***Вопросы и ответы Сергей Сергеевич Программирование***

***Теоретические вопросы:***

1. **Опишите понятие требований к программному обеспечению, классификацию требований.**

**Требования к программному обеспечению** - это описание функциональных и нефункциональных возможностей, которые должно обладать разрабатываемое программное обеспечение. Они представляют собой четкую формулировку потребностей заказчика или пользователей и являются основой для проектирования и разработки программного продукта.

**Классификация требований** может быть выполнена по различным критериям. Одна из самых распространенных классификаций основана на разделении требований на функциональные и нефункциональные.

1. ***Функциональные требования:***

- Описывают функции и задачи, которые должны выполняться программным обеспечением.

- Определяют, какие действия или операции должны быть доступны пользователю.

- Примеры: создание пользователя, отправка электронного письма, вычисление суммы чисел и т.д.

2. ***Нефункциональные требования:***

- Описывают атрибуты или характеристики программного обеспечения, которые не связаны прямо с его функциями.

- Определяют ограничения производительности, надежности, безопасности, эргономики и другие аспекты.

- Примеры: время отклика системы, количество одновременных пользователей, требования к безопасности данных и т.д.

1. **Уровни требований к программному обеспечению. Понятие, описание уровней, примеры.**

**Уровни требований к программному обеспечению** - это структурированный подход к определению и организации требований к разрабатываемому ПО. Это позволяет разработчикам и заказчикам находить общий язык и легко распознавать их характеристики и связи.

**Основные уровни требований к ПО:**

1. **Уровень бизнес-требований:** Этот уровень описывает цели и потребности бизнеса, которые должны быть реализованы в программном обеспечении. Примеры: улучшение эффективности бизнес-процессов, повышение удовлетворенности клиентов.

2. **Уровень пользовательских требований:** Этот уровень описывает ожидания пользователей от программного обеспечения. Примеры: простой и интуитивно понятный интерфейс, быстрая и отзывчивая работа.

3. **Уровень системных требований:** Этот уровень описывает функциональные и нефункциональные требования, которые должны быть выполнены программным обеспечением. Примеры: совместимость с определенными операционными системами, безопасность данных.

4. **Уровень архитектурных требований:** Этот уровень описывает архитектурные решения и требования, которые должны быть учтены при разработке ПО. Примеры: использование клиент-серверной архитектуры, модульное построение системы.

5. **Уровень компонентных требований:** Этот уровень описывает требования к индивидуальным компонентам и модулям программного обеспечения. Примеры: точные описания функциональности каждого компонента, требования к API.

1. **Жизненный цикл программного обеспечения. Понятие, основные этапы.**

***Жизненный цикл ПО*** – период разработки и эксплуатации ПО.

***Этапы Жизненного цикла:***

1. Возникновение и исследование идеи.
2. Анализ требований и проектирование.
3. Программирование.
4. Тестирование и отладка
5. Ввод программы в действие.
6. Эксплуатация и сопровождение.
7. Завершение эксплуатации.
8. **Опишите процесс анализа предметной области.**

***Анализ предметной области***

Для того, чтобы разработать программную систему, приносящую реальные выгоды определенным пользователям, необходимо сначала выяснить, какие же задачи она должна решать для этих людей и какими свойствами обладать.

***Требования к ПО***определяют, какие свойства и характеристики оно должно иметь для удовлетворения потребностей пользователей и других заинтересованных лиц. Однако сформулировать требования к сложной системе не так легко. В большинстве случаев будущие пользователи могут перечислить набор свойств, который они хотели бы видеть, но никто не даст гарантий, что это — исчерпывающий список. Кроме того, часто сама формулировка этих свойств будет непонятна большинству программистов: могут прозвучать фразы типа «должно использоваться и частотное, и временное уплотнение каналов», «передача клиента должна быть мягкой», «для обычных швов отмечайте бригаду, а для доверительных — конкретных сварщиков», и это еще не самые тяжелые для понимания примеры.

1. **Опишите процесс разработки и оформления технического задания на разработку программного обеспечения.**

**Процесс разработки и оформления технического задания** на разработку программного обеспечения (ТЗ) включает несколько этапов.

**Вот общая структура этого процесса:**

***1. Подготовительный этап:***

- Определение потребностей заказчика и формулирование целей и требований к программному продукту.

- Исследование рынка и конкурентов.

- Определение бизнес-процессов и основных функций программного продукта.

- Сбор и анализ всей доступной информации, связанной с проектом.

***2. Написание ТЗ:***

- Описание общей концепции программного продукта.

- Формулирование целей и задач разработки.

- Определение требований к функциональности и пользовательскому интерфейсу.

- Описание архитектуры системы и компонентов.

- Определение требований к надежности, безопасности, производительности, масштабируемости и другим характеристикам системы.

- Определение требований к интеграции с другими системами и сторонними сервисами.

- Разработка схем баз данных и описание их структуры и связей.

- Определение требований к документации и тестированию.

***3. Утверждение ТЗ:***

- Проведение совещания с заказчиком и исполнителями для обсуждения и уточнения ТЗ.

- Внесение необходимых изменений и дополнений в ТЗ на основе обратной связи и согласований между заказчиком и исполнителями.

- Получение одобрения и подписания ТЗ со стороны заказчика.

***4. Внесение изменений в ТЗ (по мере необходимости):***

- При необходимости внесения изменений в ТЗ следует произвести анализ влияния этих изменений на работу всей системы именно в аспектах контракта.

- Изменение ТЗ должно быть оформлено дополнением или обновлением соглашения.

**5. Пример ТЗ**

1. Введение
   1. Наименование программы.
   2. Краткая характеристика области применения.
2. Назначение разработки.
   1. Основные особенности приложения.
   2. Технические требования.
3. Структура приложения.
   1. Тестирование программы.
   2. Сроки выполнения проекта.

***6. Окончательное утверждение:***

- При получении одобрения заказчика и исполнителями, ТЗ подписывается и считается окончательно утвержденным.

1. **Опишите процесс разработки и оформления брифа на разработку программного обеспечения.**

Пример №1

***О продукте***

* 1. Что представляет собой продукт?
  2. ОС, с которой будет работать приложение.
  3. Принципиальные отличия от существующих аналогов.
  4. Какие группы пользователей будут работать с приложением.
  5. Для чего продукт нужен пользователю? Какие проблемы пользователя он решает?
  6. Функциональные особенности приложения. Какие принципиальные процессы должны выполняться при работе с приложением.
  7. Приложение будет работать с другими ПО.

***Интерфейс и функциональность***

2.1 Опишите, каким вы видите интерфейс пользователя.

2.2 Какие цвета на ваш взгляд лучше использовать при разработке интерфейса.

2.3 Опишите основные принципы работы программного интерфейса.

2.4 Название продукта.

Пример №2

1. Введение
   1. Наименование программы.
   2. Краткая характеристика области применения.
2. Назначение разработки.
   1. Основные особенности приложения.
   2. Технические требования.
3. Структура приложения.
   1. Тестирование программы.
   2. Сроки выполнения проекта.
4. **Какими последовательными действиями можно представить процесс создания программ?**

**При создании ПП можно выделить шесть основных этапов работы:**

1. ***планирование программного проекта;***

В течение этапа планиро­вания определяются все основные задачи, которые должны быть выполнены в процессе разработки, производится оценка финан­совых, людских, технических и нетехнических ресурсов, объемов и сложности разрабатываемого ПП

1. ***составление требований заказчика;***

В течение этого этапа разра­ботчики анализируют требования к ПП (форма представления информации, необходимые функции, желательны интерфейсы, существующие ограничения и т.д.).

1. ***проектирование ПП;***

Этап проектирования предназначен для выработки и детализации модели разрабатыва­емого ПП.

1. ***разработка ПП;***

В процессе выполнения этого этапа разработчики преобразуют результаты этапа проектирова­ния в коды программ на используемом языке программирования в соответствии со стандартами кодирования.

