленный доступ часто является единственно возможным условием организации постоянного онлайн-мониторинга разветвленных технологических систем. Без предоставления удаленного доступа существенно возрастает стоимость проведения работ по техническому обслуживанию, оперативной настройке и элементарной перезагрузке оборудования. В некоторых экстренных случаях удаленный доступ просто необходим для подключения специалистов производителя и обслуживающей организации.

**VPN: программный уровень**

VPN может быть организован путем установки специального программного обеспечения. Наиболее распространенная программа – OpenVPN, либо можно воспользоваться встроенными средствами операционной системы. Очевидный недостаток такого подхода в том, что вся вычислительная нагрузка на шифрование, туннелирование и прочее ложится на вашу технологическую сеть.

**VPN: аппаратный уровень**

Организация VPN на аппаратном уровне требует, исходя из названия, покупки/наладки/обслуживания определенного оборудования. Это дорогостоящий инструмент, отличающийся практически неограниченной безопасностью, эффективностью и масштабируемостью. В данном случае нагрузка на создание VPN ложится на специальные межсетевые экраны, поддерживающие протоколы IPSEC, которые выпускаются такими компаниями, как Cisco, Palo Alto, Juniper и др. Экраны ставятся на вход в сеть. На клиентской стороне устанавливается специальное программное обеспечение, поставляемое совместно с оборудованием, оно может скачиваться при обращении к устройству.

**Создание облачного сервера сети**

Пожалуй, самым высокотехнологичным и радикальным инструментом организации доступа к технологической сети является создание облачного сервера сети. Все подключения к нему будут априори носить удаленный характер. Не потребуется никакого дополнительного программного обеспечения и серьезной вычислительной мощности на клиентской стороне. К серверу можно будет безопасно подключиться из любой точки мира, база данных и конфигурация станут постоянно резервироваться, а на страже контура будут самые современные технологии информационной безопасности.

**Обычный 3G-модем**

На практике распространенным среди инженеров на объектах способом является предоставление удаленного доступа к оборудованию сверхзащищенной сети через обычный 3G-модем, несанкционированно установленный на сервере.

**Таким образом** наиболее практичным инструментом для организации удаленного доступа является VPN. В зависимости от имеющегося бюджета можно организовать VPN-сеть с защитой как на программном, так и на аппаратном уровне, причем последняя дороже, но эффективнее. Прямой доступ к отдельным элементам технологической сети также лучше организовать через VPN.

## 23. Основные принципы разработки прикладного программного и аппаратного обеспечения ИС и АС.

Выбор принципов определяется целями, стоящими перед качеством прикладного обеспечения.

*Принцип разделения задач* позволил работать с различными аспектами одной и той же задачи, концентрируясь на них так, как будто это разные задачи. Разделение задач – это применение здравого смысла, которому стараются следовать в повседневной жизни для преодоления встречающихся трудностей. Этот же принцип применяется в разработке программного обеспечения для преодоления свойственной ему сложности.

Прежде всего, задачи следует разнести во времени. Данный принцип лежит в основе итеративной модели жизненного цикла программного обеспечения. Данная модель определяет последовательность действий, которую необходимо соблюдать в ходе разработки программного обеспечения. Это, как уже было упомянуто ранее, анализ и спецификация требований, проектирование, реализация и тестирование, и, наконец, поставка и сопровождение программного обеспечения. Далее, задачи, которые менее всего связаны друг с другом, изолируются, а затем реализуются раздельно с учётом только имеющих отношение друг к другу связей.

*Принцип модульности*. Главное преимущество модульности состоит в возможности применить принцип разделения задач на двух этапах: при работе с элементами каждого модуля отдельно (игнорируя элементы других модулей) и при работе с общими характеристиками всех модулей и отношениями между ними.

Принцип модульности применяется как при проектировании, так и при разработке программного обеспечения. Использование принципа модульности даст следующие преимущества:

- разделение системы на простые части;

- сборка системы из готовых модулей;

- интерпретация системы как состоящую из составных частей;

- модификация системы путём модификации лишь небольшого количества её частей;

- проведение автоматического модульного тестирования.

*Принцип предусмотрения изменений.* В ходе разработки программный продукт постоянно претерпевает изменения, которые обусловлены не только необходимостью устранения ошибок, не замеченных в процессе тестирования, но и необходимостью поддерживающего развития приложения по мере появления новых требований инженерно-технологического персонала или изменения старых. Для обеспечения способности программного обеспечения к развитию требуется предусмотреть, где и когда вероятна необходимость изменений. Возможные изменения изолируются в особых частях программного обеспечения таким образом, чтобы эти изменения затрагивали только эти небольшие части. Если программное приложение разрабатывается в условиях, когда изначально требования к нему сформулированы не полностью, то рассматриваемый принцип особенно важен.

*Принцип общности* состоит в решении, по возможности, более общей проблемы, которая может быть скрыта за рассматриваемой задачей. Вполне возможно, что обобщённая проблема окажется не сложнее – на самом деле, даже проще, чем исходная. Более того, весьма вероятно, что выше окажется и потенциал многократного использования алгоритма решения обобщенной задачи или такое решение уже реализовано в готовых программных продуктах.

*Принцип инкрементности.* Инкрементность характеризует процесс, который происходит пошаговым образом, приращениями. Требуемую цель пытаются достичь последовательно приближающимися аппроксимациями к этой цели. Каждая следующая аппроксимация есть приращение предыдущей. Инкрементность используется во многих видах инженерной деятельности. В применении к программному обеспечению инкрементность означает, что требуемое приложение производится в результате процесса его постепенного развития.

Обычно применение инкрементности состоит в определении основных функций приложения, которые должны быть реализованы в первую очередь и представлены конечным пользователям с целью получения их раннего отклика. Это позволит управлять развитием приложения в условиях, когда начальные требования к нему неустойчивы и не до конца понятны. Обоснованием данного подхода является то, что невозможно выяснить все требования к приложению до тех пор, пока не разработан его прототип, который доступен для практических экспериментов. Таким образом, после реализации базового набора функций, к приложению постепенно добавлялись новые возможности.

## 24. Современные аппаратные платформы ИС и АС.

Аппаратная платформа — уровень, образованный микроархитектурой, микропрограммой управления ядром микропроцессора и архитектурой набора команд на аппаратной базе конкретных микросхем процессора, чипсета, других физических компонентов, которые в совокупности составляют аппаратную модель вычислительной системы.

Предназначен для запуска определенных семейств программных продуктов (операционная система, прикладное программное обеспечение), которые, в свою очередь, разработаны исходя из возможностей и для запуска на данной аппаратуре. Конкретно, аппаратные платформы отличаются друг от друга совокупностью аппаратуры (процессором, чипсетом), а также разработанными (и запускаемыми) программными компонентами.

Одной из наиболее распространённых офисных платформ и персональных компьютеров является IBM PC. На рынке персональных компьютеров также распространены компьютеры Apple. Эти платформы являются широко известными брендами.

## 25. Понятие встраиваемых аппаратных платформ для ИС и АС.

**Встра́иваемая систе́ма** (встро́енная систе́ма, англ. embedded system) — специализированная микропроцессорная система управления, контроля и мониторинга, концепция разработки которой заключается в том, что такая система будет работать, будучи встроенной непосредственно в устройство, которым она управляет.

В связи с тем, что система будет размещаться внутри более сложного устройства, при её разработке ключевую роль играют следующие факторы:

1. минимальное собственное энергопотребление (возможно, автономное питание);
2. минимальные собственные габариты и вес;
3. собственная защита (корпус) минимальна и обеспечивается прочностью и жёсткостью конструкции и применёнными элементами;
4. функции отвода тепла (охлаждения) обеспечивают минимум требований тепловых режимов.
5. микропроцессор и системная логика, а также ключевые микросхемы по возможности совмещены на одном кристалле;

## 26. Инструментальные средства разработки и отладки программного обеспечения ИС и АС.

Инструментальные средства разработки ПО– это совокупность аппаратно-программных средств, позволяющих осуществить написание и отладку программ для микропроцессорных систем с большой степенью достоверности их работоспособности в реальных системах.

К ним относятся:

- внутрисхемные эмуляторы (ВСЭ) - программно аппаратное средство, способное замещать собой эмулируемый процессор в реальном устройстве;

- программные симуляторы - программное средство способное имитировать работу микроконтроллера и его памяти;

- мониторы отладки - специальная программа, загружаемая в память отлаживаемой системы и вынуждающая процессор выполнять, кроме прикладной задачи, еще и отладочные функции.

- платы развития (Evaluation Boards - оценочные платы) - своеобразные конструкторы для макетирования прикладных систем. Обычно на печатной плате устанавливается вся необходимая аппаратура для демонстрации возможностей микроконтроллера, обеспечивается связь с компьютером и предоставляется поле для монтажа прикладных схем пользователя;

- эмуляторы ПЗУ - программно-аппаратное средство, позволяющее заменить ПЗУ отлаживаемого устройства на ОЗУ, в которое можно загрузить программу с компьютера через один из стандартных каналов связи.

- интегрированные среды разработки, содержащие в своем составе текстовые редакторы, компиляторы, редакторы связей, загрузчики и симуляторы.

Отладкой называют процесс устранения в программе ошибок, которые были найдены на этапе тестирования. Первая задача, решаемая при отладке, - это локализация ошибки, т. е. выявление места в программе, где она была допущена. Вторая задача - это исправление локализованной ошибки.

Отладочные функции:

- загрузка прикладных кодов в свободную от монитора память;

- установка точек останова;

- запуск и останов загруженной программы в реальном времени;

- проход программы пользователя по шагам;

- просмотр, редактирование содержимого памяти и управляющих регистров.

## 27. Инструментальные средства разработки и отладки аппаратного обеспечения ИС и АС.

Современные программно-инструментальные средства разработки и отладки (средства разработки) позволяют провести весь этап разработки электронного устройства- от выработки схемотехнического решения до выпуска полного набора конструкторской документации и моделирования физических процессов, происходящих в устройстве.

Разработка аппаратных средств включает в себя разработку общей принципиальной схемы, разводку топологии плат, монтаж макета и его автономную отладку. Время выполнения этих этапов зависит от имеющегося набора апробированных функционально-топологических модулей, опыта и квалификации разработчика.

Во многих случаях можно обеспечить значительное сокращение сроков и повышение качества разработки АС путем использования «полуфабрикатов» или готовых изделий, выпускаемых рядом производителей. Этот класс средств разработки получил название плат развития. Условно их можно разделить на следующие типы:

1.системные комплекты (evaluation kit) – наборы размещенных на плате аппаратных средств, достато-х для реализации несложных устро-тв;

2.отладочные платы и сист-ы– размещенные на плате программно-аппаратные комплексы, обеспечивающие моделирование и отладку устройств различного назначения на базе определенных моделей микроконтроллеров;

3.целевые платы – программно-аппаратные комплексы, ориентированные на использование после отладки в качестве прототипной МКС;

4.одноплатные компьютеры и контроллеры – конструктивные комплексы, предназначенные для использования в качестве базо-х модулей при реализации целевых МКС промы-го примене-я. Эти средства могут использо-ся для следующих целей:

1) изучение функционирования определенных моделей микроконтроллеров, получение навыков их практического прим-я;

2) тестирование и отладка программного обеспечения устройств на основе реальных образцов микроконтроллеров;

3) комплексная отладка макета МКС, используемого затем в качестве образца для реализации прототипной системы;

4) сборка и отладка прототипной или целевой МКС, в состав которого входят платы развития в качестве базовых модулей.

Отладка аппаратуры на основе микроконтроллеров с открытой архитектурой предполагает контроль состояния многоразрядных магистралей адреса и данных с целью проверки правильности обращения к внешним ресурсам памяти и периферийным устройствам. Закрытая архитектура микроконтроллера предполагает реализацию большинства функций разрабатываемого устройства внутренними средствами микроконтроллера. Поэтому разрабатываемый контроллер будет иметь малое число периферийных ИС, а обмен с ними будет идти преимущественно по последовательным интерфейсам. Здесь на первый план выйдут вопросы согласования по нагрузочной способности параллельных портов МК и отладка алгоритмов обмена по последовательным каналам.Помимо этого, можно рассказать часть 26 вопроса.

## 28. Интеллектуальные технологии разработки программного обеспечения ИС и АС.

пишем про CASE, смотреть вопрос 30

матлаб

симулинк

моделсим для симулиса

сим ин тех sim in tech

**MATLAB** (сокращение от англ. *«Matrix Laboratory»*, в русском языке произносится как **Матла́б**) — пакет прикладных программ для решения задач технических вычислений. Пакет используют более миллиона инженерных и научных работников, он работает на большинстве современных операционных систем, включая Linux, Mac OS, Solaris (начиная с версии R2010b поддержка Solaris прекращена) и Windows.

### **Математика и вычисления**[

MATLAB предоставляет пользователю большое количество (несколько сотен) функций для анализа данных, покрывающие практически все области математики, в частности:

* Матрицы и линейная алгебра — алгебра матриц, линейные уравнения, собственные значения и векторы, сингулярности, факторизация матриц и другие.
* Многочлены и интерполяция — корни многочленов, операции над многочленами и их дифференцирование, интерполяция и экстраполяция кривых и другие.
* Математическая статистика и анализ данных — статистические функции, статистическая регрессия, цифровая фильтрация, быстрое преобразование Фурье и другие.
* Обработка данных — набор специальных функций, включая построение графиков, оптимизацию, поиск нулей, численное интегрирование (в квадратурах) и другие.
* Дифференциальные уравнения — решение дифференциальных и дифференциально-алгебраических уравнений, дифференциальных уравнений с запаздыванием, уравнений с ограничениями, уравнений в частных производных и другие.
* Разреженные матрицы — специальный класс данных пакета MATLAB, использующийся в специализированных приложениях (функция появилась в 1992 году[12]).
* Целочисленная арифметика — выполнение операций целочисленной арифметики в среде MATLAB.

### **Разработка алгоритмов**[править | править код]

MATLAB предоставляет удобные средства для разработки алгоритмов, включая высокоуровневые с использованием концепций объектно-ориентированного программирования. В нём имеются все необходимые средства интегрированной среды разработки, включая отладчик и профайлер. Функции для работы с целыми типами данных облегчают создание алгоритмов для микроконтроллеров и других приложений, где это необходимо.

### **Визуализация данных**[править | править код]

В составе пакета MATLAB имеется большое количество функций для построения графиков, в том числе трёхмерных, визуального анализа данных и создания анимированных роликов.

Встроенная среда разработки позволяет создавать графические интерфейсы пользователя с различными элементами управления, такими как кнопки, поля ввода и другими.

### **Независимые приложения**[править | править код]

Программы MATLAB, как консольные, так и с графическим интерфейсом пользователя, могут быть собраны с помощью модуля *MATLAB Compiler* в независимые от MATLAB исполняемые приложения или динамические библиотеки, для запуска которых на других компьютерах, однако, требуется установка свободно распространяемой среды MATLAB Runtime[13] (ранее называлась *MATLAB Compiler Runtime* MCR)[14].

### **Внешние интерфейсы**[править | править код]

Пакет MATLAB включает различные интерфейсы для получения доступа к внешним подпрограммам, написанным на других языках программирования, данным, клиентам и серверам, общающимся через технологии Component Object Model или Dynamic Data Exchange, а также периферийным устройствам, которые взаимодействуют напрямую с MATLAB. Многие из этих возможностей известны под названием MATLAB API.

#### **COM**[править | править код]

Пакет MATLAB предоставляет доступ к функциям, позволяющим создавать, манипулировать и удалять COM-объекты (как клиенты, так и серверы). Поддерживается также технология ActiveX. Все COM-объекты принадлежат к специальному COM-классу пакета MATLAB. Все программы, имеющие функции контроллера автоматизации (англ. *Automation controller*) могут иметь доступ к MATLAB как к серверу автоматизации (англ. *Automation server*).

#### **.NET**[править | править код]

Пакет MATLAB в Microsoft Windows предоставляет доступ к программной платформе .NET Framework. Имеется возможность загружать .NET сборки (Assemblies) и работать с объектами .NET классов из среды MATLAB. В версии MATLAB 7.11 (R2010b) поддерживается .NET Framework версий 2.0, 3.0, 3.5 и 4.0.

#### **DDE**[править | править код]

Пакет MATLAB содержит функции, которые позволяют ему получать доступ к другим приложениям среды Windows, равно как и этим приложениям получать доступ к данным MATLAB, посредством технологии динамического обмена данными (DDE). Каждое приложение, которое может быть DDE-сервером, имеет своё уникальное идентификационное имя. Для MATLAB это имя — *Matlab*.

#### **Веб-сервисы**[править | править код]

В MATLAB существует возможность использовать веб-сервисы. Специальная функция создаёт класс, содержащий методы API веб-сервиса, что позволяет обращаться к веб сервису, через вызовы методов класса.

MATLAB взаимодействует с клиентом веб-сервиса с помощью получения от него данных, их обработки и отправки результата. Поддерживаются следующие технологии: Simple Object Access Protocol (SOAP) и Web Services Description Language (WSDL).

#### **Последовательный порт**[править | править код]

Интерфейс для последовательного порта пакета MATLAB обеспечивает прямой доступ к периферийным устройствам, таким как модемы, принтеры и научное оборудование, подключающееся к компьютеру через последовательный порт (COM-порт). Интерфейс работает путём создания объекта специального класса для последовательного порта. Имеющиеся методы этого класса позволяют считывать и записывать данные в последовательный порт, использовать события и обработчики событий, а также записывать информацию на диск компьютера в режиме реального времени. Это бывает необходимо при проведении экспериментов, симуляции систем реального времени и для других приложений.

