

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  информационных технологий | **Кафедра**  информационных систем |

**Реферат № 5**

по дисциплине «**Основы программной инженерии**»

на тему: «Проектирование программных систем»

|  |  |  |
| --- | --- | --- |
| **Студент** группа ИДБ–22–06 |  | **Мустафаева П.М.** |
|  | подпись |  |
| **Руководитель**  к.э.н., доцент |  | **Ахмедов Э.Р.** |
|  | подпись |  |

**1. Для чего необходим проект программной системы? Дайте определение проектирования ПС.**

Проект программной системы необходим для того, чтобы четко определить структуру, функциональность и поведение будущего программного продукта. На этапе проектирования формируется архитектура системы, выделяются её основные компоненты, определяются интерфейсы взаимодействия между ними, а также выбираются подходящие алгоритмы и технологии. Без тщательно проработанного проекта разработка может стать хаотичной, что приведет к увеличению сроков, перерасходу ресурсов и появлению критических ошибок в итоговом продукте.

Проектирование программной системы – это процесс создания детализированной модели, которая описывает, как система будет работать, из каких частей состоять и как эти части будут взаимодействовать. Оно включает в себя несколько уровней детализации: от высокоуровневого архитектурного проектирования, где определяется общая структура системы, до детального проектирования, в рамках которого прорабатываются алгоритмы, структуры данных, интерфейсы модулей и другие технические аспекты. Важной частью проектирования является выбор подходящих технологий, инструментов и методологий разработки, которые обеспечат выполнение требований к системе.

**2. Каковы основные цели проектирования? Что такое процесс проектирования?**

Основные цели проектирования программной системы:

* Реализация требований – обеспечение соответствия системы функциональным и нефункциональным требованиям (производительность, безопасность, удобство использования).
* Оптимизация структуры – создание логичной, модульной и масштабируемой архитектуры, упрощающей разработку и поддержку.
* Минимизация рисков – выявление и устранение потенциальных проблем на ранних этапах, до начала реализации.
* Эффективное использование ресурсов – выбор оптимальных технологий, алгоритмов и подходов для снижения затрат времени и средств.
* Обеспечение сопровождаемости – проектирование системы таким образом, чтобы её можно было легко модифицировать и расширять в будущем.
* Документирование решений – создание четких спецификаций для согласованной работы команды и последующего анализа.

Процесс проектирования – инженерная деятельность в рамках ЖЦ ПП.

Процесс проектирования представляет собой последовательность этапов, которой следуют проектировщики при поиске и реализации решений. Это и есть систематизированный подход. По сути это похоже на «научный метод», преподаваемый учащимся. Единого процесса проектирования, принятого во всем мире, не существует. Большинство проектировщиков используют собственные методики в процессе работы. Процесс начинается с постановки задачи и завершается представлением готового решения, но промежуточные этапы могут быть различными.

Применение процесса проектирования:

* Этап 1 - ПОНЯТЬ - Определить задачу.
* Этап 2 - ИССЛЕДОВАТЬ - Произвести исследования.
* Этап 3 – ОПРЕДЕЛИТЬ – Выделить спецификации решения.
* Этап 4 – ПРЕДСТАВИТЬ – Выработать решения.
* Этап 5 – ПРОТОТИП – Понять, работают ли концепции.
* Этап 6 – ВЫБРАТЬ – Выделить окончательную концепцию.
* Этап 7 – ДОРАБОТАТЬ – Детально проработать проект.
* Этап 8 – ПОКАЗАТЬ – Получить реакцию и утверждение.
* Этап 9 – ПРИМЕНИТЬ – Применить решение.
* Этап 10 – ИСПЫТАТЬ – Испытать решение на соответствие рабочих показателей.
* Этап 11 – ПОВТОРИТЬ.