1. ***тестирование ПП;***

Тестировщик разрабатывает тесты, выполняет процедуру тес­тирования и составляет отчеты о результатах тестирования.

1. ***сопровождение ПП.***

На этапе сопровождения основное внимание уделяется внесению изменений в ПП. Эти из­менения могут быть связаны с устранением ошибок, дополнитель­ными пожеланиями заказчика, появившимися в результате работы с ПП, изменением среды окружения и функционирования.

1. **Опишите методологии и стандарты, регламентирующие работу с требованиями к программному обеспечению.**

**Существует несколько методологий и стандартов, которые регламентируют работу с требованиями к программному обеспечению. Некоторые из них включают:**

1. Методология разработки программного обеспечения (SDLC) - это общий подход к разработке программного обеспечения, который включает в себя этапы анализа требований, проектирования, разработки, тестирования, внедрения и поддержки.

2. Agile - это гибкая методология разработки программного обеспечения, которая акцентирует внимание на постоянном взаимодействии с заказчиком и быстрой поставке рабочего программного обеспечения.

3. Scrum - это конкретная методология Agile, которая предполагает работу в небольших автономных командах, выполняющих короткие циклы разработки (итерации).

4. ISO/IEC 29110 - это международный стандарт, ориентированный на малые организации, который содержит рекомендации по управлению жизненным циклом программного обеспечения, включая управление требованиями.

5. IEEE 830 - это стандарт, определяющий формат и содержание документации по требованиям к программному обеспечению.

1. **Понятие «Программный продукт». Дайте определение, приведите классификацию.**

***Программные продукт*** – комплекс взаимосвязанных программ для решения определенной проблемы (задачи) массового спроса, подготовленный к реализации как любой вид промышленной продукции.

***Программные продукты (изделия)*** предназначены для удовлетворения потребностей пользователей, широкого распространения и продажи.

***Классификация программных продуктов может быть следующей:***

**1. По назначению:**

- Прикладное программное обеспечение (ППО) - предназначено для решения конкретных задач пользователей, например, текстовые редакторы, браузеры, игры и т.д.

- Системное программное обеспечение - предназначено для управления аппаратными ресурсами компьютера и обеспечения работы другого программного обеспечения, например, операционные системы, драйверы и т.д.

**2. По способу распространения:**

- Проприетарное программное обеспечение - разработано и продается конкретной компанией, доступ к исходным кодам ограничен.

- Свободное (открытое) программное обеспечение - разработано с открытым доступом к исходным кодам, может быть свободно использовано, изменено и распространено.

**3. По способу использования:**

- Локальное программное обеспечение - устанавливается и работает на компьютере пользователя.

- Облачное программное обеспечение - работает на удаленных серверах и доступно через интернет.

**4. По типу задач, которые они решают:**

- Бизнес-приложения - предназначены для автоматизации бизнес-процессов, учета, анализа данных и т.д.

- Научно-техническое программное обеспечение - используется в научных и инженерных расчетах, моделировании и т.д.

1. **Опишите современные принципы и методы разработки программных приложений.**

***Современные принципы разработки ПП***

1. Модульность.

Этот принцип подразумевает разделение программы на небольшие, независимые модули, каждый из которых отвечает за выполнение конкретной функции.

1. Читаемость кода.

Читаемость кода влияет на его понимание и поддержку программы в долгосрочной перспективе. Читаемый код легче анализировать и исправлять ошибки, а также вносить изменения и добавлять новые функции.

1. Тестирование

Тестирование является неотъемлемой частью разработки программных приложений. Этот принцип позволяет обнаружить и исправить ошибки и проблемы в коде, а также убедиться в правильности работы программы в различных ситуациях и условиях. Тестирование должно проводиться на всех этапах разработки — от начальной фазы до финального тестирования перед запуском.

Существуют разные методы тестирования, включая модульное тестирование, интеграционное тестирование и системное тестирование. Каждый из них имеет свою цель и позволяет обнаружить разные виды ошибок.

1. **Опишите методы организации работы в команде разработчиков.**

***Организация работы в команде разработчиков включает в себя следующие методы:***

1***.* Agile-подход:** Команда разработчиков может использовать Agile-методологии, такие как Scrum или Kanban, для организации своей работы. Это позволяет разбивать работу на небольшие итерации, называемые спринтами, и регулярно проводить обзоры итераций для оценки прогресса.

2. **Коммуникация:** Важным аспектом работы в команде разработчиков является эффективная коммуникация. Команда может использовать инструменты для обмена сообщениями, проведения видеоконференций и совместной работы над кодом, чтобы обеспечить прозрачность и согласованность в работе.

3.**Code Review:** Code Review представляет собой процесс обзора кода другими членами команды с целью выявления потенциальных ошибок, улучшения качества кода и обмена знаниями. Это помогает повысить качество программного обеспечения и обеспечить согласованность в стиле программирования.

4. **Разделение обязанностей:** Команда разработчиков может разделить обязанности между своими членами в зависимости от их экспертизы и навыков. Например, одни разработчики могут заниматься фронтенд-разработкой, другие – бэкенд-разработкой, а третьи – тестированием.

5. **Использование инструментов управления проектами: Для** эффективной организации работы команда разработчиков может использовать специализированные инструменты управления проектами, такие как Jira, Trello или Asana, для отслеживания прогресса задач, управления заданиями и планирования работы.

Эти методы помогают командам разработчиков организовать свою работу таким образом, чтобы достичь высокой производительности, эффективности и качества в разработке программного обеспечения.

1. **Понятие «Алгоритм». Дайте определение, приведите классификацию, опишите свойства алгоритмов.**

***Алгоритм*** - это точное описание упорядоченной последовательности действий, приводящей за конечное число шагов к необходимому результату.

***Свойства алгоритмов:***

1. Понятность.
2. Однозначность.
3. Дискретность (пошаговость).
4. Массовость (универсальность).
5. Результативность.
6. Конечность.
7. Безошибочность.
8. **Системы контроля версий. Дайте определение, приведите примеры, назовите преимущества работы с СКВ.**

**Системы контроля версий (СКВ)** - это программное обеспечение, предназначенное для отслеживания изменений в исходном коде программы. Оно позволяет разработчикам работать над одним и тем же проектом, контролируя изменения и управляя версиями кода. **Примеры систем контроля версий включают** Git, Subversion, Mercurial и другие.

**Преимущества работы с СКВ** включают возможность отката к предыдущим версиям кода, возможность параллельной работы нескольких разработчиков, а также улучшение общей стабильности и надежности программного продукта.

1. **Опишите основные подходы к интегрированию программных модулей.**

Основные подходы к интегрированию программных модулей ***включают вертикальную и горизонтальную интеграцию.***

**Вертикальная интеграция предполагает** интеграцию различных уровней системы, таких как интеграция базовых модулей с более высокоуровневыми модулями или интеграция прикладного программного обеспечения с операционной системой.

**Горизонтальная интеграция предполагает** интеграцию различных компонентов на одном уровне, например, интеграцию модулей, разработанных разными командами разработчиков или интеграцию различных функциональных модулей внутри приложения.

Оба подхода имеют свои преимущества и недостатки, и выбор конкретного подхода зависит от конкретной ситуации и требований проекта.

1. **Стандарты кодирования. Описание, область применения, состав стандартов кодирования, примеры стандартов.**

**Стандарты кодирования** – это набор правил и рекомендаций, которые определяют, каким образом следует писать и оформлять программный код. Целью таких стандартов является улучшение читаемости, поддерживаемости и качества программного кода, а также обеспечение единообразия в разработке программного обеспечения.

**Область применения стандартов кодирования включает** различные языки программирования, такие как Java, C++, Python, JavaScript и другие. Каждый язык программирования может иметь свои собственные стандарты кодирования, которые определяются сообществом разработчиков данного языка.

**Состав стандартов кодирования обычно включает в себя следующие аспекты:**

1. Оформление и структура кода: отступы, расстановка фигурных скобок, использование пробелов и табуляции и т.д.