#### **MEX-файлы**[править | править код]

Пакет MATLAB включает интерфейс взаимодействия с внешними приложениями, написанными на языках C и Фортран. Осуществляется это взаимодействие через MEX-файлы. Существует возможность вызова подпрограмм, написанных на C или Фортране из MATLAB, как будто это встроенные функции пакета. MEX-файлы представляют собой динамически подключаемые библиотеки, которые могут быть загружены и исполнены интерпретатором, встроенным в MATLAB. MEX-процедуры имеют также возможность вызывать встроенные команды MATLAB.

#### **DLL**[править | править код]

Интерфейс MATLAB, относящийся к общим DLL, позволяет вызывать функции, находящиеся в обычных динамически подключаемых библиотеках, прямо из MATLAB. Эти функции должны иметь C-интерфейс.

Кроме того, в MATLAB имеется возможность получить доступ к его встроенным функциям через C-интерфейс, что позволяет использовать функции пакета во внешних приложениях, написанных на C. Эта технология в MATLAB называется *C Engine*.

Simulink – среда динамического междисциплинарного моделирования сложных технических систем и основной инструмент для модельно-ориентированного проектирования. Его основным интерфейсом является графический инструмент для построения диаграмм и настраиваемый набор библиотек блоков. Он предлагает тесную интеграцию с остальной средой MATLAB и может либо использовать MATLAB, либо создавать сценарии из него. Simulink широко используется в автоматическом управлении и цифровой обработке сигналов для многодоменного моделирования и проектирования на основе моделей.

В сочетании с другими своими продуктами Simulink может автоматически генерировать исходный код на языке C для реализации систем в режиме реального времени. Поскольку эффективность и гибкость кода улучшаются, он становится все более широко распространенным для производственных систем, в дополнение к тому, что является инструментом для разработки встроенных систем из-за его гибкости и способности к быстрой итерации, Встроенный кодер создает код, достаточно эффективный для использования во встроенных системах.

# IntelliCode

Расширение IntelliCode для Visual Studio Code предлагает основанные на искусственном интеллекте возможности IntelliSense для Python, Java, TypeScript и JavaScript. Технология IntelliSense на основе искусственного интеллекта прогнозирует API, который, скорее всего, понадобится разработчику, вместо того чтобы просто предоставлять их список в алфавитном порядке. Он формирует список динамически на основе текущего контекста кода конкретного разработчика.

SimInTech – российская система модельно-ориентированного проектирования систем автоматического управления (САУ).

Программное обеспечение SimInTech состоит из графической среды разработки и исполнительной системы реального времени NordWind.

SimInTech – среда создания математических моделей, алгоритмов управления, интерфейсов управления и автоматической генерации кода для программируемых контроллеров и графических дисплеев.

NordWind – исполнительная система реального времени, которая позволяет запускать сформированные алгоритмы на контроллере.

SimInTech предназначен для детального исследования и анализа нестационарных процессов в различных объектах управления. Разработка математических моделей и алгоритмов управления в SimInTech происходит в виде структурного проектирования логико-динамических систем, описываемых во входо-выходных отношениях, в виде систем обыкновенных дифференциальных уравнений и/или дифференциально-алгебраических уравнений.

SimInTech решает задачи моделирования объектов ВПК/ОПК, энергетических объектов, атомных объектов, нефтегазовых объектов, аэрокосмических объектов, транспортных объектов и других. Кроме этого, SimInTech предназначен для создания тренажёров.

## 29. Интеллектуальные технологии разработки аппаратного обеспечения ИС и АС.

пишем про CASE, смотреть вопрос 30

MATLAB SIMULINK, MODELSIM, SIMINTECH (ПРО ЭТО)

ModelSim - это многоязычная среда от Mentor Graphics [1] для моделирования языков описания оборудования, таких как VHDL, Verilog и SystemC, и включает встроенный отладчик C. [2] [1] ModelSim можно использовать независимо или вместе с Intel Quartus Prime, Xilinx ISE или Xilinx Vivado. [3] Моделирование выполняется с использованием графического пользовательского интерфейса (GUI) или автоматически с использованием сценариев. [4]

Продукты Mentor HDL для моделирования предлагаются в нескольких редакциях, таких как ModelSim PE и Questa Sim.

Questa Sim предлагает высокопроизводительные и расширенные возможности отладки, в то время как ModelSim PE - это симулятор начального уровня для любителей и студентов. [1] Questa Sim используется в больших многомиллионных проектах ворот и поддерживается в Microsoft Windows и Linux, в 32-битных и 64-битных архитектурах. [1]

ModelSim также можно использовать с MATLAB / Simulink, используя Link для ModelSim. [5] [6] Ссылка для ModelSim - это быстрый двунаправленный интерфейс совместного моделирования между Simulink и ModelSim. [6] [5] Для таких проектов MATLAB предоставляет набор инструментов для численного моделирования, а ModelSim предоставляет инструменты для проверки аппаратной реализации и временных характеристик проекта [6].

**SimInTech** (Simulation In Technic) — среда динамического моделирования технических систем, предназначенная для расчётной проверки работы систем управления сложными техническими объектами. SimInTech осуществляет моделирование технологических процессов, протекающих в различных объектах, с одновременным моделированием системы управления, и позволяет повысить качество проектирования систем управления за счет проверки принимаемых решений на любой стадии проекта.

SimInTech предназначен для детального исследования и анализа нестационарных процессов в ядерных и тепловых энергоустановках, в системах автоматического управления, в следящих приводах и роботах, и в любых технических системах, описание динамики которых может быть представлено в виде системы дифференциально-алгебраических уравнений и/или реализовано методами структурного моделирования. Основными направлениями использования SimInTech являются создание моделей, проектирование алгоритмов управления, их отладка на модели объекта, генерация исходного кода на языке Си для программируемых контроллеров.

Для SimInTech созданы и разрабатываются модули расширения позволяющие создавать модели на базе специализированных расчетных кодов и интегрировать их в комплексные модели и проекты.

Может:

* использоваться для моделирования нестационарных процессов в физике, в электротехнике, в динамике машин и механизмов, в астрономии и т. д., а также для решения нестационарных краевых задач (теплопроводность, гидродинамика и др.);
* функционировать в многокомпьютерных моделирующих комплексах, в том числе и в системах удаленного доступа к технологическим и информационным ресурсам;
* функционировать как САПР при групповой разработке и сопровождении жизненного цикла изделия (проекта) при модельно-ориентированном подходе к проектированию.

SimInTech содержит библиотеки типовых блоков для моделирования:

* теплогидравлики/пневматики;
* электроцепей, в действующих и мгновенных значениях;
* силовых машин гидравлических/пневматических;
* механических взаимодействий;
* точечной кинетики нейтронов;
* баллистики космических аппаратов;
* динамики полета летательных аппаратов в атмосфере;
* электрических приводов;

Для разработки алгоритмов управления в SimInTech есть общетехнические библиотеки блоков автоматики. Среди них библиотеки:

* конечных автоматов;
* релейной автоматики;
* нечеткой логики.

Кроме этого, SimInTech обладает:

* инструментами для создания интерфейсов управления (SCADA);
* библиотекой цифровой обработки сигналов;
* библиотекой статистики;
* функционалом оптимизации/подбора параметров;
* протоколами обмена (OPC, UDP, TCP/IP, MODBUS, RS, FMI и т.д.);
* функционалом распараллеливания расчетов на разных вычислительных узлах;
* модулем для верификации кода ПЛИС;
* модулем анализа надежности, безопасности и живучести системы на принципиальной схеме;
* библиотекой нейронных сетей;
* библиоткой видеообработки.

Автоматический генератор кода SimInTech для контроллеров позволяет сгенерировать код на языках:

* ANSI C;
* ST.

В SimInTech есть шаблоны, позволяющие произвести сборку исполняемого кода для различных целевых систем:

* QNX/КПДА;
* Linux;
* Schneider Electric;
* B&R;
* любых микроконтроллеров и ДСП-процессоров, в том числе фирмы «Миландр».

Программа **Simulink** является приложением к пакету **MATLAB**. При моделировании с использованием **Simulink** реализуется принцип визуального программирования, в соответствии с которым, пользователь на экране из библиотеки стандартных блоков создает модель устройства и осуществляет расчеты. При этом, в отличие от классических способов моделирования, пользователю не нужно досконально изучать язык программирования и численные методы математики, а достаточно общих знаний требующихся при работе на компьютере и, естественно, знаний той предметной области в которой он работает.

**Simulink** является достаточно самостоятельным инструментом **MATLAB** и при работе с ним совсем не требуется знать сам **MATLAB** и остальные его приложения. С другой стороны доступ к функциям **MATLAB** и другим его инструментам остается открытым и их можно использовать в **Simulink.** Часть входящих в состав пакетов имеет инструменты, встраиваемые в **Simulink** (например, **LTI-Viewer** приложения **Control System Toolbox –** пакета для разработки систем управления). Имеются также дополнительные библиотеки блоков для разных областей применения (например, **Power System Blockset** – моделирование электротехнических устройств, **Digital Signal Processing Blockset** – набор блоков для разработки цифровых устройств и т.д).

При работе с **Simulink** пользователь имеет возможность модернизировать библиотечные блоки, создавать свои собственные, а также составлять новые библиотеки блоков.

При моделировании пользователь может выбирать метод решения дифференциальных уравнений, а также способ изменения модельного времени (с фиксированным или переменным шагом). В ходе моделирования имеется возможность следить за процессами, происходящими в системе. Для этого используются специальные устройства наблюдения, входящие в состав библиотеки **Simulink**. Результаты моделирования могут быть представлены в виде графиков или таблиц.

Преимущество **Simulink** заключается также в том, что он позволяет пополнять библиотеки блоков с помощью подпрограмм написанных как на языке **MATLAB,** так и на языках **С++,** **Fortran** и **Ada**.

## 30. CASE-методы разработки программного и аппаратного обеспечения ИС и АС.

В последнее время сформировалось новое направление - CASE-технологии (Computer-Aided Software/System Engineering – компьютерная поддержка проектирования). Современные CASE средства обеспечивают поддержку различных методологий разработки программного обеспечения. Наиболее трудоемкими являются этапы анализа и проектирования программного обеспечения, наиболее простым – этап кодировки.

CASE-средства содержат следующими компоненты:

* репозитарий для хранения компонентов проекта программного обеспечения;
* графические средства анализа и проектирования, обеспечивающие создание различного вида моделей АИС;
* средства разработки приложений (генераторы кодов программ);
* средства конфигурационного управления;
* средства документирования;
* средства тестирования;
* средства управления проектом;
* средства реинжиниринга.

CASE-средства различаются по функциональной ориентации на определенные стадии жизненного цикла программного обеспечения, по уровню интеграции выполняемых функций, по методологиям разработки программ, доступным аппаратным и программным платформам и др. Наиболее часто применяется следующая классификация CASE-средств:

1. +Средства анализа (Upper CASE) – построение и анализ предметной области (BPwin, Design IDEF Meta Soft)
2. Средства анализа и проектирования (Middle CASE) – создание проектных спецификаций для различных методологий проектирования (SilverRun, Team Builder, PRO-IV и др.)
3. Средства проектирования БД (Erwin, S-Designer и др.)
4. Средства разработки приложений (4GL, PowerBuilder, Delphi и дрю)
5. Средства реинжиниринга – обеспечение анализа программных кодов и схемы БД, формирование на их основе различных проектов и программных спецификаций.

## 31. Классификация CASE-средств разработки программного и аппаратного обеспечения ИС и АС.

Из всего многообразия CASE средств, существующих на сегодняшний день, можно выделить три основные группы. Эти группы связаны с этапами разработки информационных систем и их жизненным циклом. Классификация CASE средств осуществляется в зависимости от того, какие из этапов разработки они поддерживают.

Выделяют следующие группы CASE средств:

CASE средства верхнего уровня. Эти CASE средства ориентированы на начальные этапы построения информационной системы. Они связаны с анализом и планированием. CASE средства верхнего уровня обеспечивают стратегическое планирование, расстановку целей, задач и приоритетов, а также графическое представление необходимой информации. Все CASE средства верхнего уровня содержат графические инструменты построения диаграмм, таких как диаграммы сущность-связь (ER диаграммы), диаграммы потока данных (DFD), структурные схемы, деревья решений и пр.

CASE средства нижнего уровня. Эти CASE средства больше сфокусированы на последних этапах разработки информационной системы – проектирование, разработка программного кода, тестирование и внедрение. CASE средства нижнего уровня зависят от данных, которые предоставляют средства верхнего уровня. Они используются разработчиками приложений и помогают создать информационную систему, однако не являются полноценными инструментами разработки программного обеспечения.

Интегрированные CASE средства (I – CASE). Эти CASE средства охватывают полный жизненный цикл разработки информационной системы. Они позволяют обмениваться данными между инструментами верхнего и нижнего уровня и являются своего рода «мостом» между CASE средствами верхнего и нижнего уровней.

Для моделирования и оптимизации бизнес процессов применяются CASE средства верхнего уровня и интегрированные CASE средства. Они позволяют повысить качество моделей бизнес процессов за счет автоматического контроля, дают возможность оценить ожидаемый результат, ускоряют процесс проектирования, обеспечивают возможности по изменению и обновлению моделей.

## 32. Графические средства для описания и документирования информационных систем.

Графические средства относят к CASE-средствам.

Графические средства моделирования предметной области позволяют разработчикам в наглядном виде изучать существующую ИС, перестраивать ее в соответствии с поставленными целями и имеющимися ограничениями.

Существуют графические средства анализа и проектирования, обеспечивающие создание и редактирование иерархически связанных диаграмм (DFD, ERD и др.), образующих модели ИС;

**CASE** (англ. *computer-aided software engineering*) — набор инструментов и методов программной инженерии для проектирования программного обеспечения, который помогает обеспечить высокое качество программ, отсутствие ошибок и простоту в обслуживании программных продуктов.[1] Также под CASE понимают совокупность методов и средств проектирования информационных систем с использованием CASE-инструментов[2].

**Средства автоматизации разработки программ** (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 ISO/IEC 14102 CASE-средства стали определять, как программные средства для поддержки процессов жизненного цикла ПО[3].

## Общая информация

Основной целью CASE-технологии является разграничение процесса проектирования программных продуктов от процесса кодирования и последующих этапов разработки, максимально автоматизировать процесс разработки. Для выполнения поставленной цели CASE-технологии используют два принципиально разных подхода к проектированию: структурный и объектно-ориентированный.

Структурный подход предполагает декомпозицию (разделение) поставленной задачи на функции, которые необходимо автоматизировать. В свою очередь, функции также разбиваются на подфункции, задачи, процедуры. В результате получается упорядоченная иерархия функций и передаваемой информацией между функциями.

Структурный подход подразумевает использование определённых общепринятых методологий при моделировании различных информационных систем:

1. SADT (structured analysis and design technique);
2. DFD (data flow diagrams);
3. ERD (entity-relationship diagrams).

Существует три основных типа моделей, используемых при структурном подходе: функциональные, информационные и структурные.

Основным инструментом объектно-ориентированного подхода является язык UML — унифицированный язык моделирования, который предназначен для визуализации и документирования объектно-ориентированных систем с ориентацией их на разработку программного обеспечения. Данный язык включает в себя систему различных диаграмм, на основании которых может быть построено представление о проектируемой системе.

## Классификация

В функции CASE входят средства анализа, проектирования и программирования программных средств, проектирования интерфейсов, документирования и производства структурированного кода на каком-либо языке программирования.[4]

CASE-инструменты классифицируются по типам и категориям.

Классификация по типам отражает функциональную ориентацию средств на те или иные процессы жизненного цикла разработки программного обеспечения, и, в основном, совпадают с компонентным составом крупных интегрированных CASE-систем, и включает следующие типы:

* средства анализа — предназначены для построения и анализа модели предметной области;
* средства проектирования баз данных;
* средства разработки приложений;
* средства реинжиниринга процессов;
* средства планирования и управления проектом;
* средства тестирования;
* средства документирования.

Классификация по категориям определяет степень интегрированности по выполняемым функциям и включают — отдельные локальные средства, решающие небольшие автономные задачи, набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла и полностью интегрированных средств, охватывающий весь жизненный цикл информационной системы и связанных общим репозиторием.

Типичными CASE-инструментами являются:

* инструменты управления конфигурацией;
* инструменты моделирования данных;
* инструменты анализа и проектирования;
* инструменты преобразования моделей;
* инструменты редактирования программного кода;
* инструменты рефакторинга кода;
* генераторы кода;
* инструменты для построения UML-диаграмм.

# **CASE средства**

CASE средства используются при создании и разработке информационных систем управления предприятиями. Применительно к моделированию бизнес процессов они могут рассматриваться как инструментарий для совершенствования и непрерывного улучшения работы.

CASE средства (Computer - Aided Software Engineering) – это инструмент, который позволяет автоматизировать процесс разработки информационной системы и программного обеспечения. Разработка и создание информационных систем управления предприятием связаны с выделением бизнес-процессов, их анализом, определением взаимосвязи элементов процессов, оптимизации их инфраструктуры и т.д. Основной целью применения CASE средств является сокращение времени и затрат на разработку информационных систем, и повышение их качества.

Многие современные CASE средства предоставляют возможности для моделирования практически всех предметных областей деятельности организаций. В составе этих средств существуют инструменты для описания моделей бизнес-процессов за счет различных диаграмм, схем, графов и таблиц.

Классификация CASE средств

Из всего многообразия CASE средств, существующих на сегодняшний день, можно выделить три основные группы. Эти группы связаны с этапами разработки информационных систем и их жизненным циклом. Классификация CASE средств осуществляется в зависимости от того, какие из этапов разработки они поддерживают.