**3. Каковы роли, которые могут участвовать в процессе проектирования? Назовите их основные задачи.**

Роли участников процесса проектирования:

* Заказчик – играет ключевую роль в проекте, так как именно он формулирует основные бизнес-требования и цели. Он участвует в согласовании ключевых решений, утверждает бюджет и принимает результаты на каждом этапе проектирования. Без четкого понимания потребностей заказчика невозможно создать эффективное решение.
* Руководитель проекта – координирует всю команду и следит за тем, чтобы процесс проектирования шел согласно плану. В его задачи входит контроль сроков, бюджета и соответствия технических решений поставленным требованиям. Он также управляет коммуникацией между участниками и минимизирует возможные риски, которые могут возникнуть в ходе работы.
* Системный администратор – оценивает инфраструктурные требования, такие как серверы, сети и системы безопасности. Он помогает выбрать оптимальные технологии для развертывания и эксплуатации системы, а также следит за тем, чтобы проектируемое решение соответствовало ИТ-политикам компании.
* Администратор базы данных – отвечает за проектирование структуры БД, ее производительность и безопасность. Он оптимизирует запросы, разрабатывает схемы хранения данных и участвует в выборе подходящей СУБД. Его работа напрямую влияет на скорость и надежность будущей системы.
* Системный архитектор – определяет общую архитектуру программного решения, будь то монолитная система, микросервисы или гибридный вариант. Он выбирает технологии и фреймворки, которые обеспечат масштабируемость, отказоустойчивость и безопасность системы.
* Архитектор базы данных – разрабатывает концептуальную и логическую модель данных, оптимизирует индексы и репликацию. Его задача – создать такую структуру БД, которая будет эффективно работать в рамках выбранной архитектуры и удовлетворять потребностям бизнеса.
* Бизнес-аналитик – выступает связующим звеном между заказчиком и технической командой. Он анализирует бизнес-процессы, преобразует их в четкие требования и формирует пользовательские сценарии. Благодаря его работе разработчики лучше понимают, что именно нужно реализовать.
* Аналитик (системный аналитик) – углубляется в технические детали, детализирует требования и проектирует функциональные спецификации. Он участвует в создании технического задания, которое служит руководством для разработчиков, и помогает избежать недопонимания на этапе реализации.
* Тестировщик – включается в процесс уже на этапе проектирования, чтобы оценить тестируемость системы и разработать стратегию проверки. Он анализирует проектные решения на предмет возможных уязвимостей и дефектов, что позволяет выявить проблемы до начала разработки.

**4. Дайте определение архитектуры программного обеспечения.**

Архитектура – это структура программы или вычислительной системы, определяющая ее работу на самом высоком концептуальном уровне, включая аппаратные и программные компоненты, видимые снаружи свойства этих компонентов, отношения между ними, а также документирование системы

Для разработки архитектуры системы привлекаются специалисты со следующими ролями: системный архитектор (проектирует систему в целом, а также отдельные ее компоненты), архитектор базы данных (занимается проектированием БД и ее структуры), системный аналитик (участвует в проектировании, подготавливает документацию), администраторы (участвуют в проектировании аппаратной части системы).

Архитектура программного обеспечения – это фундаментальная структура системы, определяющая её компоненты, их взаимодействие и принципы разработки.

Она определяет, как различные модули программы взаимодействуют между собой, в каком виде данные передаются в системе и как приложение будет развиваться в будущем.

Задачи архитектуры программного обеспечения:

* Улучшение и повышение продуктивности процессов.
* Уменьшение затрат.
* Улучшение операционной деятельности.
* Повышение эффективности управления.
* Уменьшение рисков
* Повышение эффективности IT-подразделений.
* Повышение продуктивности работы пользователей.
* Повышение возможности и прозрачности взаимодействия.
* Уменьшение стоимости «поддержки» жизненного цикла.
* Улучшение характеристик безопасности.
* Повышение управляемости.

Обшей техникой постепенной доработки архитектуры является итеративная методика (пока не будут удовлетворены все требования и ограничения), включающая пять основных этапов (рис. 1):

1. Определение целей архитектуры.

На этом этапе формулируются ключевые требования и ограничения системы. Архитектор совместно с заказчиком определяет приоритетные характеристики: производительность, безопасность, масштабируемость. Эти цели становятся основой для принятия проектных решений.

1. Выявление основных сценариев.

Анализируются типичные и критические пути использования системы. Выделяются наиболее важные пользовательские потоки и экстремальные нагрузки. Это помогает понять реальные требования к работе системы и выявить ключевые компоненты.