2. Именование переменных, функций, классов и других элементов кода.

3. Использование комментариев и документации.

4. Обработка ошибок и исключений.

5. Использование стандартных библиотек и API.

**Примеры стандартов кодирования включают** Google Java Style Guide для языка Java, PEP 8 для языка Python, Airbnb JavaScript Style Guide для JavaScript и многие другие.

1. **Опишите основные стратегии разработки программных средств и систем.**

**Основные стратегии разработки программных средств и систем включают в себя:**

1. Водопадная модель - это классическая стратегия разработки, которая предполагает последовательное выполнение этапов анализа, проектирования, разработки, тестирования и внедрения. Каждый этап зависит от завершения предыдущего, и изменения требований в процессе разработки могут быть затруднительны.

2. Итеративная модель - в этой стратегии разработка происходит через серию коротких циклов, называемых итерациями. На каждой итерации создается рабочий прототип или версия программного продукта, которая затем улучшается и дополняется на следующих этапах.

3. Инкрементальная модель - в этой стратегии разработка происходит путем поэтапного добавления новых функций и возможностей к уже существующему программному продукту. Каждый инкремент представляет собой новую версию программы с улучшенным функционалом.

4. Agile - это гибкая стратегия разработки, которая акцентирует внимание на быстрой поставке рабочего программного обеспечения, постоянном взаимодействии с заказчиком и готовности к изменениям требований в процессе разработки.

5. DevOps - это стратегия, объединяющая разработку (Development) и операции (Operations), с целью автоматизации процессов разработки, тестирования, поставки и обслуживания программного обеспечения.

1. **Диаграммы UML. Назначение, виды диаграмм, преимущества использования.**

**UML-диаграмма** — это схема, нарисованная с применением символов UML. Она может содержать множество элементов и соединений между ними.

**Диаграммы UML (Unified Modeling Language)** являются стандартным набором графических обозначений, используемых для моделирования программных систем. Они предоставляют разработчикам инструменты для визуализации, спецификации, построения и документирования архитектуры программных систем.

**Назначение диаграмм UML включает в себя следующее:**

1. Визуализация: диаграммы UML позволяют визуализировать различные аспекты программной системы, такие как ее структура, поведение, взаимодействие и т.д.

2. Спецификация: они помогают формализовать требования к системе и определить ее функциональность.

3. Построение: диаграммы UML используются для построения архитектуры программных систем, определения их компонентов и связей между ними.

4. Документирование: они служат для документирования процесса разработки и архитектуры системы.

**Виды диаграмм UML включают в себя:**

1. Диаграммы классов: используются для моделирования структуры классов и их отношений в системе.

2. Диаграммы вариантов использования: показывают различные сценарии использования системы.

3. Диаграммы последовательностей: отображают последовательность взаимодействия объектов в системе.

4. Диаграммы активностей: используются для моделирования бизнес-процессов или последовательности действий в системе.

5. Диаграммы состояний: показывают различные состояния объектов в системе и переходы между ними.

**Преимущества использования диаграмм UML включают в себя:**

1. Улучшенное понимание системы: они помогают разработчикам и заинтересованным сторонам лучше понять архитектуру и функциональность системы.

2. Улучшенная коммуникация: они служат важным инструментом коммуникации между членами команды разработки и заказчиками.

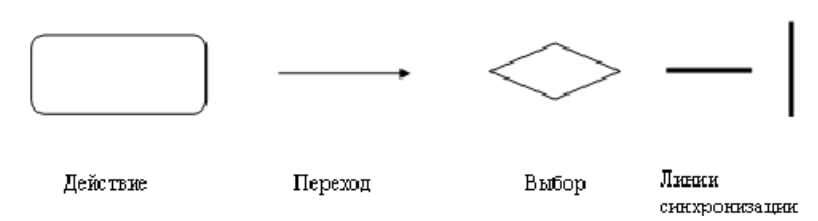
3. Стандартизация: использование стандартных графических обозначений упрощает понимание и интерпретацию диаграмм для всех заинтересованных сторон.

4. Повышение качества разработки: они помогают выявить проблемы и ошибки на ранних этапах разработки, что способствует улучшению качества программного обеспечения.

1. **Диаграмма Деятельности. Понятие, основные элементы, особенности, этапы разработки.**

**Диаграмма деятельности** - это своеобразная блок-схема, которая описывает последовательность выполнения операций во времени. Их можно использовать для моделирования динамических аспектов поведения системы.

**В диаграммах деятельности используются пиктограммы** "действие", "переход", "выбор" и "линии синхронизации". В языке UML действие изображается в виде прямоугольника с закругленными углами, переходы - в виде направленных стрелок, элементы выбора - в виде ромбов, линии синхронизации - в виде горизонтальных и вертикальных линий.



**Особенности диаграммы деятельности:**

1. Подходит для описания сложных процессов с большим количеством действий и ветвлений.

2. Позволяет увидеть последовательность и параллельность действий.

3. Необходимо явно указывать все возможные пути и условия переходов между действиями.

**Этапы разработки диаграммы деятельности:**

1. Идентификация цели моделирования и определение актеров и основных действий.

2. Определение последовательности действий и их условий выполнения с помощью ветвлений и решений.

3. Размещение действий на диаграмме и связывание их стрелками для обозначения порядка выполнения.

4. Добавление расхождений и слияний для ветвления и объединения процессов.

5. Проверка диаграммы на корректность и соответствие требованиям системы.

1. **Диаграмма Вариантов использования. Понятие, основные элементы, особенности, этапы разработки.**

**Диаграмма вариантов** использования является исходной моделью, с которой начинается процесс моделирования на языке UML. Она описывает функциональное назначение системы в самом общем виде с точки зрения всех ее пользователей и заинтересованных лиц. Диаграмма вариантов использования пре6дставляет собой диаграмму, на которой изображаются варианты использования проектируемой системы, как правило, заключенные в границу субъекта и внешние актеры, а также определенные взаимоотношения между актерами и вариантами использования.

**Основные элементы диаграммы Вариантов использования:**

1. Актеры: пользователи или внешние системы, взаимодействующие с системой.

2. Варианты использования (Use Case): функциональные возможности, которые предоставляются системой.

3. Ассоциации: связи между актерами и вариантами использования, обозначающие вовлечение актеров в определенные сценарии использования.

**Особенности диаграммы Вариантов использования:**

1. Простота: диаграмма предоставляет наглядное представление функционала системы и ее взаимодействия с актерами.

2. Системная ориентация: диаграмма фокусируется на том, как система служит потребностям актеров.

3. Гибкость: диаграмма может быть изменена и дополнена на разных этапах разработки системы.

**Этапы разработки диаграммы Вариантов использования:**

1. Идентификация актеров: определение пользователей или систем, взаимодействующих с системой.

2. Определение вариантов использования: выделение функциональных возможностей, которые должна предоставлять система.

3. Установление связей: определение связей между актерами и вариантами использования.

4. Документирование и анализ: создание документации на базе диаграммы, анализ и обсуждение с заинтересованными сторонами.

5. Итеративное уточнение: диаграмма может быть дополнена и изменена в процессе разработки и тестирования системы.

1. **Диаграмма Последовательностей. Понятие, основные элементы, особенности, этапы разработки.**

**Диаграмма последовательностей** - это графическое представление взаимодействия различных элементов и объектов в системе в рамках конкретной последовательности действий или сценария. Она позволяет визуализировать обмен сообщениями между объектами и последовательность выполнения операций.

**Основные элементы диаграммы последовательностей:**

1. Объекты - представляют собой элементы системы, которые взаимодействуют друг с другом.

2. Временная ось - показывает порядок выполнения операций и время, затраченное на каждую операцию.

3. Линии жизни - представляют объекты и показывают, когда они существуют и взаимодействуют.

4. Сообщения - стрелки, указывающие направление передачи информации между объектами. Они могут быть синхронными (жирные стрелки), асинхронными (пунктирные стрелки), ответами и т. д.