Выделяют следующие группы CASE средств:

* CASE средства верхнего уровня. Эти CASE средства ориентированы на начальные этапы построения информационной системы. Они связаны с анализом и планированием. CASE средства верхнего уровня обеспечивают стратегическое планирование, расстановку целей, задач и приоритетов, а также графическое представление необходимой информации. Все CASE средства верхнего уровня содержат графические инструменты построения диаграмм, таких как диаграммы сущность-связь (ER диаграммы), диаграммы потока данных (DFD), структурные схемы, деревья решений и пр.
* CASE средства нижнего уровня. Эти CASE средства больше сфокусированы на последних этапах разработки информационной системы – проектирование, разработка программного кода, тестирование и внедрение. CASE средства нижнего уровня зависят от данных, которые предоставляют средства верхнего уровня. Они используются разработчиками приложений и помогают создать информационную систему, однако не являются полноценными инструментами разработки программного обеспечения.
* Интегрированные CASE средства (I – CASE). Эти CASE средства охватывают полный жизненный цикл разработки информационной системы. Они позволяют обмениваться данными между инструментами верхнего и нижнего уровня и являются своего рода «мостом» между CASE средствами верхнего и нижнего уровней.

Для моделирования и оптимизации бизнес процессов применяются CASE средства верхнего уровня и интегрированные CASE средства. Они позволяют повысить качество моделей бизнес процессов за счет автоматического контроля, дают возможность оценить ожидаемый результат, ускоряют процесс проектирования, обеспечивают возможности по изменению и обновлению моделей.

Характеристики CASE средств

Основными характеристиками CASE средств, важными с точки зрения моделирования и оптимизации бизнес процессов, являются следующие:

* Наличие графического интерфейса. Для представления моделей процессов CASE средства должны обладать возможностью отображать процессы в виде схем. Схемы много проще в использовании, чем различные текстовые и числовые описания. Это позволяет получать легко управляемые компоненты модели, обладающие простой и ясной структурой.
* Наличие репозитория. Репозиторий это общая база данных, которая содержит описание элементов процессов и отношений между ними. Каждый объект репозитария должен обладать перечнем свойств, характерных только для этого объекта.
* Гибкость применения. Эта характеристика дает возможность представлять бизнес процессы в различных вариантах, важных с точки зрения анализа. CASE средства должны позволять проводить анализ процессов и создавать модели, сфокусированные на различных аспектах деятельности предприятия.
* Возможность коллективной работы. Анализ и моделирование процессов может требовать совместной работы нескольких человек. Для одновременной работы над моделями процессов CASE средства должны обеспечивать управление изменениями любыми фрагментами моделей и их модификацией при коллективном доступе.
* Построение прототипов. Прототипы процессов необходимы для того, чтобы на ранних стадиях изменения процессов можно было понять, насколько процесс будет соответствовать требованиям.
* Построение отчетов. CASE средства должны обеспечивать построение отчетов по всем моделям процессов с учетом взаимосвязи элементов. Такие отчеты необходимы для анализа моделей и определения возможностей по оптимизации. За счет отчетов обеспечивается контроль полноты и достаточности моделей, уровень декомпозиции процессов, правильность синтаксиса диаграмм и типов применяемых элементов.

Выбор CASE средств

Выбор CASE средств для анализа и моделирования процессов зависит от многих факторов – финансовых возможностей, функциональных характеристик, подготовки персонала, применяемых информационно-технических средств и пр. Приводить исчерпывающий состав этих факторов не имеет смысла, т.к. в ситуации выбора для каждого конкретного случая этот состав будет изменяться. Тем не менее, можно определить набор «базовых» факторов, на основании которых определяются критерии по выбору CASE средств.

К таким «базовым» факторам можно отнести следующие:

* Цели моделирования и анализа процессов. Исходя из целей моделирования, определяются необходимые методы, которые должны поддерживать CASE средства. Также цели моделирования определяют необходимый уровень детализации моделей и формы представления отчетов.
* Удобство для пользователей. Этот фактор определяет набор критериев для представления результатов моделирования наиболее понятным и приемлемым способом. Выбор CASE средств необходимо проводить с учетом того, чтобы пользователям приходилось затрачивать как можно меньше усилий на работу в среде CASE средств. CASE средства должны быть визуально и интуитивно понятны пользователям.
* Применение стандартных методологий. Этот фактор определяет критерии выбора CASE средств, связанные с применением стандартных методологий анализа и моделирования бизнес процессов. Как правило, моделирование не заканчивается созданием новых моделей процессов. Модели используются для внедрения информационных систем управления и автоматизации процессов. За счет стандартизации обеспечивается упрощение взаимодействия между CASE средствами и различными информационными системами.
* Удобство эксплуатации. При выборе CASE средств необходимо учитывать такие характеристики как эффективность применения, сопровождаемость, переносимость моделей с одной системы на другую. Этот фактор в значительной степени связан с критериями, относящимися к техническим характеристикам аппаратного обеспечения.
* Трудоемкость. Этот фактор определяет набор критериев, связанных с освоением и изучением работы CASE средств. При выборе следует учесть, сколько времени потребуется на обучение пользователей.
* Субъективность. Данный фактор также не следует исключать из набора критериев по выбору CASE средств. При выборе могут существовать субъективные соображение выбора того или иного CASE средства, не связанные с рациональными критериями выбора.

## 33. CASE-средства генерации исходных текстов и реализации интегрированного окружения поддержки полного жизненного цикла (ЖЦ) разработки ПО.

CASE-средства генерации исходных текстов и реализации интегрированного окружения поддержки полного ЖЦ разработки ПО (2-ая генерация CASE-II).

CASE-II отличается значительно более развитыми возможностями, улучшенными характеристиками и исчерпывающим подходом к полному ЖЦ. В ней в первую очередь используются средства поддержки автоматической кодогенерации, а также обеспечивается полная функциональная поддержка для порождения графических системных требований и спецификаций проектирования; контроля, анализа и связывания системной информации, а также информации по управлению проектированием; построения прототипов и моделей системы; тестирования, верификации и анализа сгенерированных программ; генерации документов по проекту; контроля на соответствие стандартам по всем этапам ЖЦ. CASE-II может включать свыше 100 функциональных компонент, поддерживающих все этапы ЖЦ, при этом пользователям предоставляется возможность выбора необходимых средств и их интеграции в нужном составе.

ИЛИ

CASE-II - более развитая технология с значительно улучшенными характеристиками. В отличии от CASE-I поддерживает полный жизненный цикл. Обеспечивает полную функциональную поддержку для создания графических спецификаций и системных требований при проектировании, контролирует, анализирует , связывает системную информацию и информацию по управлению разработкой.

ИЛИ

Важной особенностью инструмента CASE второго поколения является то, что предназначение любой специализированной методологии. Для этого может потребоваться работа организации-администратора CASE, которая может адаптировать инструмент CASE к конкретной методологии. Кроме того, инструменты CASE второго поколения имеют следующие особенности:

o-**Интеллектуальная поддержка диаграмм:**

Тот факт, что методы схематизации полезны для системного анализа и стиля, хорошо известен. инструменты CASE, рассчитанные на более длительный срок, могут помочь в эстетическом и механическом размещении диаграмм.

o-**Интеграция со средой реализации:**

Инструменты CASE должны обеспечивать интеграцию между стилем и реализацией.

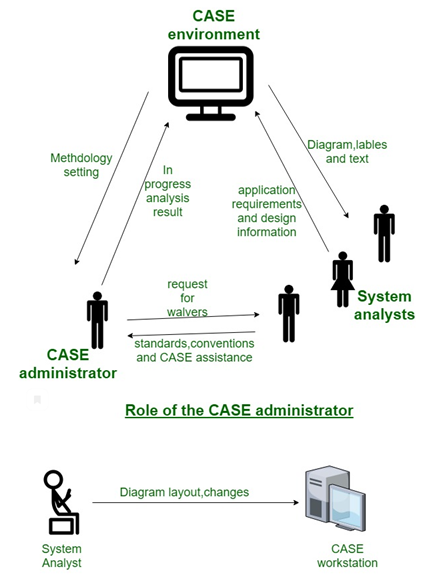
o-**Стандарты словаря данных:**

Пользователь должен иметь возможность интегрировать несколько инструментов разработки в одну настройку. Крайне маловероятно, чтобы кто-либо из продавцов был готов предоставить полное разрешение. Более того, наиболее популярному инструменту потребуется калибровка для выбранной системы. поэтому пользователь будет действовать как системный планиметр. это часто, вероятно, учитывая, что появляется какое-то банальное в словарях знаний.

o-**Поддержка настройки:**

Пользователь должен иметь возможность наметить новые виды объектов и соединений. Этот объект также привык к созданию некоторых специальных методологий. В идеале, было бы целесообразно указать принципы стратегии для механизма правил для завершения необходимых проверок согласованности.

Ниже на рисунках представлена функциональность инструмента автоматизации второго поколения CASE:



CASE-модель жизненного цикла ПО



## 34. CASE-средства программирования.

Средства этой группы поддерживают этапы программирования и тестирования, а также автоматическую кодогенерацию из спецификаций, получая полностью документированную выполняемую программу: *COBOL 2/Workbench (Mikro Focus), DECASE (DEC), NETRON/CAP (Netron), APS (Sage Software).* Помимо диаграммеров различного назначения и средств поддержки работы с репозитарием, в эту группу средств включены и традиционные генераторы кодов, анализаторы кодов (как в статике, так и в динамике), генераторы наборов тестов, анализаторы покрытия тестами, отладчики.

ПРИМЕРЫ

|  |  |
| --- | --- |
| Средства программирования | SQL Windows, Power Builder, Delphi |

1) PowerBuilder - это объектно-ориентированный инструмент для профессиональной разработки приложений в среде клиент/сервер, позволяющий коллективам разработчиков легко и быстро создавать графические приложения, которые имеют доступ к базам данных и другой корпоративной информации, хранящейся локально или на сетевых серверах.

2) **Delphi** — структурированный объектно-ориентированный язык программирования с жесткой привязкой к типам данных. В основном используется для прикладного программирования в среде Windows. Ранее среда разработки Delphi имела возможность разрабатывать приложения только для операционной системы Windows, со временем появился вариант для OS Linux под названием Kylix

**Объектно-ориентированные особенности языка**

· Инкапсуляция - Позволяет объединить методы и данные единый объект с возможностью скрыть детали от пользователя. Своего рода «черный ящик», пользователь не знает детали реализации и работает с ним путем предоставленного интерфейса.

### · Наследование - Очень удобная особенность, позволяет унаследовать от родителя все свойства и функции, как бы копирует объект для дальнейшего усовершенствования и добавления новых возможностей.

### · Полиморфизм - Третья особенность объектно-ориентированного языка дельфи, позволяет объединить данные разных объектов с единой спецификацией имея различную реализацию.

## 35. CASE-средства проектирования баз данных.

Средства проектирования баз данных, обеспечивающие моделирование данных и генерацию схем баз данных (как правило, на языке SQL) для наиболее распространенных СУБД. К ним относятся ERwin (Logic Works), S-Designor (SDP) и DataBase Designer (ORACLE). Средства проектирования баз данных имеются также в составе CASE-средств Vantage Team Builder, Designer/2000, Silverrun и PRO-IV;

ПРИМЕРЫ

|  |  |
| --- | --- |
| Средства проектирования баз данных и файлов | ERwin, S-Designor, PRO-IV, CASE.Синтез+ |

**ERwin -** CASE-средство, которое в качестве методологии использует IDEF1X. Также имеется возможность использовать методологию IE.

IDEF1X была разработана прежде всего для использования в армии США. Широко используется в государственных учреждениях США, в корпорациях финансовых и промышленных направленностей. Методологию IE (Information Engineering) разработали Мартин и Финкельштейн. Она используется большей частью в промышленности.

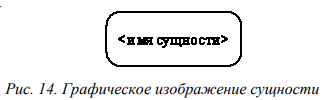
Выпуск ERwin осуществляется в нескольких конфигурациях, они ориентированны на распространенные средства разработки приложений 4GL.

**S-Designor** - CASE-средство фирмы *Powersoft,* реализующее стандартную методологию. Является средством проектирования реляционных баз данных . По функциональным возможностям близок к CASE-средству ERwin.

НЕ ЗНАЮ, ПРО ЭТО ВСТАВЛЯТЬ ЧИ НЕ ВСТАВЛЯТЬ

Наиболее распространенным средством моделирования данных являются диаграммы "сущность-связь" (ERD). С их помощью определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). ERD непосредственно используются для проектирования реляционных баз данных.

Сущность (Entity) - реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению (рисунок 14). Рис. 14.



Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Каждая сущность должна обладать некоторыми свойствами:

каждая сущность должна иметь уникальное имя, и к одному и тому же имени должна всегда применяться одна и та же интерпретация. Одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;

сущность обладает одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;

сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;

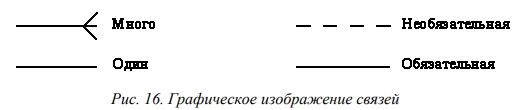
каждая сущность может обладать любым количеством связей с другими сущностями модели.

Связь (Relationship) - поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Связь - это ассоциация между сущностями. При этом каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком. Каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя. Связи может даваться имя, выражаемое грамматическим оборотом глагола и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Имя связи всегда формируется с точки зрения родителя, так что предложение может быть образовано соединением имени сущности-родителя, имени связи, выражения степени и имени сущности потомка. Например, связь продавца с контрактом может быть выражена следующим образом:

продавец может получить вознаграждение за 1 или более контрактов;

контракт должен быть инициирован ровно одним продавцом.

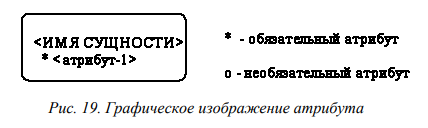
Степень связи и обязательность графически изображаются следующим образом (рисунок 16).



Последним шагом моделирования является идентификация атрибутов.

Атрибут - любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных со множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, пар предметов и т.д.). Экземпляр атрибута - это определенная характеристика отдельного элемента множества. Экземпляр атрибута определяется типом характеристики и ее значением, называемым значением атрибута. В ERмодели атрибуты ассоциируются с конкретными сущностями. Таким образом, экземпляр сущности должен обладать единственным определенным значением для ассоциированного атрибута.

Атрибут может быть либо обязательным, либо необязательным (рисунок 19).



Обязательность означает, что атрибут не может принимать неопределенных значений (null values). Атрибут может быть либо описательным (т.е. обычным дескриптором сущности), либо входить в состав уникального идентификатора (первичного ключа).

Уникальный идентификатор - это атрибут или совокупность атрибутов и/или связей, предназначенная для уникальной идентификации каждого экземпляра данного типа сущности. В случае полной идентификации каждый экземпляр данного типа сущности полностью идентифицируется своими собственными ключевыми атрибутами, в противном случае в его идентификации участвуют также атрибуты другой сущности-родителя. Каждый атрибут идентифицируется уникальным именем, выражаемым грамматическим оборотом существительного, описывающим представляемую атрибутом характеристику. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Атрибуты, определяющие первичный ключ, размещаются наверху списка и выделяются знаком "#". Каждая сущность должна обладать хотя бы одним возможным ключом. Возможный ключ сущности - это один или несколько атрибутов, чьи значения однозначно определяют каждый экземпляр сущности. При существовании нескольких возможных ключей один из них обозначается в качестве первичного ключа, а остальные - как альтернативные ключи.

## 36. CASE-средства реинженеринга.

Средства реинжиниринга, обеспечивающие анализ программных кодов и схем баз данных и формирование на их основе различных моделей и проектных спецификаций. Средства анализа схем БД и формирования ERD входят в состав Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin и S-Designor. В области анализа программных кодов наибольшее распространение получают объектно-ориентированные CASEсредства, обеспечивающие реинжиниринг программ на языке С++ (Rational Rose (Rational Software), Object Team (Cayenne)).

*ПРИМЕРЫ*

|  |  |
| --- | --- |
| Средства сопровождения и реинжиниринга | PRO-IV, Rational Rose, Object Team |

**1)**--Rational Rose является самым известным CASE-средством.CASE-средство Rational Rose - продукт Американской фирмы Rational Software Corporation, который автоматизирует этапы проектирования и анализа программного обеспечения. В основе работы данного средства лежит синтез-методология объектно-ориентированного анализа и проектирования.

## 37. Верхние, средние и нижние CASE-средства.

Можно выделить классификацию по уровням. Различают 3 уровня CASE-средств:

1)--Upper CASE;

Верхние (Upper) CASE часто называют средствами компьютерного планирования. Они призваны повышать эффективность деятельности руководителей фирмы и проекта путем сокращения затрат на определение политики фирмы и на создание общего плана проекта. Этот план включает цели и стратегии их достижения, основные действия в свете целей и задач фирмы, установление стандартов на различные виды взаимосвязей и т.д. Использование верхних CASE позволяет построить модель предметной области, отражающую всю существующую специфику. Она направлена на понимание общего и частного механизмов функционирования, имеющихся возможностей, ресурсов, целей проекта в соответствии с назначением фирмы. Эти средства позволяют проводить анализ различных сценариев (в том числе наилучших и наихудших), накапливая информацию для принятия оптимальных решений

2)--Middle CASE;

Средние (Middle) CASE считаются средствами поддержки этапов анализа требований и проектирования спецификаций и структуры ПО. Их использование существенно сокращает цикл разработки проекта; при этом важную роль играет возможность накопления и хранения знаний, обычно имеющихся только в голове разработчика-аналитика, что позволит использовать накопленные решения при создании других проектов. Основная выгода от использования среднего CASE состоит в значительном облегчении проектирования систем, проектирование превращается в итеративный процесс, включающий следующие действия:

пользователь обсуждает с аналитиком требования к проектируемой системе;

аналитик документирует эти требования, используя диаграммы и словари входных данных;

пользователь проверяет эти диаграммы и словари, при необходимости модифицируя их;

аналитик отвечает на эти модификации, изменяя соответствующие спецификации.