1. Создание прототипа приложения.

Разрабатывается упрощенная версия системы, реализующая основную функциональность. Прототип позволяет проверить архитектурные решения на практике и выявить потенциальные проблемы до начала полномасштабной разработки.

1. Выявление потенциальных проблем.

На основе анализа прототипа и архитектуры выявляются слабые места: узкие места производительности, уязвимости безопасности, проблемы интеграции. Особое внимание уделяется критическим сценариям использования.

1. Определение вариантов решений.

Для каждой выявленной проблемы рассматриваются альтернативные подходы. Оцениваются различные технологии и архитектурные паттерны. Выбирается оптимальное решение, учитывающее поставленные цели и ограничения проекта.



Рис. 1 Процесс создания архитектуры ПО

**5. Какие задачи решает разработка архитектуры приложения?**

Архитектура приложения – это каркас, на котором строится любая программа – от простого сайта до сложной корпоративной системы.

Это общий план будущего программного обеспечения, в котором определяют основные части программы и как они будут взаимодействовать друг с другом и внешним миром.

Разработка архитектуры приложения решает следующие ключевые задачи:

* Структурирование системы – определяет логическую организацию компонентов, их взаимодействие и границы ответственности. Создает четкую модульную структуру для упрощения разработки и поддержки.
* Обеспечение качества системы – решает вопросы производительности, надежности, безопасности и масштабируемости через продуманные архитектурные решения.
* Управление сложностью – декомпозирует систему на manageable компоненты, снижая когнитивную нагрузку на разработчиков.
* Поддержка эволюции системы – закладывает гибкость для будущих изменений и расширений без необходимости полного перепроектирования.
* Оптимизация ресурсов – определяет эффективные способы использования вычислительных мощностей, памяти и сетевых ресурсов.
* Стандартизация разработки – устанавливает единые подходы, паттерны и принципы для всей команды разработчиков.
* Интеграционная совместимость – обеспечивает корректное взаимодействие с внешними системами и сервисами.
* Управление техническими рисками – выявляет и нивелирует потенциальные проблемы на ранних этапах проектирования.
* Поддержка бизнес-требований – транслирует бизнес-цели в технически реализуемые решения.
* Обеспечение тестируемости – закладывает возможности для эффективного тестирования системы.

**6. Как определяются исходные данные для проектирования архитектуры приложения?**

Исходные данные для проектирования архитектуры приложения определяются на этапе анализа требований и играют ключевую роль в создании эффективной и устойчивой системы. В первую очередь, необходимо собрать и проанализировать функциональные требования – какие задачи должно выполнять приложение, какие функции предоставлять пользователям, и каковы бизнес-цели проекта. Эти требования формируются на основе общения с заказчиком, анализа текущих бизнес-процессов и изучения аналогичных решений.

Затем следует определить нефункциональные требования, такие как производительность, масштабируемость, безопасность, удобство поддержки, отказоустойчивость и кроссплатформенность. Эти характеристики оказывают влияние на выбор технологий, шаблонов проектирования и структуру компонентов системы.

Также важно учитывать ограничения и внешние факторы, которые могут повлиять на архитектуру. Это могут быть ограничения по времени и бюджету, используемые платформы и операционные системы, требования к совместимости с существующими системами, а также предпочтения в используемых языках программирования, фреймворках и базах данных.

Дополнительно анализируется целевой пользователь и среда использования приложения. Отсюда можно определить подходящий пользовательский интерфейс, уровень доступа, ожидаемую нагрузку и сценарии взаимодействия с системой. Все собранные данные документируются и служат основой для проектирования архитектурного решения, направленного на оптимальное удовлетворение требований проекта.

**7. Какие задачи и ограничения определяют цели архитектуры приложения?**

Цели архитектуры приложения формируются на основе задач, которые должно решать приложение, и ограничений, с которыми сталкивается команда разработки. Эти цели направлены на создание такой структуры системы, которая будет эффективно выполнять свои функции, легко масштабироваться и сопровождаться, а также соответствовать требованиям бизнеса.