**Особенности диаграммы последовательностей:**

1. Представляет собой динамическое взаимодействие объектов в системе.

2. Позволяет увидеть последовательность выполнения операций и взаимодействие между различными элементами системы.

3. Часто используется для моделирования и проектирования систем в объектно-ориентированном программировании.

**Этапы разработки диаграммы последовательностей:**

1. ***Определение цели и контекста*** - определение целей, которые должна выполнять диаграмма, и анализ контекста, в котором она должна использоваться.

2*.* ***Определение объектов и взаимодействия*** - определение объектов, которые будут участвовать в диаграмме, и их взаимодействия. Это может включать определение операций, вызываемых объектами, и передаваемых сообщений.

3. ***Построение диаграммы*** - создание графического представления, включая объекты, линии жизни и сообщения.

4*.* ***Проверка и анализ*** - проверка диаграммы на наличие ошибок, неполадок или неоднозначностей. Анализ и интерпретация диаграммы для получения нужной информации.

5. ***Документирование*** - документирование диаграммы и ее использование в рамках проекта или разработки системы.

1. **Диаграмма Развертывания. Понятие, основные элементы, особенности, этапы разработки.**

**Диаграмма развертывания UML (Deployment diagram)** представляет собой визуальное представление аппаратного и программного обеспечения, включая физическое расположение компонентов системы, их взаимодействие и конфигурацию.

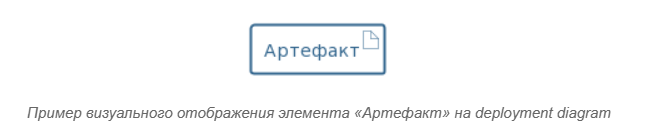
**Целью диаграммы развертывания является** описание архитектурного размещения компонентов системы на различных узлах (например, серверах, компьютерах и других вычислительных ресурсах) и связей между ними.

**Элементы диаграммы развертывания:**

1. ***Узел на диаграмме развертывания UML представляет*** собой важный элемент, используемый для описания физической инфраструктуры и распределения компонентов системы. Он представляет собой физическое устройство или вычислительный ресурс, на котором развертываются компоненты системы. Узлы на диаграмме развертывания обычно представляются в виде иконок или прямоугольников с названием и другими атрибутами, позволяющими идентифицировать устройство и его характеристики.

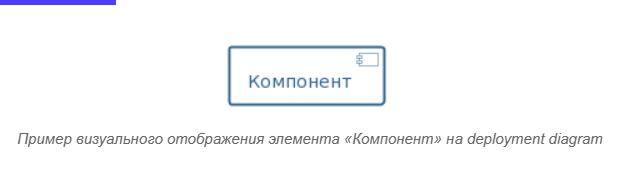
**

1. ***Артефакты представляют*** собой существенные элементы, играющие важную роль на диаграмме развертывания UML. Они представляют собой физические или логические компоненты, которые развертываются и существуют в системе. Артефакты на диаграмме развертывания включают в себя файлы, библиотеки, исполняемые программы, конфигурационные файлы, базы данных, документацию и другие элементы, необходимые для развертывания и функционирования системы.

**

1. ***Компонент на диаграмме развертывания UML представляет*** собой ключевой элемент, используемый для моделирования и описания функциональных частей системы, которые могут быть развернуты на узлах среды выполнения. Он представляет собой независимую единицу программного обеспечения, которая имеет четко определенные функции и интерфейсы.

***Компоненты представляются как*** прямоугольники с названием и другими атрибутами, которые помогают идентифицировать и описывать компонент. Эти атрибуты могут включать имя компонента, его тип, версию, используемые интерфейсы и другие характеристики, которые необходимы для понимания его функциональности и связей с другими компонентами.



1. ***Отношения (Связи)*** а диаграмме развертывания UML представляют собой ассоциации и зависимости между элементами системы. Они имеют важное значение при моделировании и анализе ее архитектуры. Они помогают определить связи и взаимодействия между компонентами, узлами среды выполнения и другими сущностями системы.
2. **Диаграмма Состояний. Понятие, основные элементы, особенности, этапы разработки.**

**Диаграмма состояний (State diagram)** — это один из видов диаграмм UML, используемых в разработке программного обеспечения, чтобы визуализировать и моделировать поведение объекта или системы в различных состояниях. Она позволяет описать все возможные состояния объекта, а также переходы между ними в ответ на определенные события.

**Компоненты диаграммы состояния:**

1. ***Начальное состояние*** — это состояние, с которого начинается выполнение процесса. Оно обозначается с помощью символа заполненного кружка, откуда исходят стрелки, представляющие переходы в другие состояния
2. ***Конечное состояние*** — это состояние, в котором процесс завершает свое выполнение. Оно обозначается обычно с помощью символа закрытого кружка. Конечное состояние указывает на завершение объекта или системы и обозначает, что дальнейшее выполнение или процесс достиг своего завершения.
3. ***Состояние (State)*** в диаграмме состояний UML представляет определенное состояние объекта или системы, которое может изменяться в ответ на определенные события, условия или действия. Оно определяет его поведение и свойства в определенный момент времени.
4. ***Составное состояние в диаграмме состояний UML представляет*** собой состояние, которое содержит внутренние состояния и переходы между ними. Оно позволяет структурировать более сложные состояния и их поведение внутри объекта или системы.
5. ***Защитное условие представляет*** собой логическое выражение, которое определяет условия, при которых может произойти переход из одного состояния в другое. Оно является частью перехода и помогает управлять потоком выполнения между состояниями.

**Этапы разработки диаграммы состояния:**

1. ***Определим объект или систему.***

Определим для какого объекта или системы вы создаете диаграмму состояний.

1. ***Выделите состояния.***

Определим все возможные состояния объекта или системы, которые имеют значимое поведение.

1. ***Определите переходы.***

Опишите переходы между состояниями, указывая события или условия, которые инициируют переход.

1. ***Укажите действия.***

Определите действия или операции, которые выполняются при переходе между состояниями.

1. ***Поддержите четкость и простоту.***

Старайтесь поддерживать диаграмму состояний простой, понятной и легко читаемой.

Выделите основные состояния и переходы, которые наиболее важны для понимания поведения объекта или системы.

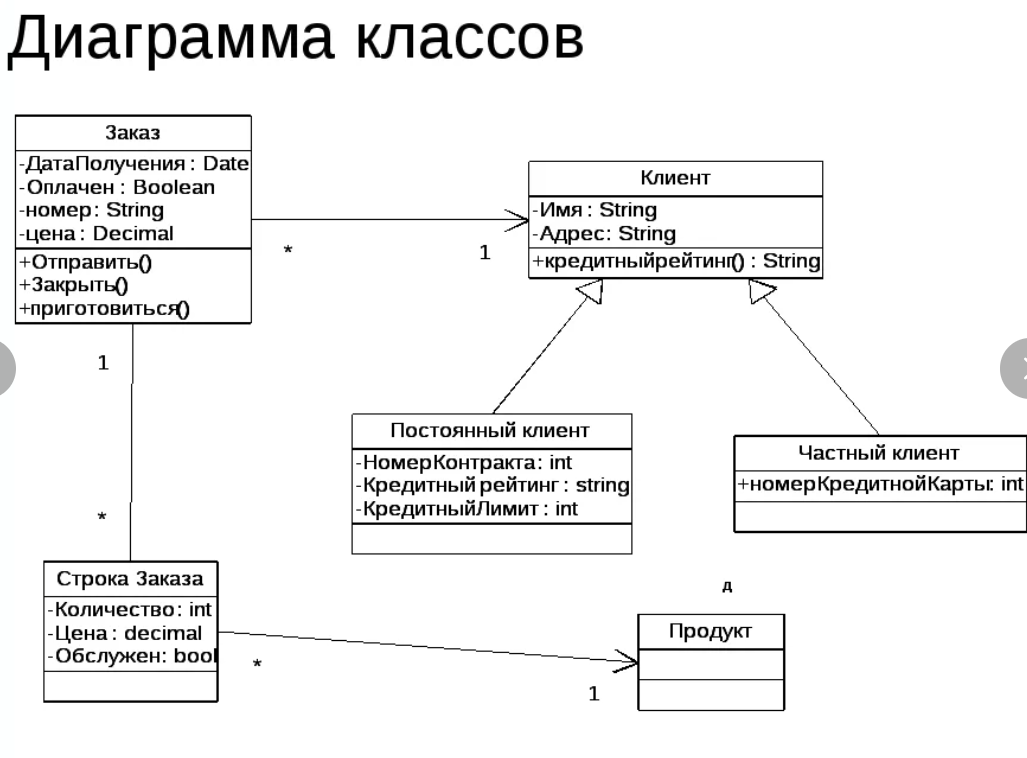
1. ***Документирование и комментируйте.***

Предоставьте достаточное количество комментариев и документации для объяснения основных аспектов модели.