Кроме того, средние CASE обеспечивают возможности быстрого документирования требований и быстрого прототипирования.

3)--Low CASE.

Нижние (Lower) CASE являются средствами разработки ПО (при этом может использоваться до 30% спецификаций, созданных средствами среднего CASE). Они содержат системные словари и графические средства, исключающие необходимость разработки физических спецификаций. Имеются системные спецификации, которые непосредственно переводятся в программные коды разрабатываемой системы (при этом автоматически генерируется до ХО-90% кодов). На эти средства возложены также функции тестирования, управления конфигурацией, формирования документации. Главными преимуществами нижних CASE являются: значительное уменьшение времени на разработку, облегчение модификаций, поддержка возможностей прототипирования (совместно со средними CASE).

Примеры CASE-средств данной классификации рассмотрены в таблице 2.

*Таблица 2*

Примеры CASE-средств, классифицированных по уровням.

|  |  |
| --- | --- |
|  | **Примеры CASE-средств** |
| Upper CASE | MS Project, SE Companion |
| Middle CASE | CASE.Аналитик, Silverrun, DESIGNER/2000 |
| Low CASE | Power Builder, Delphi, Uniface, JAM |

**ВЫСШИЕ**

**1)**--**MS Project** - это программа для управления проектами. Она помогает менеджерам контролировать исполнение поставленных задач, а также грамотно распределять ресурсы. Основной рабочий инструмент в ней — диаграмма Ганта.

Локальные решения MS Project включают в себя следующие пакеты:

·-- Стандартный, или Standart

Версия для небольших проектов. Рассчитана на 1 пользователя.

·-- Профессиональный, или Professional

Корпоративная программа для совместного доступа к управлению.

·-- Portfolio Server

Программа для анализа проектов и ключевых показателей эффективности по ним.

Облачное решение Project позволяет работать над проектами, просматривать отчетность, анализировать прогресс, и все это в браузере.

**2)**--**SE Companion** – набор инструментов управления процессами и онлайновыми средствами, облегчающий разработчикам перестройку делового процесса.

Оно позволяет:

· создать гипертекстовое описание методологии в виде иерархии описания стадий, этапов и операций разработки;

· создать гипертекстовое описание всех методов и методик реализации процессов ЖЦ ПО;

· выделить из гипертекстового описания иерархию процессов ЖЦ ПО для планирования и управления процессом создания ПО (иерархию работ);

· изменять гипертекстовые описания ЖЦ и методов так, как это необходимо разработчику, иными словами, производить авторизацию методологии и отслеживать эти изменения в иерархии работ, предназначенной для управления проектом;

· привязать к процессам ЖЦ инструментальные средства поддержки этих процессов и обеспечить вызов инструментальных средств из соответствующих экранов гипертекстового справочника;

· обеспечить просмотр гипертекстовых экранов описания используемых методов из инструментальных средств;

· обеспечить поддержку процесса управления разработкой, в частности, за счет взаимодействия со средством планирования работ MS Project, оценивания трудоемкости проекта, отслеживания выполнения работ, создания графиков работ, и др.

**СРЕДНИЕ**

**1)**--**CASE.Аналитик** - отечественное CASE-средство. На данный момент наиболее конкурентоспособно на российском рынке. База данных является открытой, реализована в СУБД Paradox. Имеется возможность создания отчетов разной направленности по проекту. Данное CASE-средство генерирует макеты документов по требованиям ГОСТ.

**2)**--**Silverrun** - CASE-средство Silverrun разработано американской фирмой CSA (Сomputer Systems Advisers, Inc.). Silverrun ориентирован на спиральную(циклическую) модель жизненного цикла программного продукта. Предназначено для проектирования и анализа ИС бизнес-класса. Структура Silverrun модульная, где каждый модуль является самостоятельным продуктом , что позволяет использовать модули отдельно друг от друга. Данное CASE-средство состоит из 4 модулей:

§ BPM - Business Process Modeler - модуль построения моделей бизнес-процессов в форме диаграмм потоков данных;

§ ERX - Entity-Relationship eXpert - модуль концептуального моделирования данных;

§ RDM - Relational Data Modeler - модуль реляционного моделирования;

§ WRM - Workgroup Repository Manager - менеджер репозитория рабочей группы.

Достоинство данного CASE-средства в его высокой гибкости, в большом разнообразии изобразительных средств при построении моделей.

Главным недостатком данного CASE-средства является отсутствие жесткого взаимоконтроля компонентов различных моделей между собой.

Silverrun функционирует на платформах MS Windows, Macintosh и OS/2 Presentation Manager. При взаимодействии с другими средствами возможно 3 способа выдачи данных работы во внешние файлы:

§ Система отчетов. Выдает отчет в форме текстового файла, определив содержимое по репозиторию. Полученный файл можно использововать в дальнейшем: добавить в другой отчет, добавить в текстовый редактор;

§ Система экспорта/импорта. Имеется возможность не просто форматирования , но и добавления в репозиторий, что позволяет обмениваться информацией с другими CASE средствами, электронными таблицами, текстовыми редакторами,СУБД. Также доступно определение не только содержимое экспортного файла, но и его оформление, т.е. определение маркера начала/конца полей и т.п.;

§ Хранение репозитория во внешних файлах через ODBC-драйверы. Имеется возможность хранить все проектные данные в формате СУБД.

НИЗКОУРОВНЕВЫЕ

1) PowerBuilder - это объектно-ориентированный инструмент для профессиональной разработки приложений в среде клиент/сервер, позволяющий коллективам разработчиков легко и быстро создавать графические приложения, которые имеют доступ к базам данных и другой корпоративной информации, хранящейся локально или на сетевых серверах.

2) **Delphi** — структурированный объектно-ориентированный язык программирования с жесткой привязкой к типам данных. В основном используется для прикладного программирования в среде Windows. Ранее среда разработки Delphi имела возможность разрабатывать приложения только для операционной системы Windows, со временем появился вариант для OS Linux под названием Kylix

**Объектно-ориентированные особенности языка**

· Инкапсуляция - Позволяет объединить методы и данные единый объект с возможностью скрыть детали от пользователя. Своего рода «черный ящик», пользователь не знает детали реализации и работает с ним путем предоставленного интерфейса.

### · Наследование - Очень удобная особенность, позволяет унаследовать от родителя все свойства и функции, как бы копирует объект для дальнейшего усовершенствования и добавления новых возможностей.

### · Полиморфизм - Третья особенность объектно-ориентированного языка дельфи, позволяет объединить данные разных объектов с единой спецификацией имея различную реализацию.

## 38. Процесс разработки программ для ИС и АС при помощи CASE-средств.

Возрастающая сложность современных автоматизированных систем управления и повышение требовательности к ним обуславливает применение эффективных технологий создания и сопровождения АС в течение всего жизненного цикла. Такие технологии, базирующиеся на методологиях подготовки информационных систем и соответствующих комплексах интегрированных инструментальных средств, а также ориентированные на поддержку полного жизненного цикла АС или его основных этапов, получили название CASE-технологий и CASE-средств. Для успешной реализации проекта АС должны быть построены полные и непротиворечивые функциональные и информационные модели системы управления. Накопленный опыт проектирования указанных моделей показывает, что это логически сложная , трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Однако во многих случаях проектирование АС выполняется в основном на интуитивном уровне с применением неформальных методов, основанных на искусстве, практическом опыте и экспертных оценках. Кроме того, в процессе создания и функционирования АС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение автоматизированных систем управления. От перечисленных недостатков в наибольшей степени свободны подходы, основанные на программно-технических средствах специального класса - CASE-средствах, реализующих CASE-технологии создания и сопровождения АС.

Под термином CASE (Computer Aided Software Engineering) понимаются программные средства, поддерживающие процессы создания и сопровождения АС, включая анализ и формулировку требований, проектирование прикладного программного обеспечения и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным программным обеспечением и техническими средствами образуют полную среду разработки АС.

На фазе анализа и планирования требований пользователи АС определяют функции, которые она должна выполнять, выделяют наиболее приоритетные из них, требующие проработки в первую очередь, описывают информационные потребности. Формулирование требований к АС осуществляется в основном силами пользователей под руководством специалистов-разработчиков. Ограничивается масштаб проекта АС, устанавливаются временные рамки для каждой из последующих фаз. Кроме того, определяется сама возможность реализации проекта в заданных размерах финансирования, на имеющихся аппаратных средствах и т.д. Результатом этого этапа должен быть список расставленных по приоритету функций будущей АС, а также предварительные функциональные модели АС.

На этапе проектирования часть пользователей принимает участие в техническом проектировании системы под руководством специалистов-разработчиков. CASE-средства используются для быстрого получения работающих прототипов приложений. Пользователи, непосредственно взаимодействуя с ними, уточняют и дополняют требования к системе, которые не были выявлены на предыдущей фазе. Более подробно рассматриваются процессы системы. Анализируется и при необходимости корректируется функциональная модель. Каждый процесс рассматривается детально. При необходимости для элементарного процесса создается частичный прототип: экран, диалог, отчет, устраняющий неясности или неоднозначности. Устанавливаются требования разграничения доступа к данным. На этой же фазе происходит определение необходимой документации. После детального определения состава процессов оценивается количество функциональных элементов разрабатываемой системы и принимается решение о разделении АС на подсистемы, поддающиеся реализации одной командой разработчиков за приемлемое для RAD-проектов время (60 - 90 дней). С использованием CASE-средств проект АС распределяется между различными командами (делится функциональная модель). Результатом данного этапа должны быть: общая информационная модель системы; функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков; точно определенные с помощью CASE-средств интерфейсы между автономно разрабатываемыми подсистемами; построенные прототипы экранов, отчетов, диалогов. Все модели и прототипы должны быть получены с применением тех CASE-средств, которые будут использоваться в дальнейшем при построении системы. Данное требование вызвано тем, что в традиционном подходе при передаче информации о проекте с этапа на этап нередко происходит неконтролируемое искажение данных. Применение единой среды хранения данных о проекте позволяет этого избежать. В отличие от обычных подходов, при которых используются специфические средства прототипирования, не предназначенные для построения реальных приложений, а прототипы выбрасываются после устранения неясностей в проекте АС, в подходе RAD каждый прототип передается будущей системе. Таким образом, на следующую фазу передается более полная и полезная информация.

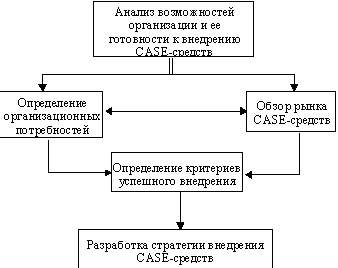
На этапе построения осуществляется непосредственно сама быстрая подготовка приложения. При этом разработчики выполняют итеративное построение реальной АСУ на основе полученных в предыдущей фазе моделей, а также требований нефункционального характера. Программный код частично формируется CASE-средствами автоматически. Конечные пользователи на этой фазе оценивают получаемые результаты и вносят коррективы, если в процессе разработки система перестает удовлетворять указанным ранее требованиям. Тестирование автоматизированной системы осуществляется в процессе разработки. После окончания работ каждой отдельной команды разработчиков производится постепенная интеграция данной части системы с остальными, формируется полный программный код, выполняется тестирование совместной работы данной части приложения, а затем тестирование АС в целом. Завершается физическое проектирование АС, включающее: определение необходимости распределения данных; анализ использования данных; физическое проектирование базы данных; определение требований к аппаратным ресурсам и способов увеличения производительности, завершение разработки документации проекта. Результатом данного этапа является готовая автоматизированная система, удовлетворяющая всем согласованным требованиям.

На фазе внедрения АС производится обучение пользователей и вносятся организационные изменения. Для этого этапа характерно то, что одновременно с внедрением новой АС осуществляется работа с существующей системой управления до полного внедрения новой. Так как фаза построения достаточно непродолжительна, планирование и подготовка к внедрению должны начинаться заранее, как правило, на этапе проектирования системы. Приведенная схема разработки АС не является окончательной. Возможны различные варианты, зависящие, например, от начальных условий, в которых ведется создание АС: а) разрабатывается совершенно новая система; б) было проведено обследование предприятия и существует модель его деятельности; в) на предприятии уже существует АС, которая может быть использована

Итак, вы решились на внедрение CASE-средств. Процесс внедрения состоит из следующих этапов:

определение потребностей в CASE-средствах;  
 оценка и выбор CASE-средств;  
 выполнение пилотного проекта;  
 практическое внедрение CASE-средств.

Определение потребностей в CASE-средствах можно проиллюстрировать следующей диаграммой (см. рис. 1).

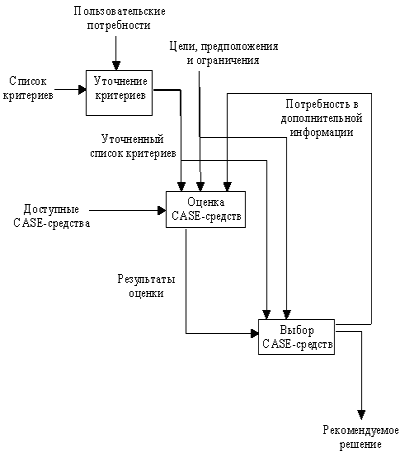


Данный этап включает достижение понимания потребностей организации и технологии последующего процесса внедрения CASE-средств. Он должен привести к выделению тех областей деятельности организации, в которых применение CASE-средств может принести реальную пользу. Результатом данного этапа является документ, определяющий стратегию внедрения.

Процесс оценки и выбора CASE-средств можно рассмотреть в виде модели. Этот процесс может преследовать несколько целей и включать:

оценку нескольких CASE-средств и выбор одного или более из них;  
 оценку одного или более CASE-средств и сохранение результатов для последующего использования;  
 выбор одного или более CASE-средств с использованием результатов предыдущих оценок.

Ниже приведена диаграмма, описывающая наиболее общую ситуацию оценки и выбора, а также показывает зависимость между ними (см. рис. 2).



Как видно из рисунка, входной информацией для процесса оценки является:

определение пользовательских потребностей;  
 цели и ограничения проекта;  
 данные о доступных CASE-средствах;  
 список критериев, используемых в процессе оценки.

Результаты оценки могут включать результаты предыдущих оценок. При этом не следует забывать, что набор критериев, использовавшихся при предыдущей оценке, должен быть совместимым с текущим набором. Конкретный вариант реализации процесса (оценка и выбор, оценка для будущего выбора или выбор, основанный на предыдущих оценках) определяется перечисленными выше целями.

Элементы процесса включают:

цели, предположения и ограничения, которые могут уточняться в ходе процесса;  
 потребности пользователей, отражающие количественные и качественные требования пользователей к CASE-средствам;  
 критерии, определяющие набор параметров, в соответствии с которыми производится оценка и принятие решения о выборе;  
 формализованные результаты оценок одного или более средств;  
 рекомендуемое решение (обычно либо решение о выборе, либо дальнейшая оценка).

Процесс оценки и/или выбора следует начинать только тогда, когда лицо, группа или организация полностью определила для себя конкретные потребности и формализовала их в виде количественных и качественных требований в заданной предметной области. Далее термин "пользовательские требования" означает именно такие формализованные требования.

Пользователь должен определить конкретный порядок действий и принятия решений с любыми необходимыми итерациями. Например, процесс можно представить в виде дерева решений с его последовательным обходом и выбором подмножеств кандидатов для более детальной оценки. Описание последовательности действий должно определять поток данных между ними.

Определение списка критериев основано на пользовательских требованиях и включает:

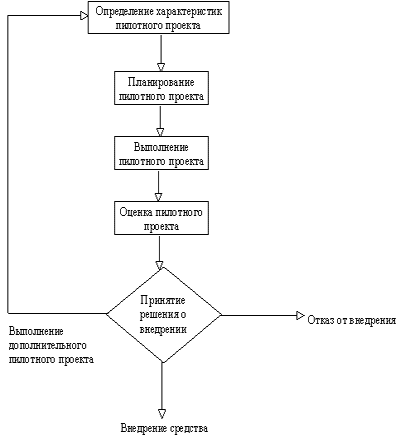
выбор критериев для использования из приведенного далее перечня;  
 определение дополнительных критериев;  
 определение области использования каждого критерия (оценка, выбор или оба процесса);  
 определение одной или более метрик для каждого критерия оценки;  
 назначение веса каждому критерию при выборе.

Перед полномасштабным внедрением выбранного CASE-средства в организации выполняется пилотный проект. Его цель — экспериментальная проверка правильности решений, принятых на предыдущих этапах, и подготовка к внедрению.

Пилотный проект представляет собой первоначальное реальное использование CASE-средства и обычно подразумевает более широкий масштаб использования CASE-средства по отношению к тому, который был достигнут во время оценки. Пилотный проект должен обладать многими из характеристик реальных проектов, для которых предназначено данное средство. Он преследует следующие цели:

подтвердить достоверность результатов оценки и выбора;  
 определить, действительно ли CASE-средство годится для использования в данной организации, и если да, то определить наиболее подходящую область его применения;  
 собрать информацию, необходимую для разработки плана практического внедрения;  
 приобрести собственный опыт использования CASE-средства.

Пилотный проект позволяет получить важную информацию, необходимую для оценки качества функционирования CASE-средства и его поддержки со стороны поставщика после того, как средство установлено. Его реализацию можно проиллюстрировать следующей схемой (см. рис. 3).



Важной функцией пилотного проекта является принятие решения относительно приобретения или отказа от использования CASE-средства. Провал пилотного проекта позволяет избежать более значительных и дорогостоящих неудач в дальнейшем, поскольку он обычно связан с приобретением относительно небольшого количества лицензий и обучением узкого круга специалистов.