Задачи, определяющие цели архитектуры, включают:

* Обеспечение функциональности: архитектура должна позволять реализовать все заявленные функции приложения.
* Поддержка масштабируемости: возможность увеличения нагрузки без потери производительности.
* Обеспечение надежности и отказоустойчивости: система должна сохранять работоспособность при сбоях отдельных компонентов.
* Гибкость и расширяемость: архитектура должна позволять легко добавлять новые функции без переработки всего приложения.
* Облегчение сопровождения и тестирования: структура системы должна быть понятной и модульной, чтобы упростить внесение изменений и исправление ошибок.
* Обеспечение безопасности: защита данных и процессов от несанкционированного доступа, и атак.

Ограничения, которые влияют на архитектуру, включают:

* Ограничения времени и бюджета: архитектурное решение должно быть реалистичным в рамках доступных ресурсов.
* Технологические ограничения: требования к использованию определённых языков программирования, баз данных, платформ или фреймворков.
* Совместимость с существующими системами: необходимость интеграции с уже работающими решениями.
* Командная компетенция: архитектура должна соответствовать опыту и знаниям команды разработчиков.
* Аппаратные и сетевые ограничения: характеристики серверов, устройств пользователей, пропускной способности сети и т.п.

**8. Каковы основные типы разрабатываемых приложений?**

Существует несколько основных типов разрабатываемых приложений, каждый из которых имеет свои особенности, цели и области применения. Выбор типа приложения зависит от задач проекта, целевой аудитории, платформы и технических требований. Ниже приведены основные категории:

1. Настольные (desktop) приложения.

Это программы, устанавливаемые и запускаемые на персональных компьютерах или ноутбуках. Они не требуют постоянного Интернет-соединения и часто используются в корпоративной среде (например, бухгалтерские системы, редакторы, инженерное ПО). Примеры: Microsoft Word, Adobe Photoshop.

2. Веб-приложения (web applications).

Работают через веб-браузер и обычно не требуют установки на устройство пользователя. Они доступны с любого устройства, подключённого к интернету. Часто применяются для информационных систем, интернет-магазинов, CRM, социальных сетей. Примеры: Gmail, Trello, интернет-банкинг.

3. Мобильные приложения (mobile applications).

Разрабатываются для смартфонов и планшетов, работают на платформах Android и iOS. Подразделяются на нативные (разрабатываются отдельно для каждой платформы), кроссплатформенные (используют общий код), и гибридные. Примеры: WhatsApp, Instagram, мобильные банки.

4. Встраиваемые приложения (embedded applications).

Используются в устройствах с встроенными микроконтроллерами и выполняют строго определённые функции. Примеры: программное обеспечение для бытовой техники, автомобилей, медицинского оборудования.

5. Облачные приложения (cloud applications).

Используют ресурсы облачной инфраструктуры и обеспечивают доступ к данным и функциям через интернет. Отличаются высокой масштабируемостью и гибкостью. Часто применяются в корпоративной сфере. Примеры: Google Workspace, Microsoft Azure приложения.

6. Гибридные и кроссплатформенные приложения.

Разрабатываются с использованием фреймворков, которые позволяют запускать одно и то же приложение на разных платформах (например, и на Windows, и на Android). Используются для снижения затрат на разработку. Примеры технологий: Flutter, React Native, Xamarin.

7. Серверные и фоновые приложения (backend services, daemons).

Не имеют пользовательского интерфейса и работают в фоновом режиме, обеспечивая обработку данных, интеграцию с внешними системами, поддержку API и др.

**9. Перечислите и опишите основные ограничения развертывания.**

При проектировании архитектуры приложения необходимо учесть корпоративную политику и процедуры, а также среду, в которой планируется развертывание приложения. Если целевая среда является фиксированной или негибкой, то архитектура приложения должна отражать существующие в этой среде ограничения. Также должны быть учтены нефункциональные требования, такие как безопасность, надежность и др. Можно выделить следующие основные ограничения развертывания:

* ограничения по аппаратным требованиям. Например, необходимо использовать процессоры, поддерживающие определенные технологии.
* ограничения по программным требованиям. Например, использование конкретной операционной системы;
* ограничения собственно по развертыванию. Возможна ситуация с использованием отдельного сервера для установки приложения либо покупки места для сервера у поставщиков таких услуг. Можно выделить ограничения пропускной способности канала, размеров доступной оперативной памяти, если развертывание будет происходить на системе, где невозможно делать аппаратные изменения.