1. **Диаграмма Классов. Понятие, основные элементы, особенности, этапы разработки.**

**Класс (class)** - категория вещей, которые имеют общие атрибуты и операции.

**Диаграмма классов в Unified Modeling Language** — это диаграмма статической структуры, демонстрирующая свойства системы, классы, операции и отношения между объектами для описания структуры системы.



**Компоненты диаграммы классов:**

1. ***Верхняя секция*** - включает имя класса. Независимо от того, обсуждаете ли вы классификатор или объект, этот раздел всегда необходим.
2. ***Средняя часть*** - содержит атрибуты класса. Опишите характеристики класса в этом разделе. Это требуется только при описании конкретного экземпляра класса.
3. ***Нижняя часть*** - содержит операции класса. Он показывает, как данные взаимодействуют с классом.
4. **Диаграмма Потоков данных. Понятие, основные элементы, особенности, этапы разработки.**

**DFD (от англ. data flow diagrams)** — это графическое представление потока данных в информационной системе. С его помощью можно описывать входящие и выходящие потоки данных и хранилища этих данных.

**Нотации DFD (Диаграмма потоков данных):**

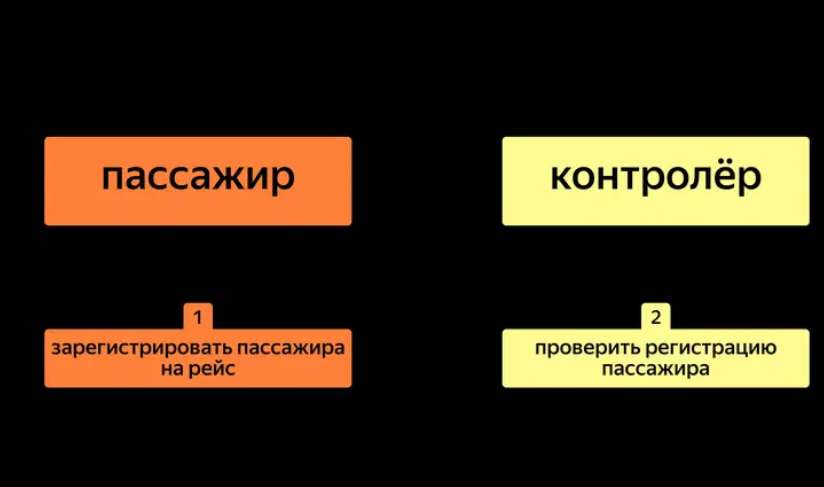
1. ***Процесс*** - активность, которая приводит к преобразованию данных. Например, в процессе оплаты заказа в интернет-магазине человек вводит данные банковской карты, а в ответ получает статус оплаты.
2. ***Внешние сущности*** - это участники процесса, которые взаимодействуют с данными. Например, оплата заказа в интернет-магазине связана с сущностями «пользователь» и «банк»
3. ***Хранилище данных*** - место, где хранится информация для последующего использования системой. Например, базы данных пользователей.
4. ***Потоки данных*** - маршруты, по которым информация перемещается между внешними сущностями, процессами и хранилищами данных.

**Этапы разработки диаграммы потоков данных:**

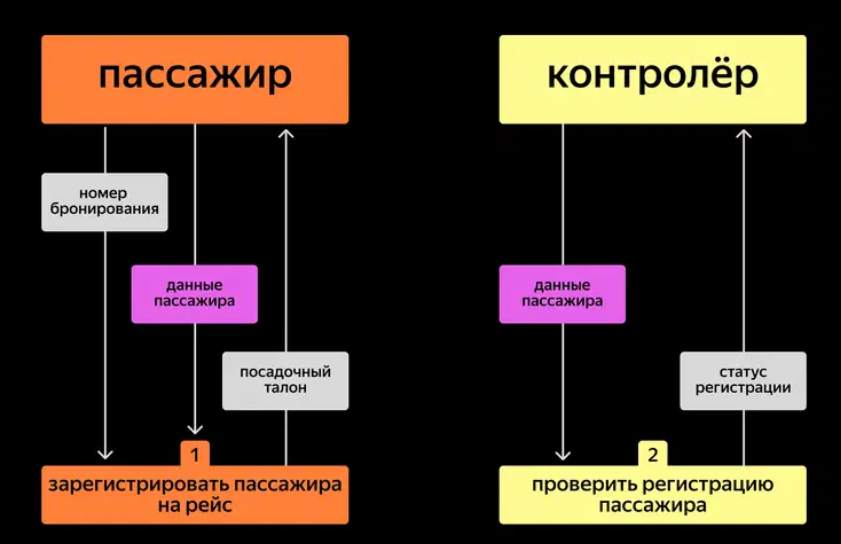
1. ***Выделить сущности*** - нужно определить все сущности, которые используют систему.



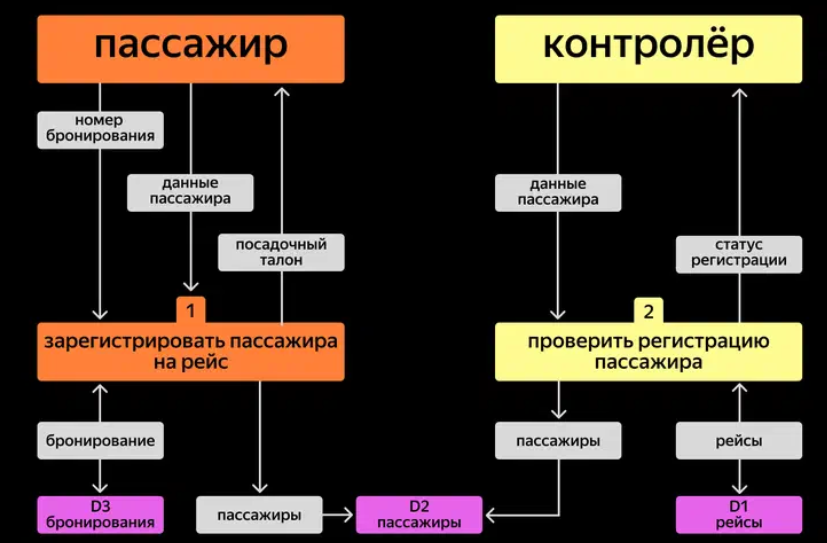
1. ***Определить процессы*** - пассажиры используют систему, чтобы регистрироваться на рейс. А контролёры используют систему, чтобы проверить статус регистрации каждого пассажира.



1. ***Указать потоки данных между сущностями и процессами*** - для того чтобы зарегистрироваться на рейс, пассажир должен предоставить номер бронирования и свои данные. Например, ФИО, серию и номер паспорта. Стрелки с этими данными идут от сущности в процесс. После регистрации на рейс пассажир получает посадочный талон. Эти данные уже направляются от процесса к сущности.