Ну и, наконец, наступает переход к практическому использованию CASE-средств. Он начинается с разработки и последующей реализации плана перехода.

План перехода должен включать следующее:

Информацию относительно целей, критериев оценки, графика и возможных рисков, связанных с реализацией плана.  
 Информацию относительно приобретения, установки и настройки CASE-средств.  
 Информацию относительно интеграции каждого средства с существующими средствами, включая как интеграцию CASE-средств друг с другом, так и их интеграцию в процессы разработки и эксплуатации ПО, существующие в организации.  
 Ожидаемые потребности в обучении и ресурсы, используемые в течение и после завершения процесса перехода.  
 Определение стандартных процедур использования средств.

Реализация плана перехода требует постоянного мониторинга использования CASE-средств, обеспечения текущей поддержки, сопровождения и обновления средств по мере необходимости. Достигнутые результаты должны периодически подвергаться экспертизе в соответствии с графиком, а план перехода — корректироваться при необходимости. Необходимо постоянно уделять внимание удовлетворению потребностей организации и критериям успешного внедрения CASE-средств.

Значимой и неотъемлемой частью реализации плана является также обучение и переобучение. Как правило, все понимают: обучение представляет собой центральное звено, обеспечивающее нормальное использование CASE-средств в организации. Тем не менее существует довольно распространенная ошибка, что начальное обучение необходимо для группы неподготовленных пользователей, а потому все ограничивается минимальным текущим обучением. Участники пилотного проекта, имеющие начальное обучение, могут быть высококвалифицированными энтузиастами новой технологии, стремящимися использовать ее во что бы то ни стало. С другой стороны, разработчикам, которые в дальнейшем примут участие в проекте, может потребоваться более интенсивное и глубокое обучение, а также текущая поддержка в использовании средства.

В дополнение к этому следует отметить, что каждая категория сотрудников (например, администраторы средств, служба поддержки рабочих мест, интеграторы средств, служба сопровождения и разработчики приложений) нуждается в различном обучении.

Обучение не должно замыкаться только на пользователях CASE-средств, обучаться должны и те сотрудники, на деятельность которых так или иначе оказывает влияние использование CASE-средств.

При дальнейшем применении CASE-средств организация должна ориентироваться на обучение как сотрудников, вновь принятых на работу, так и специалистов, выполняющих проекты с использованием данных средств. Именно поэтому обучение должно стать неотъемлемой частью нормативных материалов, касающихся деятельности организации, которые предлагаются новым сотрудникам.

Чтобы определить, насколько эффективно новое CASE-средство повышает продуктивность и/или качество, организация должна опираться на некоторые базовые данные. К сожалению, лишь немногие организации в настоящее время накапливают данные для реализации программы текущей количественной оценки и усовершенствования процессов. Для доказательства эффективности CASE-средств и их возможностей улучшать продуктивность необходимы следующие базовые метрические данные:

использованное время;  
 время, выделенное персонально для конкретных специалистов;  
 размер, сложность и качество ПО;  
 удобство сопровождения.Еще до начала внедрения CASE-средств метрическая оценка должна начинаться с реальной оценки текущего состояния среды и поддерживать процедуры постоянного накопления данных.

Период, в течение которого выполняется количественная оценка воздействия, оказываемого внедрением CASE-средств, является весьма значимой величиной с точки зрения определения степени успешности перехода. Некоторые организации, успешно внедрившие в конечном счете CASE-средства, столкнулись с кратковременными негативными эффектами в начале процесса. Другие, успешно начав, недооценили долговременные затраты на сопровождение и обучение. Таким образом, чтобы преодолеть любые негативные эффекты на начальном этапе, а также смоделировать будущие долговременные затраты, наиболее приемлемый временной интервал для оценки степени успешности внедрения должен быть достаточно большим. С другой стороны, данный интервал должен соответствовать целям организации и ожидаемым результатам.

В конечном счете, опыт, полученный при внедрении CASE-средств, может отчасти изменить цели организации и ожидания, возлагаемые на CASE-средства. Например, организация может сделать вывод, что средства целесообразно использовать для большего или меньшего круга пользователей и процессов в цикле создания и сопровождения ПО. Такие изменения в ожиданиях зачастую дают положительные результаты, но могут и внести соответствующие коррективы в определение степени успешного внедрения CASE-средств.

Итогом данного этапа является внедрение CASE-средств в повседневную практику организации, при этом больше не требуется какого-либо специального планирования. Кроме того, поддержка CASE-средств включается в план текущей поддержки ПО в данной организации.

## 39. Функционально-ориентированное (структурное) проектирование программного обеспечения при помощи CASE-средств.

Основными идеями функционально-ориентированной CASE-тех­но­логии являются идеи структурного анализа и проектиро­вания информационных систем. Они заключаются в следующем:

1) декомпозиция всей системы на некоторое множество иерар­хически подчиненных функций;

2) представление всей информации в виде графической нота­ции. Систему всегда легче понять, если она изображена графи­чески.

В структурном анализе используются в основном три группы моделей, которые:

- иллюстрируют функции, выполняемые системой;

- описывают отношения между данными;

- описывают поведение системы во времени.

В качестве инструментальных средств структурного анализа и проектирования выступают следующие диаграммы:

– BFD (Bussiness Function Diagram) – диаграмма бизнес-функ­ций (функциональные спецификации);

– DFD (Data Flow Diagram) – диаграмма потоков данных;

– STD (State Transition Diagram) – диаграмма переходов состо­яний (матрицы перекрестных ссылок);

– ERD (Entity Relationship Diagram) – ER-модель данных пред­метной области (информационно-логические модели «сущ­ность–связь»);

– SSD (System Structure Diagram) – диаграмма структуры про­граммного приложения.

Все они содержат графические и текстовые средства моделирования. Графические средства обеспечивают удобство демонстрирования основных компонент модели.

Текстовые - служат для обеспечения точного определения компонентов модели и ее связей.

Диаграммы потоков данных (DFD-диаграммы) показывают внешние, по отношению к системе источники, и стоки (адресаты) данных, идентифицируют логические функции (процессы) и группы элементов данных (потоки), которые связывают одну функцию с другой. Они также идентифицируют хранилища (накопители) данных к которым осуществляется доступ.

Каждая логическая функция (процесс) может быть детализирована с помощью DFD-диаграмм нижнего уровня. Когда детализация перестает быть полезной, переходят к выражению логики функции при помощи спецификаций процесса.

Модель хранилища данных раскрывается с помощью диаграмм «сущность-связь» (ERD-диаграммы).

В случае необходимости учета реального времени, диаграммы потоков данных дополняются средствами описания зависящего от времени поведения системы. Поведение системы раскрывается с использованием диаграмм переходов состояний (STD).

Использование этих моделей позволяет построить логическую функциональную модель информационной системы, то есть подробное описание того, что должна делать система, освобожденное от рассмотрения возможных путей ее реализации.

## 40. Методология разработки программных систем msf (Microsoft Solutions Framework).

**Microsoft Solutions Framework** (**MSF**) — методология разработки программного обеспечения, предложенная корпорацией Microsoft. MSF опирается на практический опыт Microsoft и описывает управление людьми и рабочими процессами в процессе разработки решения.

MSF представляет собой согласованный набор концепций, моделей и правил.

Наиболее популярные прикладные варианты MSF, разработанные Microsoft:

* методика внедрения решений в области Управления проектами;
* методика управления IT-проектами на базе методологий MSF и Agile.

Важность прикладных вариантов MSF подчёркивает тот факт, что в «чистом варианте» саму методику MSF в своих IT-проектах компания Microsoft не использует. В проектах *Microsoft Consulting Services* используется гибридная методология MSF и Agile. Несмотря на внешние существенные различия прикладных вариантов MSF, разработанных экспертами Microsoft, общая база методов MSF для них остается общая и отражает общие методологические подходы к итеративному ведению проектов.

MOF призван обеспечить организации, создающие критически важные (англ. *mission-critical*) IT решения на базе продуктов и технологий Microsoft, техническим руководством по достижению их надёжности (англ. *reliability*), доступности (англ. *availability*), удобства сопровождения (англ. *supportability*) и управляемости (англ. *manageability*). MOF затрагивает вопросы, связанные с организацией персонала и процессов, технологиями и менеджментом в условиях сложных (англ. *complex*), распределённых (англ. *distributed*) и разнородных (англ. *heterogeneous*) IT-сред. MOF основан на лучших производственных методиках, собранных в *IT Infrastructure Library* (ITIL), составленных *Central Computer and Telecommunications Agency* — Агентством правительства Великобритании.

Создание бизнес-решения в рамках отведенных времени и бюджета требует наличия испытанной методологической основы. MSF предлагает проверенные методики для планирования, проектирования, разработки и внедрения успешных IT-решений. Благодаря своей гибкости, масштабируемости и отсутствию жестких инструкций MSF способен удовлетворить нужды организации или проектной группы любого размера. Методология MSF состоит из принципов, моделей и дисциплин по управлению персоналом, процессами, технологическими элементами и связанными со всеми этими факторами вопросами, характерными для большинства проектов.

MSF состоит из двух моделей и трех дисциплин. Они подробно описаны в 5 whitepapers. Начинать изучение MSF лучше с моделей, а затем перейти к дисциплинам.

MSF содержит:

* **модели**:
  + модель проектной группы
  + модель процессов
* **дисциплины**:
  + дисциплина *управление проектами*
  + дисциплина *управление рисками*
  + дисциплина *управление подготовкой*

**Модель проектной группы** MSF (MSF Team Model) описывает подход Майкрософт к организации работающего над проектом персонала и его деятельности в целях максимизации успешности проекта. Данная модель определяет ролевые кластеры, их области компетенции и зоны ответственности, а также рекомендации членам проектной группы, позволяющие им успешно осуществить свою миссию по воплощению проекта в жизнь.

Модель проектной группы MSF разрабатывалась в течение нескольких лет и возникла в результате осмысления недостатков пирамидальной, иерархической структуры традиционных проектных групп.

В соответствии с моделью MSF проектные группы строятся как небольшие многопрофильные команды, члены которых распределяют между собой ответственность и дополняют области компетенций друг друга. Это дает возможность четко сфокусировать внимание на нуждах проекта. Проектную группу объединяет единое видение проекта, стремление к воплощению его в жизнь, высокие требования к качеству работы и желание самосовершенствоваться.

Ниже описываются основные принципы, ключевые идеи и испытанные методики MSF в применении к модели проектной группы.

MSF включает в себя ряд **основных принципов**. Вот те из них, которые имеют отношение к успешной работе команды:

1. Распределение ответственности при фиксации отчетности
2. Наделяйте членов команды полномочиями
3. Концентрируйтесь на бизнес-приоритетах
4. Единое видение проекта
5. Проявляйте гибкость — будьте готовы к переменам
6. Поощряйте свободное общение

Успешное использование модели проектной группы MSF основывается на ряде **ключевых концепций** (key concepts):

1. Команда соратников
2. Сфокусированность на нуждах заказчика
3. Нацеленность на конечный результат
4. Установка на отсутствие дефектов
5. Стремление к самосовершенствованию
6. Заинтересованные команды работают эффективно

MSF основан на постулате о шести качественных целях, достижение которых определяет успешность проекта. Эти цели обуславливают модель проектной группы. В то время как за успех проекта ответственна вся команда, каждый из её ролевых кластеров, определяемых моделью, ассоциирован с одной из упомянутых шести целей и работает над её достижением.

В проектную группу входят такие **ролевые кластеры**:

* управление программой
* управление продуктом
* разработка
* тестирование
* управление релизом
* удовлетворение потребителя

Они ответственны за различные области компетенции (functional areas) и связанные с ними цели и задачи. Иногда ролевые кластеры называются просто ролями. Но в любом случае суть концепции остается той же — построить основу производственных отношений и связанную с ней модель команды такими, чтобы они были приспосабливаемыми (масштабируемыми) для удовлетворения нужд любого проекта.

Как уже было сказано выше, проектная группа по MSF состоит из шести ролевых кластеров, каждый из которых **отвечает за**:

* управление программой (program manager) — разработку архитектуры решения, административные службы;
* разработку (developer) — разработку приложений и инфраструктуры, технологические консультации;
* тестирование (QAE) — планирование, разработку тестов и отчетность по тестам;
* управление выпуском (release manager) — инфраструктуру, сопровождение, бизнес-процессы, выпуск готового продукта;
* удовлетворение заказчика (user experience) — обучение, эргономику, графический дизайн, техническую поддержку;
* управление продуктом (product manager) — бизнес-приоритеты, маркетинг, представительство интересов заказчика.

Наличие шести ролевых кластеров не означает, что количество членов команды должно быть кратным шести — один человек может совмещать несколько ролей и наоборот, ролевой кластер может состоять из нескольких лиц в зависимости от размера проекта, его сложности и профессиональных навыков, требуемых для реализации всех областей компетенции кластера. **Минимальный коллектив по MSF может состоять всего из трех человек**. Модель не требует назначения отдельного сотрудника на каждый ролевой кластер. Смысл состоит в том, что в команде должны быть представлены все шесть качественных целей. Обычно, выделение как минимум одного человека на каждый ролевой кластер обеспечивает полноценное внимание к интересам каждой из ролей, но это экономически оправданно не для всех проектов. Зачастую члены проектной группы могут объединять роли.

В малых проектных группах объединение ролей является необходимым. При этом должны соблюдаться два принципа:

1. Роль команды разработчиков не может быть объединена ни с какой другой ролью.
2. Избежание сочетания ролей, имеющих предопределенные конфликты интересов.

Как и в любой другой командной деятельности, подходящая комбинация ролей зависит от самих членов команды, их опыта и профессиональных навыков. На практике совмещение ролей встречается нередко. И если проектная группа производит его обдуманно и управляет связанными с таким объединением рисками, возникающие проблемы будут минимальными.

MSF не предоставляет конкретных рецептов управления проектами и не содержит объяснений разнообразных методов работы, которые применяют опытные менеджеры. Принципы MSF формируют такой **подход к управлению проектами**, при котором:

* ответственность за управление проектом распределенная между лидерами ролевых кластеров внутри команды — каждый член проектной группы отвечает за общий успех проекта и качество создаваемого продукта.
* профессиональные менеджеры выступают в качестве консультантов и наставников команды, а не выполняют функции контроля над ней — в эффективно работающей команде каждый её член имеет необходимые полномочия для выполнения своих обязанностей и уверен, что получит от коллег все необходимое.

Как следует из вышесказанного, одна из характерных особенностей MSF — отсутствие должности менеджера проекта!

Модель проектной группы MSF предлагает разбиение больших команд (более 10 человек) на малые **многопрофильные группы направлений** (feature teams). Эти малые коллективы работают параллельно, регулярно синхронизируя свои усилия. Кроме того, когда ролевому кластеру требуется много ресурсов, формируются т. н. функциональные группы (functional teams), которые затем объединяются в ролевые кластеры.

Использование ролевых кластеров не подразумевает и не навязывает никакой специальной структуры организации или обязательных должностей. Административный состав ролей может широко варьироваться в разных организациях и проектных группах. Чаще всего роли распределяются среди различных подразделений одной организации, но иногда часть их отводится сообществу потребителей или внешним по отношению к организации консультантам и партнерам. Ключевым моментом является четкое определение работников, ответственных за каждый ролевой кластер, их функций, ответственности и ожидаемого вклада в конечный результат.

Модель проектной группы MSF не обеспечивает успех сама по себе. Есть много других факторов, определяющих успех или неудачу проекта, но структура проектной группы, безусловно, вносит существенный вклад.

Подходящая структура команды является фундаментом успеха, и реализация модели MSF с использованием лежащих в её основе принципов поможет сделать проектные группы более эффективными и, как следствие, более успешными.

**Модель процессов MSF** (MSF process model) представляет общую методологию разработки и внедрения IT решений. Особенность этой модели состоит в том, что благодаря своей гибкости и отсутствию жестко навязываемых процедур она может быть применена при разработке весьма широкого круга IT проектов. Эта модель сочетает в себе свойства двух стандартных производственных моделей: каскадной (waterfall) и спиральной (spiral). Модель процессов в MSF 3.0 была дополнена ещё одним инновационным аспектом: она покрывает весь жизненный цикл создания решения, начиная с его отправной точки и заканчивая непосредственно внедрением. Такой подход помогает проектным группам сфокусировать своё внимание на бизнес-отдаче (business value) решения, поскольку эта отдача становится реальной лишь после завершения внедрения и начала использования продукта.

Процесс MSF ориентирован на «**вехи**» (milestones) — ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата. Этот результат может быть оценен и проанализирован, что подразумевает ответы на вопросы: «Пришла ли проектная группа к однозначному пониманию целей и рамок проекта?», «В достаточной ли степени готов план действий?», «Соответствует ли продукт утверждённой спецификации?», «Удовлетворяет ли решение нужды заказчика?» и т. д.

Модель процессов MSF учитывает постоянные изменения проектных требований. Она исходит из того, что разработка решения должна состоять из коротких циклов, создающих поступательное движение от простейших версий решения к его окончательному виду.

Модель процессов MSF тесно связана с базовыми принципами MSF, рассмотренными выше. Вообще говоря, **тремя особенностями модели процессов MSF** являются:

* Подход, основанный на фазах и вехах.
* Итеративный подход.
* Интегрированный подход к созданию и внедрению решений.

Модель процессов включает такие основные фазы процесса разработки:

* Выработка концепции (Envisioning)
* Планирование (Planning)
* Разработка (Developing)
* Стабилизация (Stabilizing)
* Внедрение (Deploying)

Кроме этого существует большое количество *промежуточных вех*, которые показывают достижение в ходе проекта определенного прогресса и расчленяют большие сегменты работы на меньшие, обозримые участки. Для каждой фазы модели процессов MSF определяет:

* что (какие артефакты) является результатом этой фазы
* над чем работает каждый из ролевых кластеров на этой фазе

В рамках MSF программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности. Затем к решению добавляются все новые и новые возможности. Такая стратегия именуется стратегией версионирования. Несмотря на то, что для малых проектов может быть достаточным выпуск одной версии, рекомендуется не упускать возможности создания для одного решения ряда версий. С созданием новых версий эволюционирует функциональность решения.