Повышенные требования безопасности. Данные требования могут ограничить возможные места расположения серверов, либо потребуется использование дополнительных аппаратных (программных) средств для обеспечения требуемого уровня надежности системы.

**10. Дайте краткую характеристику архитектурным стилям проектирования.**

Архитектурный стиль (парадигма) может рассматриваться как обобщенный шаблон для проектирования архитектуры ПО, опирающийся на некий набор принципов и обеспечивающий абстрактную базу для определенного семейства систем ПО, опирающийся на некий набор принципов и обеспечивающий абстрактную базу для определенного семейства систем.

Каждый стиль определяет набор правил, которые задают типы компонентов, используемых для компоновки системы, и типы отношений, применяемых при компоновке, а также ограничения по способам компоновки.

Архитектура программной системы практически никогда не ограничена лишь одним архитектурным стилем и, как правило, сочетает в себе несколько архитектурных стилей.

Ниже приведен список типовых архитектурных стилей:

* Клиент-серверная архитектура: Система разделяется на клиентскую и серверную части, где клиент посылает запросы к серверу. Во многих случаях в роли сервера выступает сервер БД, а логика приложения представлена процедурами хранения.
* Компонентная архитектура: Дизайн приложения разлагается на функциональные (логические) компоненты, предоставляющие тщательно проработанные интерфейсы связи, с возможностью их повторного использования.
* Проблемно-ориентированное проектирование (дизайн на основе предметной области): Объектно-ориентированные стиль, ориентированный на моделирование сферы деловой активности и определяющий бизнес-объекты на основании сущностей данной предметной области.
* Многослойная архитектура: Функциональные области приложения разделяются на многослойные группы (уровни).
* Архитектура на основе канала сообщений: Стиль, предписывающий использование ПС, которая может принимать и отправлять сообщения по одному или более каналу связи. Приложения получают возможность взаимодействия, не располагая конкретными сведениями друг о друге.
* N-уровневая/3-уровневая архитектура: Функциональность выделяется в отдельные сегменты во многом аналогично многослойному стилю, но сегменты физически располагаются на разных компьютерах (уровнях).
* Объектно-ориентированная архитектура: Парадигма проектирования, основанная на распределении ответственности приложения (системы) между отдельными, многократно используемыми и самостоятельными объектами, содержащими данные и правила поведения.
* Сервисно-ориентированная архитектура (SQA): Описывает приложения, предоставляющие и потребляющие функциональность в виде сервисов с помощью контрактов и сообщений.

**11. Какие из архитектурных стилей можно совмещать в одном архитектурном решении и какие нельзя? Почему?**

Многие архитектурные стили могут быть успешно совмещены в одном архитектурном решении, поскольку они отвечают за разные уровни абстракции, аспекты организации кода или взаимодействия компонентов. Например, клиент-серверная архитектура, многослойная архитектура и N-уровневая архитектура часто используются вместе. Клиент-сервер определяет общую структуру взаимодействия между клиентом и сервером, многослойность помогает организовать внутреннюю логику приложения (например, слои представления, бизнес-логики и данных), а N-уровневая архитектура реализует эти слои на разных физических узлах.

Компонентная архитектура также легко сочетается с вышеуказанными стилями, поскольку компоненты могут быть размещены в разных слоях или уровнях и обеспечивают модульность, переиспользуемость и изоляцию функциональности. Вместе с ней хорошо работает объектно-ориентированная архитектура, поскольку компоненты часто реализуются как совокупность взаимодействующих объектов, использующих принципы ООП.

Проблемно-ориентированное проектирование (Domain-Driven Design) также можно успешно объединять с многослойной архитектурой и ООП. В рамках DDD строится предметная модель (domain model), которая организована в виде объектов и может быть разделена на слои: домен, приложение, инфраструктура. DDD не заменяет архитектуру, а задаёт подход к построению модели, поэтому хорошо сочетается с другими стилями.