1. ***Определить хранилища данных и потоки к ним*** - какие данные получает система на входе и какие данные она должна отдать на выходе, на диаграмму нужно добавить хранилища. В них будут направляться данные, там они будут храниться и извлекаться по требованию пользователя. Вся информация, которая нужна системе для выполнения двух процессов, сосредоточена в трёх хранилищах: «Бронирования», «Пассажиры» и «Рейсы».



1. **Диаграмма Компонентов. Понятие, основные элементы, особенности, этапы разработки.**

**Диаграмма компонентов (component diagram)** — это одна из структурных диаграмм UML, которая описывает структуру системы и взаимосвязи между ее компонентами. Она предоставляет визуальное представление компонентов системы и позволяет лучше понять, как они взаимодействуют друг с другом.

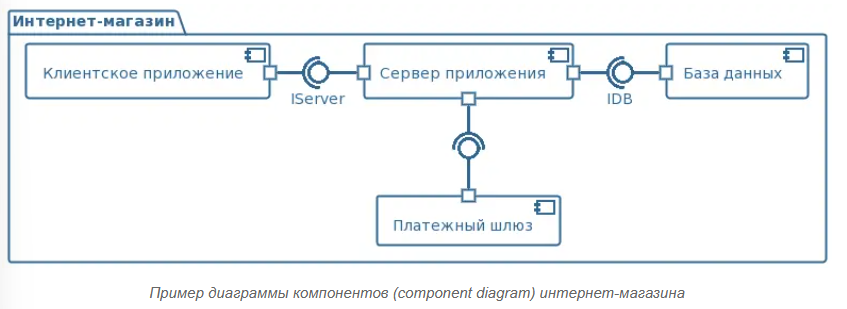
Диаграмма компонентов состоит из блоков, представляющих компоненты системы, и стрелок, обозначающих связи между ними. Компонентами могут быть модули, классы, библиотеки, пакеты и другие логические или физические элементы, из которых состоит система и которые взаимодействуют друг с другом для выполнения определенных функций системы.

**Элементы диаграммы компонентов:**

1. ***Компонент*** - представляет отдельную часть системы, которая выполняет определенную функцию или имеет определенную роль. Он может быть программным модулем, классом, библиотекой, сервисом, физическим устройством и т. д. Он обычно отображается в виде прямоугольника с его именем или обозначением.
2. ***Предоставляемый интерфейс*** - на диаграмме компонентов указывает на то, какие возможности или функции предоставляет конкретный компонент для взаимодействия с другими компонентами или системами. Он определяет способ, по которому другие компоненты могут использовать функциональность этого компонента или обмениваться данными с ним.
3. ***Требуемый интерфейс*** - на диаграмме компонентов указывает на интерфейсы, которые компонент ожидает от других компонентов. Он определяет, какие операции, методы или данные должны быть предоставлены другими компонентами, чтобы компонент мог взаимодействовать с ними.
4. ***Порт (Port)*** - представляет точку входа или выхода компонента, через которую он взаимодействует с другими компонентами. Он указывает на интерфейс компонента, через который он предоставляет или получает данные, вызывает методы или осуществляет обмен сообщениями.
5. ***Зависимость*** - указывает на связи между компонентами, когда один компонент зависит от другого. Зависимость это упрощенное отображения отношения ball-and-socket (предусмотренный интерфейс и требуемый интерфейс) между компонентами. Она обозначается пунктирной линией со стрелкой на конце. Стрелка указывает на направление зависимости.

**Построение диаграммы компонентов:**

1. ***Определение предметной области.***
2. ***Выделение компонентов*** - этот этап помогает определить, какие элементы системы будут представлены на диаграмме и какие функциональные части системы будут учтены.
3. ***Определение интерфейса.***
4. ***Размещение компонентов на диаграмме*** - этот шаг включает в себя размещение компонентов, выделенных на предыдущих этапах, на диаграмме компонентов.
5. ***Добавление портов и их связей -*** это добавление портов и связей уточняет, как компоненты взаимодействуют через интерфейсы, и делает диаграмму более информативной и понятной для разработчиков и аналитиков.



1. **Опишите модели быстрой разработки приложений. Приведите примеры.**

**Модель быстрой разработки приложений (Rad)** - это модель разработки, которая способствует быстрому созданию прототипов и немедленной обратной связи по сравнению с длительными, затяжными циклами разработки и тестирования.

**Модели быстрой разработки приложений (RAD)** - это методы разработки программного обеспечения, которые акцентируют внимание на быстрой поставке рабочего продукта и вовлечении заказчика в процесс разработки.

**Примеры моделей быстрой разработки приложений включают в себя:**

1. Agile - методология, которая акцентирует внимание на постоянном взаимодействии с заказчиком, быстрой поставке рабочего программного обеспечения и готовности к изменениям требований в процессе разработки. Примером может быть методология Scrum, которая предполагает работу в коротких итерациях (спринтах) и постоянное обновление итерации на основе обратной связи от заказчика.

2. Прототипирование - это метод, при котором создается рабочий прототип программного продукта, который затем улучшается и дополняется на следующих этапах. Примером может быть создание прототипа мобильного приложения для демонстрации его функционала и дизайна заказчику перед началом полноценной разработки.

3. Spiral model - это модель разработки, которая комбинирует элементы водопадной модели и прототипирования, позволяя быстро создавать рабочие версии продукта и постепенно улучшать его на основе обратной связи. Примером может быть разработка сложной системы управления проектами с постоянным добавлением новых функций и возможностей.

1. **Оценка качества программных средств. Назовите основные методики, приведите примеры регламентирующей нормативной документации.**

**Оценка качества программных средств может проводиться с помощью различных методик, включая:**

1. Тестирование - это процесс проверки программного обеспечения на соответствие заданным требованиям и выявление дефектов. Примеры методик тестирования включают в себя модульное тестирование, интеграционное тестирование, системное тестирование, приемочное тестирование и др.

2. Аудит кода - это процесс проверки и анализа программного кода для выявления потенциальных проблем, ошибок или нарушений стандартов программирования.

3. Использование метрик качества - это методика, основанная на измерении различных характеристик программного обеспечения, таких как производительность, надежность, поддерживаемость и др.

1. **Опишите цели и задачи тестирования программного продукта.**

**Цели и задачи тестирования** — основной задачей тестирования ПО является получение информации о статусе готовности заявленной функциональности системы или приложения.

**Задачи** — повысить вероятность того, что приложение, предназначенное для тестирования, будет работать правильно при любых обстоятельствах, будет соответствовать всем описанным требованиям.

**Цели тестирования:**

1. ***Обнаружение дефектов***
2. ***Повышение уверенности в уровне качества***
3. ***Предоставление информации для принятия решений***
4. ***Предотвращение дефектов***
5. **Опишите основные виды тестирования программного продукта.**

**Основные методы тестирования:**

1. ***Функциональное тестирование*** - этот тип тестирования оценивает функциональность или поведение программного приложения на основе требований и спецификаций приложения.
2. ***Модульное тестирование*** - этот тип функционального тестирования проводится на уровне модуля или компонента. Целью модульного тестирования является тестирование отдельных модулей кода (или юнитов), чтобы убедиться, что они функционируют правильно.
3. ***Тестирование в белом ящике*** – включает в себя тестирование внутренней работы программного приложения, включая код, структуру и дизайн.
4. ***Интеграционное тестирование*** - этот тип функционального тестирования исследует взаимодействие между различными модулями или компонентами программного приложения. Цель здесь - убедиться, что компоненты работают правильно при интеграции, и это может выполняться на разных уровнях.
5. ***Инкрементное тестирование*** - включает в себя интеграцию компонентов по одному и тестирование их по мере интеграции. Это помогает выявлять ошибки на ранних стадиях процесса разработки, поскольку вы можете определить, правильно ли работает компонент, прежде чем интегрировать его с другими компонентами.
6. ***Тестирование в «сером ящике» -*** включает в себя тестирование программного приложения с некоторыми знаниями о внутренних функциях и процессах.
7. ***Системное тестирование -*** этот тип функционального тестирования оценивает все программное приложение в целом.
8. ***Тестирование черного ящика -*** этот вид тестирования программного обеспечения оценивает новые функции или изменения, чтобы проверить, работают ли они должным образом.
9. ***Стресс-тестирование -*** используется для проверки производительности приложений при экстремальных нагрузках или условиях.
10. **Опишите процесс разработки тестового сценария программного продукта.**

**Процесс разработки тестового сценария программного продукта включает в себя несколько этапов:**

1. Понимание требований. В первую очередь необходимо понять требования к программному продукту, которые были определены в спецификации или других документах. Это позволит определить, какие функции и возможности нужно протестировать.