Итеративный подход к процессу разработки требует использования гибкого способа ведения документации. «Живые» документы (living documents) должны изменяться по мере эволюции проекта вместе с изменениями требований к конечному продукту. В рамках MSF предлагается ряд шаблонов стандартных документов, которые являются артефактами каждой стадии разработки продукта и могут быть использованы для планирования и контроля процесса разработки.

Решение не представляет бизнес-ценности, пока оно не внедрено. Именно по этой причине модель процессов MSF содержит весь жизненный цикл создания решения, включая его внедрение — вплоть до момента, когда решение начинает давать отдачу.

Управление рисками (risk management) — это одна из ключевых дисциплин Microsoft Solutions Framework® (MSF). MSF видит в изменениях и возникающей из-за них неопределенности неотъемлемые части жизненного цикла информационных технологий. Дисциплина управления рисками в MSF (MSF risk management discipline (недоступная ссылка)) отстаивает превентивный подход к работе с рисками в условиях такой неопределенности, непрерывное оценивание рисков и использование информации о рисках в рамках процесса принятия решений на протяжении всего жизненного цикла проекта. Данная дисциплина предлагает принципы, идеи и рекомендации, подкрепленные описанием пошагового процесса для успешного активного управления рисками. Этот процесс включает в себя выявление и анализ рисков; планирование и реализацию стратегий по их профилактике и смягчению возможных последствий; отслеживание состояния рисков и извлечение уроков из обретенного опыта. **Девиз** MSF — **мы не боремся с рисками — мы ими управляем**.

Как уже говорилось выше, в MSF нет роли «менеджер проекта». Деятельность по управлению проектом распределяется между лидерами групп и ролевым кластером «Управление программой».

Для лидеров групп и ролевого кластера «Управление программой» инструментом управления проектом, облегчающим создание планов и календарных графиков, является **WBS**. Иерархическая структура работ (Work Breakdown Structure — WBS) — это структуризация работ проекта, отражающая его основные результаты и определяющая его рамки. Работа, не описанная в WBS, находится вне границ проекта. В MSF создание WBS является коллективной деятельностью, в которую вовлекаются все ролевые кластеры. Каждая роль ответственна за предоставление детального описания собственной работы.

**Управление подготовкой** — это также одна из ключевых дисциплин Microsoft Solutions Framework (MSF). Она посвящена управлению знаниями, профессиональными умениями и способностями, необходимыми для планирования, создания и сопровождения успешных решений. Дисциплина управления подготовкой MSF описывает фундаментальные принципы MSF и дает рекомендации по применению превентивного подхода к управлению знаниями на протяжении всего жизненного цикла информационных технологий. Эта дисциплина также рассматривает планирование процесса управления подготовкой. Будучи подкрепленной испытанными практическими методиками, дисциплина управления подготовкой предоставляет проектным группам и отдельным специалистам базу для осуществления этого процесса.

Следует отметить, что MSF не навязывает использование других продуктов Microsoft. Например, для организации процесса производства ПО можно использовать MSF и при этом применять инструменты Borland, хотя будущая версия MSF 4.0 будет жестко привязана к Microsoft Team System — новому инструментальному средству Майкрософт для поддержки командной работы.

## 41. Современный стек технологий для разработки программного обеспечения ИС и АС.

Перед запуском каждого проекта в сфере IT проходит множество процессов планирования и принятия решений. Эта фаза планирования имеет решающее значение и может оказать значительное влияние на будущее проекта.

Выбор правильного стека технологий может повлиять на время разработки, стоимость, качество приложения, а также на масштабируемость, поэтому так важно принять правильное решение, даже если вам нужно потратить больше времени, анализируя все плюсы и минусы доступных решений.

В этой статье мы хотим объяснить вам, что на самом деле представляет собой стек технологий веб-приложения, и, кроме того, рассмотрим доступные технологии, их плюсы и минусы и попытаемся выбрать, какая технология подходит для конкретного случая. Кроме того, мы выбрали несколько важных факторов, которые следует учитывать перед принятием решения о стеке технологий.

Когда технологии выбраны, каждому внешнему проекту по-прежнему необходимо принять еще несколько решений, таких как структура вашего внешнего приложения, которые помогут вам поддерживать и масштабировать его.

### Что такое стек технологий?

Стек технологий - это комбинация языков программирования, фреймворков и программного обеспечения, используемых для создания приложения.

Каждое веб-приложение состоит из двух частей: клиентской и серверной. Клиентская сторона - это все, что пользователи могут видеть на экране. Важнейшими элементами технологического стека на стороне клиента являются:

1. HTML, который отвечает за отображение содержимого в браузере,
2. CSS стилизует контент,
3. Javascript отвечает за интерактивную часть веб-приложения.

Эти технологии можно использовать с полезными фреймворками, такими как Bootstrap или React.js.

Серверная часть приложения не видна конечному пользователю, она обеспечивает данные для клиентской части. Что касается серверной части, мы должны принять во внимание выбор:

1. язык бэкэнд-программирования, такой как Java или C #,
2. такие фреймворки, как Spring или .NET,
3. база данных, такая как PostgreSQL или MongoDB,
4. сервер, такой как Apache или Nginx, или выберите бессерверную архитектуру.

Когда вы планируете следующий технологический стек приложения, вы должны учитывать серверную и клиентскую стороны. В этой статье мы углубимся в стек технологий внешнего интерфейса и рассмотрим различные возможности разработки внешнего интерфейса.

### Обзор фреймворков Frontend

Современные фреймворки внешнего интерфейса могут управлять всеми тремя элементами пользовательского интерфейса; у них есть HTML-шаблоны, стили и интерактивные функции.

Прямо сейчас есть три наиболее распространенных фреймворка на выбор; это ReactJS, Angular и VueJS.

Чтобы иметь возможность выбрать любую из них, хорошо знать больше о каждой из технологий, поэтому мы пройдемся по каждой и опишем их с точки зрения стека внешних технологий и архитектуры внешнего интерфейса.

#### ReactJS

ReactJS - это не фреймворк; это UI-библиотека, созданная Facebook и поддерживаемая огромным сообществом. ReactJS подходит для менее сложных приложений и больше ориентирован на расширенный пользовательский интерфейс и повторно используемые компоненты на очень сложной логике внешнего интерфейса.

ReactJS имеет богатую экосистему, которая состоит из самого ReactJS и библиотеки React-DOM для управления объектами DOM. Далее React-Router, который отвечает за маршрутизацию. Кроме того, он поставляется с JSX, который является расширением синтаксиса для Javascript для создания шаблонов в ReactJS. Чтобы сделать разработку проще и быстрее, очень полезны инструменты разработчика React.

Кроме того, если вы планируете мобильное приложение для своего проекта, ReactJS имеет структуру для мобильной разработки под названием React Native, которая очень похожа на сам ReactJS.

С точки зрения компании, наиболее существенными преимуществами ReactJS являются относительно низкая стоимость разработки и короткие сроки разработки. Кроме того, найти разработчиков со знаниями ReactJS несложно, а технология присутствует на рынке уже давно, поддерживается большим создателем и огромным сообществом. Это означает, что ReactJS не исчезнет и не перестанет поддерживаться в ближайшее время.

Давайте быстро подведем итоги, когда выбор ReactJS для вашего следующего проекта будет хорошей идеей и почему.

1. Если приложение, которое вы планируете, имеет расширенный пользовательский интерфейс со множеством повторно используемых компонентов и множеством манипуляций с DOM, хорошо использовать ReactJS.
2. Если у вас сжатые сроки, ReactJS также будет хорошей идеей.
3. Если у вас средний бюджет и хотя бы один senior-разработчик с опытом планирования и создания приложений ReactJS.

Это не лучший выбор, если приложение имеет расширенную логику или не может нанять опытного разработчика, который смог бы спланировать рабочий процесс приложения.

ReactJS легко изучить, но он не требует от разработчиков писать качественный код, поэтому вам всегда нужен опытный человек в команде, который позаботится об этом.

#### Angular

Большой конкурент ReactJS - Angular. В отличие от первой описанной технологии, Angular - это фреймворк, и он хорош для сложных проектов с более продвинутой логикой.

Angular имеет встроенную надежную экосистему разработки, такую как маршрутизация или управление состоянием. У него также очень чистая архитектура, и все разбито на три разных файла (логика, шаблон и стили).

Еще одним преимуществом сайта Angular является то, что его легко интегрировать с бэкэндом MVC.

Angular также заставляет разработчиков планировать и разрабатывать приложения до того, как они начнут работать над ним, что продлевает разработку и повышает качество проекта.

С точки зрения компании и руководства Angular обеспечивает очень высокое качество разрабатываемого проекта, но увеличивает стоимость приложения, а время разработки немного больше. Angular был создан Google и поддерживается одной из крупнейших IT-компаний, поэтому не нужно беспокоиться о том, что наш код быстро устареет. Недостатком выбора Angular в качестве основной технологии проекта является то, что нелегко найти разработчиков, знающих эту технологию, а обучение - непростое.

Подведем итог и по Angular.

1. Если проект большой и сложный, со сложной логикой, но довольно простым пользовательским интерфейсом, Angular будет хорошим решением.
2. Если вы заботитесь о качестве кода и хотите, чтобы ваш проект имел надежную масштабируемость, Angular - ваш выбор.
3. Если у вас в команде есть Angular-разработчики, выбрать этот фреймворк - отличная идея.

Если приложение, которое вы планируете, больше ориентировано на пользовательский интерфейс и имеет много манипуляций с DOM, тогда было бы гораздо лучше выбрать что-то другое.

#### VueJS

Последней из самых популярных технологий, используемых в настоящее время во внешнем интерфейсе, является VueJS. Это самый младший из трех. VueJS, как и ReactJS, представляет собой библиотеку пользовательского интерфейса, представляющую собой смесь концепций ReactJS и Angular.

Vuex, который является менеджером состояний для VueJS, намного проще поддерживать, чем Redux.

VueJS, как и ReactJS, требует внешних свобод для создания полноценной экосистемы, но готовых решений гораздо меньше, что делает это немного рискованным.

Кроме того, у VueJS нет большой компании, и он не поддерживается огромным сообществом, поэтому неизвестно, как долго он будет поддерживаться.

С точки зрения руководства и компании VueJS - это малобюджетное решение, и разработчики могут легко изучить структуру, поэтому вам не нужен высококвалифицированный разработчик с многолетним опытом работы в этой технологии. Если вы хотите сократить время разработки, а проект небольшой, VueJS будет отличным решением.

Подведем итоги решения VueJS.

1. Если приложение, которое вы разрабатываете, представляет собой небольшой побочный проект или небольшой MVP, то VueJS - отличное решение.
2. Если у вас небольшой бюджет и только начинающие разработчики, также неплохо выбрать VueJS.
3. Если у проекта нет большой логики и больших требований к пользовательскому интерфейсу, а дедлайн близок, VueJS - хорошее решение.

### Javascript или Typescript?

Есть два решения при выборе основного языка программирования для внешнего интерфейса; это Javascript и Typescript, которые являются синтаксическим расширением Javascript.

В некоторых случаях требуется Typescript; например, когда вы решите создать свой проект с помощью Angular. Но VueJS и ReactJS не требуют использования Typescript, хотя могут иметь много преимуществ.

Во-первых, использование Typescript сокращает время разработки, поскольку сокращает количество ошибок, влияет на обслуживание приложения и упрощает его. Кроме того, эту технологию поддерживают все современные фреймворки. Кроме того, с помощью Typescript может быть проще разработать интерфейс, если бэкэнд еще не готов, но мы знаем, какой тип данных мы получим.

Единственным недостатком Typescript является то, что его не очень просто настроить с помощью ReactJS или VueJS.

Хорошая новость заключается в том, что Typescript создан Microsoft, поэтому очень мала вероятность, что он будет забыт, и вам придется переделывать приложение.

Подводя итог, можно сказать, что выбор Typescript может упростить и ускорить процесс разработки и помочь избежать частой отладки, что может быть очень важно, особенно до дедлайна.

### CSS, Less или Sass?

Хотя современные фреймворки внешнего интерфейса могут предоставить вам все элементы, важные в стеке технологий вашего внешнего проекта, вы решаете, как вы хотели бы писать стили в приложении.

Написание стилей на чистом CSS не очень удобно в больших проектах. Наличие огромных файлов CSS может превратить разработку любого проекта в кошмар, затягивая процесс.

Чтобы упростить задачу, неплохо решить использовать какой-либо препроцессор, например Less или Sass. Их очень легко реализовать в проекте, использующем Webpack, и код становится намного чище.

И Less, и Sass позволяют нам использовать миксины, наследование или переменные, поэтому, если необходимо срочно изменить брендинг и цвета во всем приложении, это больше не проблема.

Единственная разница между Sass и Less заключается в том, что Sass написан на Ruby, поэтому он должен быть установлен на вашем компьютере, а Less написан на Javascript, поэтому для его запуска требуется установленный Node.js.

С точки зрения компании, самым большим преимуществом использования препроцессоров является экономия времени, и это позволяет нам избежать трудностей с поддержкой приложения, особенно когда оно будет расти.

### Что следует учитывать при выборе технологического стека для проекта?

На этом этапе мы хотели бы остановиться на некоторых факторах, которые должны быть важны для каждого, кто принимает решение о техническом стеке интерфейсных приложений.

**Размер приложения**

Первый фактор, который следует учитывать при выборе технического стека, вероятно, довольно очевиден, потому что самым важным является размер приложения, потому что с размером проекта растет и сложность.

**Время разработки**

Еще один важный фактор при выборе технологического стека для проекта - это время вывода приложения на рынок.

Если времени мало, вам нужно выбрать технологию, которая не требует длительного планирования, разработка не сложна, и вы можете обратиться к разработчикам, которые знают эту технологию.

В случае длительного вывода на рынок вы можете рассмотреть несколько факторов, чтобы сделать правильный выбор.

**Безопасность**

Учитывая, что вы тратите много денег на приложение, вам нужно, чтобы оно было безопасным. Тем более, что количество кибератак продолжает расти, нужно серьезно отнестись к этому, чтобы позаботиться о безопасности своих данных.

Большинство популярных фреймворков внешнего интерфейса содержат некоторые рекомендации по безопасности. Вам нужно позаботиться о том, чтобы следовать всем этим рекомендациям в процессе разработки.

**Стоимость**

Для многих компаний это один из самых важных факторов. Большинство фреймворков и инструментов являются бесплатными и имеют открытый исходный код, но все же вам нужно нанять разработчиков, поэтому вам нужно рассчитать, сколько разработчиков вам понадобится, чтобы завершить приложение за определенный период времени с помощью выбранной технологии.

Помимо стоимости разработки, вам необходимо принять во внимание расходы на обслуживание и серверные расходы. Если вы хотите сократить расходы на обслуживание, возможно, стоит подумать о бессерверной архитектуре для вашего приложения.

**Масштабируемость и поддержка**

Также важно думать о будущем приложения. Если вы хотите, чтобы оно увеличивалось, оно должно быть легко масштабируемым и поддерживаемым. Добиться этого легко, просто нужно выбрать хорошую структуру проекта и позаботиться об использовании надежных технологий.

В зависимости от типа проекта вам может быть необходимо обратить внимание на еще несколько факторов, например производительность. Не забудьте переосмыслить каждый аспект приложения, чтобы вы могли спланировать его до его запуска, а внедрение любых изменений архитектуры становится действительно дорогостоящим.

### Вывод

Выбор правильного технического стека важен для успеха приложения, которое вы планируете создать, и для плавного завершения процесса разработки до дедлайна.

Чтобы сделать правильный выбор среди всех доступных на рынке интерфейсных технологий, вы должны внимательно изучить наиболее важные функции приложения и посмотреть на факторы, которые могут так или иначе повлиять на решение.

Разным приложениям нужен другой подход, поэтому стоит помнить, что если предыдущий проект был выполнен с помощью Angular и был успешным, текущему может потребоваться другая технология.

Также стоит взглянуть на разницу между Javascript и Typescript и подумать, какой из них будет лучшим выбором для приложения.

## 42. Нативные средства, библиотеки и инструменты разработки программного и аппаратного обеспечения ИС и АС.

**Нативные приложения** (англ. *native app(lication)s*) — это прикладные программы, которые были разработаны для использования на определённой платформе или на определённом устройстве.

Одно из преимуществ нативных приложений — то, что они оптимизированы под конкретные операционные системы, поэтому они могут работать корректно и быстро. Также они имеют доступ к аппаратной части устройств, то есть могут использовать в своём функционале камеру смартфона, микрофон, акселерометр, геолокацию, адресную книгу, плеер и т.д.

В **iOS** используются Objective-C, Swift, C и С++ и среда программирования XCode. В **Android:** Android Studio язык Java и Kotlin

XCode и Android Studio содержат пакет утилит и инструментов, необходимых для написания кода, отладки и тестирования приложений. Они заточены под конкретную платформу, существенно упрощают процесс разработки, делают его более гибким и быстрым.

Чтобы работать с C/C++ кодом в Android используют **NDK** (native development kit).