Сервисно-ориентированная архитектура (SOA) тоже может быть частью комплексного архитектурного решения. Каждый сервис внутри SOA может быть реализован как многослойное приложение, использовать компонентный подход и внутренне опираться на объектно-ориентированную модель или DDD. При этом обмен между сервисами может быть организован через архитектуру на основе канала сообщений, особенно если требуется асинхронное взаимодействие. Эти два стиля хорошо дополняют друг друга в распределённых системах.

Однако не все стили можно безусловно совмещать. Например, архитектура на основе канала сообщений и классическая клиент-серверная архитектура могут вступать в противоречие, если используются одновременно в одной и той же части системы. Клиент-серверная модель подразумевает синхронные запросы и прямой отклик, тогда как архитектура сообщений – асинхронный, косвенный обмен через брокеры. Их можно использовать в рамках разных подсистем, но совмещение этих подходов без ясных границ приведёт к усложнению логики и проблемам с поддержкой.

Таким образом, большинство стилей из списка можно сочетать, особенно если они отвечают за разные аспекты (логика, структура, взаимодействие). Однако при совмещении архитектурных подходов важно избегать конфликтов между синхронной и асинхронной моделями взаимодействия и чётко разграничивать зоны ответственности каждого стиля.

**12. На какие ключевые вопросы следует обращать внимание при проектировании? Почему?**

При проектировании архитектуры приложения важно обращать внимание на ряд ключевых вопросов, так как они напрямую влияют на устойчивость, эффективность и возможность развития системы в будущем. Один из самых главных вопросов – какие задачи должно решать приложение. Это помогает понять, какие функции необходимо реализовать, а значит, определяет основу всей архитектуры. Без чёткого понимания целей проекта можно построить систему, которая будет неудобной для пользователя или не решать поставленные бизнес-задачи.

Также важно учитывать, кто будет пользоваться приложением и в каких условиях. Например, если приложение ориентировано на массового пользователя с мобильного телефона, оно должно быть лёгким, быстрым и иметь простой интерфейс. Если же это корпоративная система, может быть важна безопасность, масштабируемость и возможность подключения к другим внутренним сервисам.

Отдельное внимание следует уделить ограничениям проекта – это могут быть сроки, бюджет, технические ограничения (например, необходимость работать только на определённой платформе), а также уровень подготовки команды. Эти факторы влияют на выбор технологий и архитектурных решений. Игнорирование ограничений может привести к тому, что проект не удастся реализовать в нужное время или с необходимым качеством.

Ещё один важный вопрос – какой объём нагрузки планируется. Если не учесть это на этапе проектирования, приложение может не выдерживать количество пользователей или объём данных, особенно если система будет развиваться. Поэтому важно заранее подумать о масштабируемости и резервировании ресурсов.

Также нужно продумать структуру кода и возможность его поддержки в будущем. Это важно для того, чтобы другие разработчики могли легко разобраться в проекте, добавлять новые функции или исправлять ошибки. Хорошая архитектура делает приложение не только работоспособным, но и удобным в развитии, что особенно важно для долгосрочных проектов.

**13. Какие методы обеспечения отказоустойчивости используют наиболее часто? Перечислите достоинства и недостатки каждого из них.**

Некоторые часто используемые методы обеспечения отказоустойчивости:

1. Резервирование.

Основано на использовании различных устройств и модулей в качестве резервных. При отказе основного элемента осуществляется переход на использование резервного.

Достоинства: отсутствие даже кратковременного перерыва в работе при отказе элементов системы.

Недостатки: одновременный износ резервных элементов системы вместе с остальными, высокая стоимость реализации метода.

2. Метод голосования.

Элементы системы работают параллельно и в любой момент времени должны находиться в одинаковом состоянии.

Достоинства: отсутствие даже кратковременного перерыва в работе при отказе элементов системы, возможность определять не только отказ элементов, но и ошибки его работы (например, программные).

Недостатки: одновременный износ основных и дублирующих элементов системы, высокая стоимость реализации метода.

3. Рациональное статистическое перераспределение задач.

Метод основывается на том, что обычно процессорные модули используют не все доступные им ресурсы.

Достоинства: более низкая стоимость реализации (чем у резервирования и голосования).