2. Идентификация тестовых случаев. На основе требований определяются конкретные сценарии использования программного продукта, которые нужно протестировать. Это могут быть различные комбинации входных данных, варианты использования или сценарии ошибок.

3. Написание тестовых сценариев. Для каждого идентифицированного тестового случая разрабатывается соответствующий тестовый сценарий, который описывает последовательность шагов для проведения тестирования. Тестовый сценарий должен быть понятным, легко воспроизводимым и содержать информацию о входных данных, ожидаемых результатах и ожидаемом поведении программы.

4. Подготовка тестовых данных. Для проведения тестирования необходимо подготовить необходимые входные данные, которые будут использоваться при выполнении тестовых сценариев.

5. Выполнение тестовых сценариев. После подготовки тестовых сценариев и данных производится их выполнение, чтобы проверить работоспособность программного продукта и выявить возможные дефекты.

6. Анализ результатов. После выполнения тестовых сценариев производится анализ результатов, чтобы оценить соответствие программного продукта требованиям и выявить возможные проблемы.

1. **Приведите критерии оценки необходимого количества тестов программного продукта.**

**Критерий выбора тестов** – решающее правило выбора состава и содержания проверок при выполнении тестирования программного обеспечения.

**Требования к идеальному критерию тестирования:**

1.***Критерий должен быть достаточным.***

показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы;

2. ***Критерий должен быть полным.***

в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку;

3***. Критерий должен быть надежным.***

любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы;

4.***Критерий должен быть легко проверяемым.***

например, вычисляемым на тестах.

1. **Анализ требований разработки программного обеспечения. Определение, основные фазы, типы требований.**

**Анализ требований** – это важнейший этап процесса разработки программного обеспечения. Он включает в себя сбор, документирование и анализ требований к программному проекту для обеспечение его соответствия потребностям заинтересованных сторон и пользователей.

**Анализ требований разработки программного обеспечения включает в себя несколько этапов:**

1. Определение требований. На этом этапе проводится сбор информации о потребностях пользователей и бизнес-целях, которые должны быть удовлетворены программным продуктом. Это может включать в себя интервью с заказчиком, анализ документации и изучение рыночных трендов.

2. Анализ требований. После определения требований проводится их анализ с целью выявления противоречий, неоднозначностей или недостаточной информации. Также на этом этапе определяются приоритеты требований и их влияние на другие части системы.

3. Документирование требований. Полученные требования фиксируются в специальных документах, таких как техническое задание или спецификация требований. Это позволяет установить единый источник правды для всей команды разработки.

**Типы требований могут включать в себя** функциональные требования (описывающие функции, которые должен выполнять программный продукт), нефункциональные требования (касающиеся производительности, безопасности, удобства использования и т. д.) и бизнес-требования (связанные с бизнес-процессами и целями).

1. **Опишите основные стандарты качества программной документации.**

**Основные стандарты качества программной документации включают следующие аспекты:**

1. Полнота. Документация должна содержать все необходимые сведения о программном продукте, включая функциональные и нефункциональные требования, архитектуру системы, процессы развертывания и обслуживания, а также любые другие важные аспекты.

2. Однозначность. Документация должна быть написана ясно и понятно, чтобы избежать недопонимания и разночтений. Использование четкой терминологии и стандартных обозначений помогает обеспечить однозначность документации

3. Согласованность. Все разделы документации должны быть логически связаны между собой и не должны содержать противоречий. Это помогает избежать недоразумений и ошибок при разработке и использовании программного продукта.

4. Точность. Документация должна быть точной и соответствовать реальному состоянию программного продукта. Неверная или устаревшая информация может привести к ошибкам и проблемам при разработке и эксплуатации.

5. Удобство использования. Документация должна быть удобной в использовании и навигации, чтобы пользователи могли быстро найти необходимую информацию. Хорошо организованные разделы, содержание, индексы и ссылки помогают обеспечить удобство использования документации.

1. **Метрика программного обеспечения. Определение, классификация метрик, достоинства и недостатки использования метрик для оценки качества программного обеспечения.**

**Метрика программного обеспечения** - это инструмент для измерения и оценки различных аспектов качества программного обеспечения. Метрики могут быть использованы для оценки процессов разработки, качества кода, производительности, надежности, безопасности и других характеристик программного обеспечения.

**Метрики программного обеспечения могут быть классифицированы по различным критериям, таким как:**

1. Метрики процесса разработки - используются для оценки эффективности и качества процессов разработки программного обеспечения, таких как время разработки, затраты на разработку, количество ошибок в процессе разработки и другие.

2. Метрики качества продукта - оценивают качество программного обеспечения на основе его функциональности, надежности, производительности, безопасности и других характеристик.

3. Метрики кода - измеряют качество и характеристики программного кода, такие как сложность, объем, структура, читаемость и другие.

4. Метрики использования - оценивают удовлетворенность пользователей программного обеспечения, частоту использования, время отклика и другие параметры использования программы.

Среди достоинств можно выделить возможность объективной оценки качества программного обеспечения, выявление проблем и улучшение процессов разработки. Однако недостатками могут быть сложность выбора подходящих метрик, зависимость от контекста использования и риски искажения результатов из-за неправильного выбора метрик.

1. **Опишите процесс оценки качества программных средств с помощью метрик.**

**Процесс оценки качества программных средств с помощью метрик включает в себя несколько этапов:**

1. Определение целей оценки: на этом этапе определяются цели и задачи оценки качества программного обеспечения. Например, целью может быть улучшение процессов разработки, повышение надежности или улучшение пользовательского опыта.

2. Выбор подходящих метрик: на основе определенных целей выбираются подходящие метрики, которые позволят измерить и оценить нужные аспекты качества программного обеспечения. Например, для оценки процессов разработки могут использоваться метрики времени разработки, затрат на разработку, количество ошибок и т.д.

3. Сбор данных: на этом этапе собираются данные, необходимые для измерения выбранных метрик. Это может включать в себя анализ кода, отзывы пользователей, статистику использования программного обеспечения и другие данные.

4. Анализ и интерпретация результатов: полученные данные анализируются и интерпретируются с учетом поставленных целей. На основе результатов анализа принимаются решения о необходимости улучшения процессов разработки, исправлении ошибок или других мероприятиях.

5. Внедрение улучшений: на последнем этапе принятые решения о внедрении улучшений в процессы разработки или само программное обеспечение.

1. **Тестовое покрытие. Определение, критерии тестового покрытия в тестировании программного обеспечения.**

**1)Тестовое покрытие –** это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

**2)Тестовое покрытие** – это критерий, отображающий добротность тестирования. Характеризует полноту охвата тестами программного кода либо требований к нему.

**Критерии тестового покрытия определяют, какие аспекты системы должны быть протестированы и до какой степени. Некоторые из распространенных критериев тестового покрытия включают следующие:**

1. Покрытие операторов: этот критерий измеряет, сколько операторов программного обеспечения было испытано. Он оценивает, были ли проверены все операторы кода.

2. Покрытие условий: этот критерий определяет, до какой степени выражения и условия в программе были протестированы. Он оценивает, были ли проверены все возможные комбинации условий.

3. Покрытие пути: этот критерий определяет, насколько хорошо испытаны различные пути выполнения программы. Он оценивает, проходили ли тестовые случаи через все возможные пути исполнения в программе.