## **Преимущества и недостатки нативного подхода**

### **Плюсы нативных приложений**

1. **Высокая производительность**Поскольку технологии, используемые при разработке платформозависимых приложений, напрямую связаны с этой платформой, собственный нативный код имеет прямой доступ ко всем функциям операционной системы.  
   Это, более простое взаимодействие приложения с собственными функциями мобильных устройств, повышает общую производительность приложения, особенно при представлении графического или мультимедийного контента.  
   Следовательно, создание нагруженных приложений с использованием нативного кода может снизить время отклика, вероятность сбоев и зависаний.
2. **Максимальное использование возможностей платформы**Нативные приложения задумываются и разрабатываются, чтобы решать конкретные задачи на конкретной платформе. Это приводит к лучшему соответствию возможностей приложений аппаратным возможностям устройств, включая Bluetooth, NFC, камеру, GPS и т. д.  
   Эта соответствие необходимо, когда приложение должно использовать такие данные, как физическое и географическое местоположение и др.
3. **Лучший пользовательский интерфейс**Поскольку нативные приложения напрямую интегрируются с мобильной операционной системой, воспринимая и используя все доступные возможности «железа», пользователи могут перемещаться по привычному интерфейсу без особых хлопот, что приводит к положительному пользовательскому опыту (UX) и стабильному повторному использованию. К примеру сейчас, при большом количестве разнообразных вариантов разрешений экранов смартфонов очень важно иметь приложение, оптимизированное под такой экран, чтобы пользователю было удобно этим приложением пользоваться.
4. **Лучшее позиционирование в магазинах приложений**Качество пользовательского опыта является важным рейтинговым показателем в магазинах приложений. Если приложение имеет высокую оценку пользовательского опыта, оно будет более высоко оценено магазином приложений, что ведет к большему числу рекомендаций для разной аудитории и увеличению доходов от приложения, соответственно.  
   Есть предположение, что в магазинах приложений сами механизмы ранжирования будут лучше представлять приложения именно нативные для платформы, из-за их заведомо более высокой производительности и простоты использования.

### **Минусы разработки нативных приложений**

1. **Дороговизна и затраты времени на разработку**Без сомнения, создание отдельных приложений сразу под каждую из нескольких операционных систем может значительно продлить процесс разработки. Один и тот же программный код не может быть развернут на разных платформах, и программистам потребуется больше времени для преобразования и перезаписи кода, что увеличивает затраты и время разработки.  
   Если компания хочет для каждой из платформ создавать отдельные приложения, она может оказаться вынуждена нанять дополнительных программистов-специалистов. Например, один разработчик будет сосредоточен на разработке приложений для iOS, а другой - на разработке приложений для Android, что еще больше увеличивает расходы.  
   **Несовместимость с другой мобильной операционной системой**Вам придется заранее согласиться с несовместимостью вашего приложения с другими ОС. Когда разрабатывается приложение под конкретную ОС, его разработчики используют язык, специфичный только для этой операционной системы: например, Objective-C или Swift - для iOS, для различных мобильных устройств на базе Android - Kotlin и Java. В этом контексте нативное приложение, которое изначально написано для iOs, не будет совместимо с устройствами на базе Android и наоборот.  
   **Упущенные возможности**Разработка приложений, ориентированных только на одну платформу, может привести к упущенным возможностям. Особенно если другие платформы заранее не принимаются во внимание. Заведомое сокращение целевого рынка может привести к потере дохода.

### **Выводы**

Для несложных корпоративных проектов, которые нужно быстро запустить, разработка кроссплатформенных приложений – наилучший вариант. При небольшом бюджете можно добиться хорошего результата.

Ну а для долгосрочных сложных проектов **нативная разработка** пока остается единственным правильным решением.

## 43. Кроссплатформенные средства, библиотеки и инструменты разработки программного и аппаратного обеспечения ИС и АС.

**Кроссплатформенность** (*межплатформенность*) — способность программного обеспечения работать с несколькими аппаратными платформами или операционными системами. Обеспечивается благодаря использованию высокоуровневых языков программирования, сред разработки и выполнения, поддерживающих условную компиляцию, компоновку и выполнение кода для различных платформ. Типичным примером является программное обеспечение, предназначенное для работы в операционных системах Linux и Windows одновременно.

## Кроссплатформенные языки программирования

Кроссплатформенными можно назвать большинство современных высокоуровневых языков программирования. Например, Си, С++, Free Pascal, FreeBASIC, PureBasic — кроссплатформенные языки на уровне компиляции, то есть для этих языков есть компиляторы под различные платформы. Это позволяет — при надлежащем качестве кода — не переписывать основной движок программы, меняются только особые системозависимые части.

Не менее важны для кроссплатформенности стандартизованные библиотеки среды выполнения. В частности, стандартом стала библиотека языка Си (POSIX). Из крупных кроссплатформенных библиотек — Qt, GTK+, FLTK, STL, Boost, OpenGL, SDL, OpenAL, OpenCL.

Существуют кросскомпиляторы — компиляторы, генерирующие исполняемый код для платформы, отличной от той, на которой запущен сам компилятор.

## Кроссплатформенные среды исполнения

PHP, Perl, Python, Tcl и Ruby — кроссплатформенные интерпретируемые языки, их интерпретаторы существуют для многих платформ.

Среды исполнения ActionScript Virtual Machine, Java Virtual Machine и .NET также кроссплатформенны, однако на их вход подаётся не исходный текст, а промежуточный код. Поэтому программы, написанные на ActionScript, Java и C#, можно запускать под разными операционными системами без предварительной перекомпиляции.

## Кроссплатформенный пользовательский интерфейс

На разных операционных системах — независимо от того, как технически достигнута работа в них — стандартные элементы интерфейса имеют разные размеры. Поэтому простое жёсткое позиционирование элементов интерфейса невозможно — под другой операционной системой они могут «налезать» друг на друга. Существует несколько подходов:

1. Единый стиль, общий для всех операционных систем, программы выглядят одинаково под всеми системами. Так работают интерфейсные библиотеки Java наподобие Swing.
   * Плюс: можно жёстко расставлять элементы управления на манер Delphi; оригинальный стиль.
   * Минус: системе приходится иметь свои экранные шрифты; стиль отличается от стиля ОС.
2. Самоадаптирующийся интерфейс, подстраивающий сетку под реальные размеры элементов управления. Типичные примеры — Qt, wxWidgets, XUL.
   * Плюс: стандартный стиль операционной системы, очень быстрый и «скинующийся» под Windows XP, Vista и 7; некоторая автоматизация локализации.
   * Минус: чтобы собрать самоадаптирующуюся сетку, требуется квалифицированный программист; затруднена плотная компоновка.
3. Гибридный подход реализован в GTK+.
   * Плюс: шрифты можно брать из системы, а не «тащить» свои; некоторая автоматизация локализации.
   * Минус: берёт все недостатки от первых двух подходов. Стиль отличается от стиля операционной системы; затруднена плотная компоновка.

В любом случае, под другими операционными системами требуется хотя бы минимальное тестирование, так как возможны ошибки компоновки.

Современные операционные системы также часто являются кроссплатформенными. Например, операционные системы с открытым исходным кодом, например, NetBSD, Linux, FreeBSD, AROS могут работать на нескольких различных платформах, чаще всего это x86, m68k, PowerPC, Alpha, AMD64, SPARC. Первый выпуск Microsoft Windows NT 4 вышедший в 1996 году поддерживал четыре платформы (x86, Alpha, MIPS и PowerPC), в дальнейших версиях Windows NT осталась только поддержка платформы х86.Современная Microsoft Windows может работать как на платформе Intel x86, так и на Intel Itanium (точнее, для Itanium есть только версии Windows 2000/XP, Windows 2003 и Windows 2008, после чего поддержка была свёрнута). Операционная система NetBSD является самой переносимой[1], она портирована на большинство существующих платформ.

Ряд IDE, в том числе Free Pascal, Lazarus, Qt Creator, работают на разных операционных системах: Linux, Windows и других

5 популярных фреймворков для кросс-платформенной разработки:

* React Native
* Flutter
* Ionic
* Xamarin
* PhoneGap

## React Native

Инструмент от Facebook. Его цель — сделать кроссплатформенные приложения такими же производительными, как нативные.

### Язык программирования

JavaScript + библиотека React.

### Кто использует

React Native достаточно популярен, поскольку его уже применяют технологические гиганты. Среди них Instagram, Facebook, Walmart, Tesla, Pinterest, UberEats и другие.

### IDE и написание кода

С момента запуска React Native прошло около 5 лет, поэтому его поддерживают почти все ведущие IDE. Изучать React Native и писать код на нём довольно просто благодаря использованию JavaScript (разумеется, если вы знаете JavaScript).

### Архитектура и исполнение кода

«Learn once, write anywhere», что можно трактовать как «научись один раз, используй везде» — главный принцип React Native, который подразумевает применение одного и того же кода для разных платформ. Также в Native есть функция Hot Reloading, позволяющая добавлять новый код и вносить правки прямо во время выполнения — это очень полезно, когда вы настраиваете пользовательский интерфейс. Среда поставляется с большим набором готовых компонентов, однако они не всегда адаптируются под разные платформы, что требует дополнительных корректировок в коде. Благодаря обширной поддержке сообщества также есть богатый выбор сторонних библиотек.

### Производительность

Так как React Native нацелен на результат, сопоставимый с нативной разработкой, в погоне за производительностью чаще всего отдают предпочтение именно этому фреймворку. Native также позволяет разработчикам использовать кастомные модули на языках для нативной разработки, но их придётся писать отдельно для каждой платформы.

## Flutter

Flutter — детище Google, уже завоевавшее хорошую репутацию в кроссплатформенной разработке. Его принцип — создание приложений с единой кодовой базой для мобильных платформ, веба и десктопа.

### Язык программирования

Dart — объектно-ориентированный язык, разработанный Google.

### Кто использует

Flutter появился на рынке не так давно, но его популярность выросла за очень короткое время. Приложения на нём можно увидеть у Alibaba, Hamilton Musical, Greentea, Google Ads.

### IDE и написание кода

Flutter поддерживается Android Studio / IntelliJ и Visual Studio Code. Что касается программирования на Dart — если вы знаете C++ или Java, вам будет проще его освоить. Однако изучение любого нового языка требует времени.

### Архитектура и исполнение кода

Flutter использует один и тот же код для всех платформ. На нём легко создавать красивые интерфейсы. Но если вам нужны разные стили для разных ОС, придётся немного поработать, поскольку автоматическая адаптация для этих целей не предусмотрена. Это связано с тем, что вместо нативных компонентов Flutter применяет свой графический движок. Однако он не отстаёт от React Native и также предлагает функцию Hot Reloading (добавление нового кода без повторной сборки) и большой набор готовых виджетов. Зато с Flutter можно выпускать приложения для разных версий Android и iOS без дополнительных телодвижений: программы спокойно запускаются даже на таких старых версиях, как Android Jelly Bean и iOS 8.

### Производительность

При прочих равных можно сказать, что Flutter превосходит конкурентов и демонстрирует самую высокую производительность благодаря современному языку Dart и собственному движку рендеринга.

## Ionic

С Ionic можно создавать кроссплатформенные гибридные приложения. Он тесно взаимодействует с фреймворком Apache Cordova, который преобразовывает веб-приложения в мобильные программы.

### Язык программирования

JavaScript + HTML, CSS.

### Кто использует

Из наиболее известных: MarketWatch, Pacifica, Sworkit, Nationwide.

### IDE и написание кода

Ionic завоевал признание среди разработчиков мобильных приложений, потому что с ним просто работать. Фреймворк построен на ECMAScript 6 и TypeScript, поэтому его можно использовать в любой IDE, поддерживающей эти языки, например в Visual Studio Code, Atom или Angular IDE. Кстати, если вы уже знакомы с Angular, React или Vue, то с освоением Ionic не возникнет особых трудностей.

### Архитектура и исполнение кода

Ionic, как и React Native и Flutter, предлагает концепцию единого кода для разных платформ, но на новом уровне. Все его компоненты автоматически адаптируются к платформе, на которой запускается приложение — а значит, разработка становится быстрее. Также с Ionic вы можете свободно использовать JavaScript, Angular, React или Vue.

### Производительность

А вот здесь Ionic проигрывает и сильно отстаёт от React Native и Flutter, поскольку для визуализации приложений он использует веб-технологии и совсем не применяет нативные компоненты. Такой подход значительно снижает скорость.

Но со стороны разработки есть и плюсы: Ionic позволяет проводить быстрое тестирование, которое можно запустить прямо в браузере.

## Xamarin

Xamarin — платформа для создания мобильных приложений от Microsoft, которая также поддерживает разработку для Windows.

### Язык программирования

C# + .NET.

### Кто использует

На Xamarin сделаны приложения Olo, The World Bank, Storyo и другие.

### IDE и написание кода

В качестве IDE можно использовать, например, Visual Studio 2019 или Rider. C# достаточно распространён, поэтому с написанием кода и освоением Xamarin проблем возникать не должно.

### Архитектура и исполнение кода

У Xamarin есть два основных инструмента: Xamarin.Android/iOS и Xamarin.Forms. По части кроссплатформенной разработки Xamarin предлагает использовать единый API Xamarin.Essentials.

Xamarin.Android и Xamarin.iOS наделяют приложение теми же возможностями и интерфейсом, которые есть у нативных решений. В случае Xamarin.iOS программа компилируется непосредственно в машинный код (AOT-компиляция), тогда как в Xamarin.Android сначала происходит компиляция в байт-код, который затем интерпретируется виртуальной машиной (JIT-компиляция).

Если же нужно ускорить процесс написания кода, лучше использовать Xamarin.Forms — более простой инструмент, в котором почти все элементы полностью совместимы с любыми платформами.

### Производительность

Производительность Xamarin также считается близкой к нативной, но зависит от того, используете вы Xamarin.Android, Xamarin.iOS или Xamarin.Forms. У Xamarin.Android/iOS хорошая оптимизация благодаря нативным компонентам. Xamarin.Forms же основан на 100% совместном использовании кода, что в целом снижает его производительность по сравнению с Xamarin.Android/iOS.

## PhoneGap

Как и Ionic, PhoneGap позволяет использовать веб-технологии в мобильной разработке. Он является дистрибутивом Apache Cordova.

### Язык программирования

JavaScript + HTML, CSS.

### Кто использует

Среди примеров приложений на PhoneGap: Logitech Squeezebox Controller, Localeur, Untappd, HealthTap.

### IDE и написание кода

Для более комфортной кроссплатформенной разработки и тестирования можно использовать Adobe Dreamweaver (версии 5.5 и выше), MyEclipse 2013, Tiggzi, ApplicationCraft. Разработка на JavaScript не должна вызывать затруднений, особенно если раньше вы уже писали на нём.

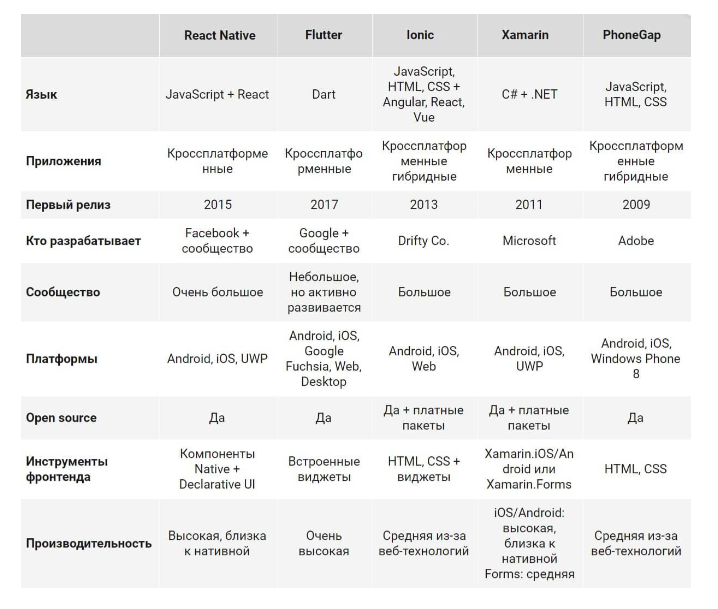
### Архитектура и исполнение кода

Приложение PhoneGap, по сути, представляет собой набор HTML-страниц, обёрнутых в нативную оболочку. Страницы хранятся в локальном каталоге или в облаке, а во время запуска на смартфоне они получают доступ к функциям устройства через плагины. Это делает приложения PhoneGap довольно лёгкими, но они выглядят менее естественно, а качество пользовательского интерфейса будет в большей степени зависеть от веб-представления конкретной ОС.

### Производительность

PhoneGap отличается невысокой производительностью по сравнению с нативными инструментами — и в этом снова виноваты веб-технологии.

## Сравнительная таблица



## 44. Десктопные приложения ИС и АС (ASS).

**Десктопное приложение** — программа, которая устанавливается на компьютер пользователя и работает под управлением операционной системы. Такие приложения являются высокопроизводительными, могут работать напрямую с принтерами, сканерами, факсами и прочей техникой.

Десктопные приложения могут быть также и многопользовательскими. Например, редактор файлов который в зависимости от логина и пароля, введенных при запуске, будет давать доступ к различным файлам. И программа, и файлы находятся на одном компьютере, просто производится локальное разграничение доступа для разных пользователей.

**Достоинства** настольного приложения:

- Работа как в онлайн так и в офлайн.

- Автономная работа как с подключением так и без подключения к интернету.

- Быстрый запуск.

- Всегда запускается с загрузкой данных из памяти устройства, на котором установлено, не требует каждый раз загрузки параметров из сети.

- Отзывчивый пользовательский интерфейс.

- Доступ к функциям операционной системы и графическим библиотекам, таким как GDI, DirectX, OpenGL, позволяет создавать приложения с богатым интерфейсом без торможений и зависаний.

- Быстрая работа.

- Максимальная эффективность работы за счет возможности многопроцессорных систем, возможности непосредственной работы с памятью, с файловой системой, локальными базами данных.