Недостатки: сложность формирования плана распределения задач, особенно в ситуации, когда нужно учесть возможность выхода из строя большого количества процессорных модулей, возможна ситуация, что у процессорных модулей недостаточно много избыточных ресурсов.

4. Репликация данных.

Дублирование информации между двумя серверами.

Достоинства: рост отказоустойчивости (если один из серверов выйдет из строя, остальные продолжат работу), рост производительности.

Недостатки: при отказе мастер-сервера, на котором происходит запись данных, все реплики переходят в режим Read Only, в это время нельзя добавлять или удалять записи в базу.

**14. Как можно графически отобразить архитектуру ПС? Опишите основные средства отображения архитектуры ПС.**

Графическое отображение архитектуры программной системы (ПС) – это важный способ визуализировать её структуру, компоненты, связи и взаимодействие между ними. Такие схемы позволяют разработчикам, аналитикам и другим участникам проекта лучше понять, как устроена система, упростить сопровождение, проектирование и развитие. Существуют различные средства и виды диаграмм, которые применяются для этих целей.

Одним из самых распространённых средств является UML (Unified Modeling Language) – универсальный язык моделирования, который включает в себя множество типов диаграмм. Для архитектуры чаще всего используются:

* Диаграмма компонентов – показывает, из каких модулей состоит система и как они связаны. Это помогает представить компоненты программного обеспечения как "блоки", которые взаимодействуют друг с другом.
* Диаграмма развертывания – отображает, как программные компоненты размещаются на физических или виртуальных устройствах (например, серверах, клиентских машинах).
* Диаграмма классов — используется для отображения структуры классов, особенно в объектно-ориентированной архитектуре.
* Диаграмма последовательности – показывает, как объекты или компоненты взаимодействуют друг с другом во времени, что полезно для анализа логики взаимодействия.

Кроме UML, применяются и блок-схемы или логические схемы, где упрощённо показываются основные части системы: пользовательский интерфейс, сервер приложений, базы данных и другие узлы. Такие схемы часто строят вручную или в инструментах вроде Microsoft Visio, Draw.io, Lucidchart и др.

Также могут использоваться архитектурные шаблоны в виде схем, например, схема многослойной архитектуры с чётким разделением слоёв (UI, логика, данные) или схема микросервисной архитектуры, где каждый сервис обозначается отдельным блоком со связями через API или сообщения.

**15. Каковы основные атрибуты анализа качества программного дизайна.**

Основные атрибуты анализа качества программного дизайна отражают свойства архитектуры, которые влияют на поведение системы во время работы, её взаимодействие с пользователем, а также удобство сопровождения и развития. Эти параметры качества охватывают такие важные характеристики, как удобство и простота использования, производительность, надёжность и безопасность. Они позволяют судить о том, насколько проект соответствует требованиям и ожиданиям как со стороны пользователей, так и со стороны разработчиков.

К числу ключевых атрибутов качества дизайна относятся:

* Тестируемость – насколько просто проверять систему и её отдельные компоненты на корректность работы.
* Переносимость – способность системы функционировать в различных средах без значительных изменений.
* Модифицируемость – возможность легко вносить изменения в код, добавлять новую функциональность или исправлять ошибки.
* Производительность – насколько быстро и эффективно работает система при выполнении задач.
* Безопасность – защита данных и предотвращение несанкционированного доступа или сбоев.

Важной частью архитектурного качества являются такие свойства, как концептуальная целостность дизайна (единство подхода и стиля при проектировании), непротиворечивость (отсутствие логических конфликтов между компонентами системы), полнота (все важные аспекты учтены) и завершённость (дизайн отражает все предусмотренные требования).

Для анализа качества дизайна используются разные техники и подходы. Среди них:

* Обзор дизайна – это может быть коллективное обсуждение архитектурных решений внутри команды, направленное на выявление слабых мест.
* Статический анализ – включает проверку соответствия дизайна требованиям, трассировку зависимостей и логики.
* Симуляция и прототипирование – позволяют оценить, как архитектура будет вести себя в реальных условиях, например, при высокой нагрузке. Это помогает заранее выявить проблемы с производительностью или масштабируемостью.