- Настольному приложению доступны все устройства, подключаемые к компьютеру, либо через фирменные драйвера, либо с использованием низкого уровня взаимодействия — COM, LPT, Ethernet и т.д.

- Обеспечивают больший уровень конфиденциальности.

**Недостатки**:

- Использование приложения ограничено наличием его на конкретной машине, нельзя (за исключением использование средств удаленного доступа) использовать приложение на машине, на которой оно не установлено.

- Занимают место на жестком диске.

- Необходима установка для каждого ПК.

- Необходимость ручного обновления.

## 45. Веб-технологии в ИС и АС.

В ИС и АС веб-технологии используются для передачи информации. Помимо десктопных приложений, существуют веб-приложения, для работы которых требуется наличие веб-браузера.

Файлы веб-приложений хранятся не на компьютере, а на удаленном сервере. Веб-браузер позволяет получить доступ к приложению и его содержимому, а также запускает все сценарии, ответственные за его функции. Что отличает простую статическую веб-страницу от веб-приложения, так это интерактивность. Они часто позволяют создавать, редактировать или манипулировать данными и контентом.

Примерами веб-приложений могут выступать онлайн редакторы документов, изображений, почтовые клиенты и так далее.

**Достоинства** веб-приложений:

- Отсутствует необходимость в установке. Веб-приложения можно запускать, просто перейдя по нужному URL-адресу. Это позволяет быстро и легко начать работу с приложением в любое удобное для вас время. На вашем жестком диске нет больших файлов, занимающих место для хранения данных, и вы можете получить доступ к ним с любого устройства.

- Автоматические обновления

- Кроссплатформенность. Для подавляющего большинства веб-приложений единственным обязательным условием является доступ в интернет. Они не зависят от аппаратных и системных спецификаций для запуска. В результате вы можете запускать их с любого устройства или платформы, где есть веб-браузер. Поскольку компоненты, отвечающие за функциональность приложения, находятся на сервере, не имеет значения, запускаете ли вы его с Windows, Mac, Linux или чего-то еще.

- Меньшее потребление ресурсов устройства

**Недостатки**:

- Угрозы безопасности. Удаленные веб-сервера являются потенциальными мишенями для злоумышленников, поэтому в процессе хранения данных может возникнуть вопрос, связанный с безопасностью.

- Зависимость от доступа в интернет. Многие современные веб-приложения используют функцию кэширования, что позволяет продолжить использование без интернета, однако, функциональность веб-приложения без интернета может быть ограничена.

- Меньшая производительность. Веб-приложения не имеют доступа к вычислительной мощности машины, на которой они запускаются.

## 46. Мобильные приложения в ИС и АС.

**Мобильные приложения** - компьютерная программа, созданная специально для использования в мобильном телефоне, смартфоне или коммуникаторе, которая предназначена для выполнения той или иной задачи.

Выделяют основных три вида:

- Веб-приложение или мобильный сайт;

- Гибридное приложение;

- Нативное приложение.

**Веб-приложение** представляет собой обычный сайт с расширенным функционалом.

Бесспорными плюсами являются возможность использования на любой мобильной платформе, низкая стоимость и быстрые сроки разработки.

**Гибридное приложение** является чем-то средним между веб-приложениями и нативными. Такие приложения скачиваются в официальных магазинах и имеют ограниченный доступ к аппаратной части мобильных устройств. Например, можно настроить push-уведомления. Но контент остается кроссплатформенным и размещенным на сервере.

Основными недостатками гибридного приложения являются:

- внешний вид, который не будет изменяться в зависимости от мобильной платформы;

- ограниченность объема хранения информации, дополнительную информацию приложение будет докачивать из интернета;

- сложный процесс оптимизации под разные размеры экранов;

- разработка некоторых компонентов (дополнительные строки, выезжающее меню и т.д.) каждый раз с нуля.

**Нативное приложение** – самое трудоемкое, но больше все подходит для каждой операционной мобильной системе. Разработка осуществляется для каждой платформы (Windows Phone, IOS, Android) отдельно. Это достаточно сложно и сроки будут более растянутыми, по сравнению с другими видами приложений. Соответственно цена будет самой дорогой. Это и есть главные два минуса нативного приложения.

В остальном плюсы очевидные:

- приложение будет работать в любом месте независимо от интернет-соединения;

- быстрая скорость и корректность работы;

- доступ к аппаратной части мобильного устройства (камера, геолокация, микрофон, адресная книга и т.д.);

- экономия батареи и памяти мобильного устройства.

## 47. Программные и аппаратные средства защиты информации в ИС и АС.

**Программно-аппаратные средства защиты** — это способы контроля оборудования и программных средств от взлома, перехвата информации, несанкционированного подключения третьих лиц. Программные и технические средства защиты информации необходимы там, где утечка данных и ценной информации влечет за собой серьезные финансовые, репутационные, производственные риски для компании.

Средства защиты можно разбить на следующие группы:

- Идентификация и аутентификация;

- Управление доступом;

- Протоколирование и аудит;

- Криптография;

- Экранирование.

**1) Идентификация** — это процедура распознавания субъекта по его идентификатору (проще говоря, это определение имени, логина или номера).

**Аутентификация** – это процедура проверки подлинности (пользователя проверяют с помощью пароля, письмо проверяют по электронной подписи и т.д.)

**2) Управление доступом** - ограниченный доступ к информации, компьютерам, сетям, приложениям, системным ресурсам, файлам и программам. В основе управления доступом лежит идентификация и аутентификация. Задача управления доступом состоит в том, чтобы для каждой пары "субъект-объект" определить множество допустимых операций и контролировать выполнение установленного порядка.

3**) Протоколирование** — сбор и накопление информации о событиях, происходящих в информационной сфере.

**Аудит** — это анализ накопленной информации, проводимый в реальном времени или периодически.

Данная система защиты выполняет важные задачи:

- составляет отчет обо всех пользователях и администраторах;

- выявляет слабые места в защите сервера, оценивает размер повреждений и возвращает к нормальной работе;

- предоставляет информацию для анализа и выявления проблем.

Характерной особенностью протоколирования и аудита является зависимость от других средств защиты информации. Идентификация и аутентификация служат началом для составления отчета о пользователях, управление доступом защищает конфиденциальность и целостность зарегистрированной информации.

4) **Криптографическая защита информации** — это механизм защиты с помощью шифрования данных, в результате которого их содержание становится недоступным без предъявления ключа криптограммы и обратного преобразования.

5) **Экран** — это средство разграничения доступа клиентов из одного множества информационных систем к серверам из другого множества посредством контроля информационных потоков между двумя множествами систем. Контроль потоков состоит в их фильтрации и выполнении некоторых преобразований.

Экран можно представить как последовательность фильтров. Каждый из фильтров, проанализировав данные, может пропустить или не пропустить их, преобразовать, передать часть данных на следующий фильтр или обработать данные от имени адресата и возвратить результат отправителю.

Главной функцией экранирования является обеспечение безопасности внутренней (защищаемой) сети и полный контроль над внешними подключениями и сеансами связи.

## 48. Технологии хранения данных ИС и АС.

Под хранением обычно понимают запись данных на некоторые накопители данных, с целью их (данных) дальнейшего использования.

Говоря о технологиях хранения, невозможно обойти вниманием термин RAID. Redundant array of independent disks — избыточный массив независимых дисков — это технология виртуализации данных, которая объединяет несколько дисков в логический элемент для повышения производительности. В зависимости от выбранного типа RAID, технологии хранения делятся на два класса:

**С использованием аппаратного RAID**. Более дорогое и не всегда оправданное решение, связанное с покупкой дополнительного компьютерного «железа» с собственной памятью и выделенным процессором. Аппаратный RAID требуется при наличии в системе как минимум четырех и более накопителей.

**С использованием программного RAID**. В этой технологии используются контроллеры на материнской плате, которые не имеют своей памяти и выделенного процессора. Они используют от 2-5% ресурсов центрального процессора сервера. Не менее надежны, чем аппаратные решения, используются в небольших системах. Оригинал статьи: https://www.kp.ru/guide/sistemy-khranenija-dannykh.html

- **DAS**. Накопители ставятся непосредственно в сервер для получения дополнительного пространства со сравнительно быстрым доступом. Самый простой и недорогой вариант.

- **NAS**. Хранилище, подключаемое по сети. Отличается гибкостью и централизованным управлением, однако скорость доступа ограничена скоростью сети.

- **SAN**. Хранилище, подключаемое через оптико-волоконный кабель. Сочетает в себе все плюсы NAS с высокой скоростью доступа.

**- Unified storage.** Универсальные системы, позволяющие совмещать в себе как функции NAS так и SAN. Чаще всего по реализации это SAN, в которой есть возможность активировать файловый доступ к дисковому пространству.

**- SDS.** Software-defined storage — программно определяемое хранилище данных, основанное на DAS, при котором дисковые подсистемы нескольких серверов логически объединяются между собой в кластер, который дает своим клиентам доступ к общему дисковому пространству.

И вообще дофига расписывать, если что, то брал отсюда: https://habr.com/ru/company/southbridge/blog/517502/

Оглавление

[1. Сфера применения современных информационных и автоматизированных систем. 1](#_Toc93159892)

[2. Применение современного программного и аппаратного обеспечения информационных систем для решения прикладных задач. 2](#_Toc93159893)

[3. Решение задач моделирования при помощи информационных систем. 4](#_Toc93159894)

[4. Решение задач интеллектуальной обработки данных при помощи информационных и автоматизированных систем. 7](#_Toc93159895)

[5. Методы и инструменты сопряжение программного и аппаратного обеспечения ИС и АВ. 11](#_Toc93159896)

[6. Методы комплексирование программного и аппаратного обеспечения ИС и АВ. 15](#_Toc93159897)

[7. Методики проектирования программного обеспечения ИС и АС. 17](#_Toc93159898)

[8. Методики проектирования аппаратного обеспечения ИС и АС. 19](#_Toc93159899)

[9. Фазы жизненного цикла программного обеспечения ИС и АС. 20](#_Toc93159900)

[10. Фазы жизненного цикла аппаратного обеспечения ИС и АС. 22](#_Toc93159901)

[ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНЫХ СРЕДСТВ ВИКИ (СЛЕГКА ВИДОИЗМЕНИТЬ) 22](#_Toc93159902)

[11. Методы обеспечения качества программного обеспечения ИС и АС. 22](#_Toc93159903)

[12. Методы обеспечения качества аппаратного обеспечения ИС и АС. 25](#_Toc93159904)

[13. Понятие технического задания на разработку ИС и АС. 28](#_Toc93159905)

[14. Базовые структуры и архитектуры программного обеспечения ИС и АС. 29](#_Toc93159906)

[15. Основные современные технологии разработки программного обеспечения ИС и АС. 32](#_Toc93159907)

[16. Программные интерфейсы информационных и автоматизированных систем. 38](#_Toc93159908)

[17. Аппаратные интерфейсы связи информационных и автоматизированных систем. 41](#_Toc93159909)

[18. Сетевые интерфейсы ИС и АС. Локальные и глобальные сети. 43](#_Toc93159910)

[19. Применение клиент-серверных технологий для построения ИС и АС. 47](#_Toc93159911)

[20. Понятие программного интерфейса (API). 49](#_Toc93159912)

[21. Облачные системы для комплексирования ИС и АС. (как найду что добавить доделаю) 50](#_Toc93159913)

[22. Методы удаленного доступа к ресурсам ИС и АС. 50](#_Toc93159914)

[23. Основные принципы разработки прикладного программного и аппаратного обеспечения ИС и АС. 52](#_Toc93159915)

[24. Современные аппаратные платформы ИС и АС. 54](#_Toc93159916)

[25. Понятие встраиваемых аппаратных платформ для ИС и АС. 54](#_Toc93159917)

[26. Инструментальные средства разработки и отладки программного обеспечения ИС и АС. 55](#_Toc93159918)

[27. Инструментальные средства разработки и отладки аппаратного обеспечения ИС и АС. 56](#_Toc93159919)

[28. Интеллектуальные технологии разработки программного обеспечения ИС и АС. 57](#_Toc93159920)

[**Математика и вычисления**[ 57](#_Toc93159921)

[**Разработка алгоритмов**[править | править код] 58](#_Toc93159922)

[**Визуализация данных**[править | править код] 58](#_Toc93159923)

[**Независимые приложения**[править | править код] 58](#_Toc93159924)

[**Внешние интерфейсы**[править | править код] 58](#_Toc93159925)

[IntelliCode 61](#_Toc93159926)

[29. Интеллектуальные технологии разработки аппаратного обеспечения ИС и АС. 62](#_Toc93159927)

[30. CASE-методы разработки программного и аппаратного обеспечения ИС и АС. 66](#_Toc93159928)

[31. Классификация CASE-средств разработки программного и аппаратного обеспечения ИС и АС. 67](#_Toc93159929)

[32. Графические средства для описания и документирования информационных систем. 68](#_Toc93159930)

[Общая информация 68](#_Toc93159931)

[Классификация 69](#_Toc93159932)

[**CASE средства** 70](#_Toc93159933)

[33. CASE-средства генерации исходных текстов и реализации интегрированного окружения поддержки полного жизненного цикла (ЖЦ) разработки ПО. 74](#_Toc93159934)

[34. CASE-средства программирования. 77](#_Toc93159935)

[· Наследование - Очень удобная особенность, позволяет унаследовать от родителя все свойства и функции, как бы копирует объект для дальнейшего усовершенствования и добавления новых возможностей. 77](#_Toc93159936)

[· Полиморфизм - Третья особенность объектно-ориентированного языка дельфи, позволяет объединить данные разных объектов с единой спецификацией имея различную реализацию. 77](#_Toc93159937)

[35. CASE-средства проектирования баз данных. 78](#_Toc93159938)

[36. CASE-средства реинженеринга. 81](#_Toc93159939)

[37. Верхние, средние и нижние CASE-средства. 82](#_Toc93159940)

[· Наследование - Очень удобная особенность, позволяет унаследовать от родителя все свойства и функции, как бы копирует объект для дальнейшего усовершенствования и добавления новых возможностей. 86](#_Toc93159941)

[· Полиморфизм - Третья особенность объектно-ориентированного языка дельфи, позволяет объединить данные разных объектов с единой спецификацией имея различную реализацию. 86](#_Toc93159942)

[38. Процесс разработки программ для ИС и АС при помощи CASE-средств. 87](#_Toc93159943)

[39. Функционально-ориентированное (структурное) проектирование программного обеспечения при помощи CASE-средств. 96](#_Toc93159944)

[40. Методология разработки программных систем msf (Microsoft Solutions Framework). 97](#_Toc93159945)

[41. Современный стек технологий для разработки программного обеспечения ИС и АС. 104](#_Toc93159946)

[Что такое стек технологий? 104](#_Toc93159947)

[Обзор фреймворков Frontend 105](#_Toc93159948)

[Javascript или Typescript? 107](#_Toc93159949)

[CSS, Less или Sass? 107](#_Toc93159950)

[Что следует учитывать при выборе технологического стека для проекта? 108](#_Toc93159951)

[Вывод 109](#_Toc93159952)

[42. Нативные средства, библиотеки и инструменты разработки программного и аппаратного обеспечения ИС и АС. 110](#_Toc93159953)

[**Преимущества и недостатки нативного подхода** 110](#_Toc93159954)

[**Плюсы нативных приложений** 110](#_Toc93159955)

[**Минусы разработки нативных приложений** 111](#_Toc93159956)

[**Выводы** 112](#_Toc93159957)

[43. Кроссплатформенные средства, библиотеки и инструменты разработки программного и аппаратного обеспечения ИС и АС. 113](#_Toc93159958)

[Кроссплатформенные языки программирования 113](#_Toc93159959)

[Кроссплатформенные среды исполнения 113](#_Toc93159960)

[Кроссплатформенный пользовательский интерфейс 113](#_Toc93159961)

[React Native 114](#_Toc93159962)

[Язык программирования 114](#_Toc93159963)

[Кто использует 114](#_Toc93159964)

[IDE и написание кода 115](#_Toc93159965)

[Архитектура и исполнение кода 115](#_Toc93159966)

[Производительность 115](#_Toc93159967)

[Flutter 115](#_Toc93159968)

[Язык программирования 115](#_Toc93159969)

[Кто использует 115](#_Toc93159970)

[IDE и написание кода 115](#_Toc93159971)

[Архитектура и исполнение кода 115](#_Toc93159972)

[Производительность 116](#_Toc93159973)

[Ionic 116](#_Toc93159974)

[Язык программирования 116](#_Toc93159975)

[Кто использует 116](#_Toc93159976)

[IDE и написание кода 116](#_Toc93159977)

[Архитектура и исполнение кода 116](#_Toc93159978)

[Производительность 116](#_Toc93159979)

[Xamarin 116](#_Toc93159980)

[Язык программирования 116](#_Toc93159981)

[Кто использует 116](#_Toc93159982)

[IDE и написание кода 117](#_Toc93159983)

[Архитектура и исполнение кода 117](#_Toc93159984)

[Производительность 117](#_Toc93159985)

[PhoneGap 117](#_Toc93159986)

[Язык программирования 117](#_Toc93159987)

[Кто использует 117](#_Toc93159988)

[IDE и написание кода 117](#_Toc93159989)

[Архитектура и исполнение кода 117](#_Toc93159990)

[Производительность 118](#_Toc93159991)

[Сравнительная таблица 118](#_Toc93159992)

[44. Десктопные приложения ИС и АС (ASS). 119](#_Toc93159993)

[45. Веб-технологии в ИС и АС. 120](#_Toc93159994)

[46. Мобильные приложения в ИС и АС. 121](#_Toc93159995)

[47. Программные и аппаратные средства защиты информации в ИС и АС. 122](#_Toc93159996)

[48. Технологии хранения данных ИС и АС. 124](#_Toc93159997)