4. Покрытие функциональности: этот критерий оценивает, насколько хорошо тестовое покрытие проверяет все функциональные требования системы. Включает в себя проверку каждой функции и её альтернатив.

5. Покрытие граничных значений: этот критерий проверяет, насколько хорошо тесты проверяют крайние значения входных данных и граничные условия. Это помогает обнаружить возможные ошибки, связанные с некорректной обработкой таких значений в программе.

1. **Опишите процесс создания тестового сценария.**

**В тестовом сценарии** подробно расписана последовательность действий, которую выполняет тестировщик для проверки работоспособности ПО

**Разработка тестового сценария происходит в пять этапов:**

1. ***Понимание требований.*** Нужно чётко и полно собрать требования к программе: изучить спецификацию, обсудить результаты работы ПО, проанализировать пользовательские сценарии.

2. ***Определение цели.*** Нужно определить, что именно будет проверяться этим тестовым сценарием. Целью может быть как проверка функциональности, так и поиск конкретных ошибок.

3. ***Описание шагов.*** Написание непосредственно сценария — последовательности шагов, которые нужно выполнить, чтобы протестировать конкретный аспект программного продукта. Шаги должны быть ясными, понятными и охватывать все необходимые действия для достижения цели сценария.

4. ***Описание ожидаемых результатов.*** Для каждого шага необходимо прописать конкретные результаты, к которым он должен привести.

5. ***Документирование.*** Формирование из последовательности действий документа формата, принятого в компании.

1. **Опишите процесс создания тестового пакета.**

**Процесс создания тестового пакета можно разделить на следующие этапы:**

1. Определение целей и требований: Вначале необходимо определить цели тестирования и требования к тестовому пакету. Это может включать в себя определение функциональных и нефункциональных требований, описание основных потоков работы и сценариев использования.

2. Планирование и создание тестовых случаев: На этом этапе мы определяем конкретные тестовые случаи, которые будут включены в тестовый пакет. Каждый тестовый случай должен быть описан с указанием предусловий, шагов выполнения тестового случая и ожидаемых результатов.

3. Организация тестовых случаев в тестовый пакет: После создания тестовых случаев необходимо организовать их в тестовый пакет. Важно убедиться, что в тестовом пакете содержится достаточное покрытие функциональности, а также учитываются зависимости между тестовыми случаями — например, если один тестовый случай требует определенных предусловий, эти предусловия должны быть выполнены перед запуском данного тестового случая.

4. Подготовка тестового окружения: Для успешного выполнения тестовых случаев необходимо подготовить тестовое окружение. Это может включать в себя создание тестовых данных, установку и настройку необходимых программных и аппаратных средств, а также настройку тестируемой системы.

5. Выполнение тестового пакета: На этом этапе мы запускаем тестовый пакет и выполняем все тестовые случаи в нем. Важно записывать результаты выполнения каждого тестового случая — это позволит проанализировать результаты и выявить проблемы.

6. Анализ результатов и исправление дефектов: После выполнения тестового пакета необходимо анализировать результаты и исправлять выявленные дефекты. Отчет об ошибках и недостатках должен быть подготовлен и отправлен разработчикам для исправления.

7. Повторное выполнение тестового пакета: После исправления дефектов может потребоваться повторное выполнение тестируемого пакета для проверки исправлений и убеждения в правильной работе системы.

8. Документирование и архивирование результатов: В конце процесса создания тестового пакета необходимо документировать результаты тестирования, включая запись о выполненных тестовых случаях, найденных дефектах и общем статусе тестирования.

1. **Верификация программного обеспечения. Понятие, цель верификации, методы проверки.**

**Верификация программного обеспечения** – это процесс проверки и анализа, в ходе которых проверяется соответствие программного обеспечения своей спецификации и требованиям заказчиков.

**Верификация** проверяет соответствие ПО системной спецификации, в частности функциональным и нефункциональным требованиям.

**Две основные методики проверки и анализа систем:**

1. ***Инспектирование ПО.*** Анализ и проверка различных представлений системы, например документации спецификации требований, архитектурных схем или исходного кода программ. Инспектирование выполняется на всех этапах процесса разработки программной системы. Параллельно с инспектированием может выполняться автоматический анализ исходного кода программ и соответствующих документов. Инспектирование и автоматический анализ – это статические методы верификации и аттестации, поскольку им не требуется исполняемая система.
2. ***Тестирование ПО.*** Запуск исполняемого кода с тестовыми данными и исследование выходных данных и рабочих характеристик программного продукта для проверки правильности работы системы. Тестирование – это динамический метод верификации и аттестации, так как применяется к исполняемой системе.
3. **Аттестация программного обеспечения. Понятие, отличие от верификации, методики аттестации.**

**Аттестация программного обеспечения** – это процесс проверки и анализа, в ходе которых проверяется соответствие программного обеспечения своей спецификации и требованиям заказчиков.

***Верификация и аттестация не одно и то же, хотя их легко перепутать. Кратко различие между ними можно определить следующим образом:***

1. Верификация отвечает на вопрос, правильно ли создана система;
2. Аттестация отвечает на вопрос, правильно ли работает система.

**Две основные методики проверки и анализа систем:**

1. ***Инспектирование ПО.*** Анализ и проверка различных представлений системы, например документации спецификации требований, архитектурных схем или исходного кода программ. Инспектирование выполняется на всех этапах процесса разработки программной системы. Параллельно с инспектированием может выполняться автоматический анализ исходного кода программ и соответствующих документов. Инспектирование и автоматический анализ – это статические методы верификации и аттестации, поскольку им не требуется исполняемая система.
2. ***Тестирование ПО.*** Запуск исполняемого кода с тестовыми данными и исследование выходных данных и рабочих характеристик программного продукта для проверки правильности работы системы. Тестирование – это динамический метод верификации и аттестации, так как применяется к исполняемой системе.
3. **Инспекция программного кода на предмет соответствия стандартам кодирования. Понятие, виды, особенности.**
4. **Надежность программного обеспечения. Понятие, показатели надежности.**

**Надежность программного продукта** – способность программного продукта безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью.

**Показатели надежности:**

1. *Безотказность* – свойство программы выполнять свои функции во время эксплуатации.
2. *Работоспособность* -свойство программы корректно работать весь заданный период эксплуатации.
3. *Безопасность* – свойство программы быть не опасной для людей и окружающих систем.
4. *Защищённость* – свойство программы противостоять случайным или умышленным вторжениям на неё.
5. **Опишите способы обеспечения и повышения надежности программ.**

**Способы обеспечения и повышения надежности про­грамм могут быть разделены на следующие основ­ные категории:**

1. Усовершенствование технологии программирова­ния;
2. Выбор алгоритмов, не чувствительных к различно­го рода нарушениям вычислительного процесса (исполь­зование алгоритмической избыточности);
3. Резервирование программ — дуальное иN-версионное программирование, другие методы введения струк­турной избыточности;
4. Контроль при вводе по диапазону изменения входных данных;
5. Контроль за зацикливанием программ, возникновением самоблокировок и т.д.;
6. Контроль и тестирование программ с последующей коррекцией.
7. **Опишите понятие корректности и устойчивости программных систем.**

**Корректность и устойчивость** - два основных качества программной системы, без которых все остальные ее достоинства не имеют особого смысла.

**Корректность** - это способность программной системы работать в строгом соответствии со своей спецификацией. Отладка - процесс, направленный на достижение корректности.

**Устойчивость** - это способность программной системы должным образом реагировать на исключительные ситуации. Обработка исключительных ситуаций - процесс, направленный на достижение устойчивости.

1. **Опишите основные причины отказов программного обеспечения.**

**Причины отказов программного обеспечения:**

1. Основными причинами, непосредственно вызывающими нарушения нормального функционирования программы, являются:
2. ошибки, скрытые в самой программе;
3. искажения входной информации, подлежащей обработке;
4. неверные действия пользователя;
5. неисправности аппаратуры установки, на которой реализуется вычислительный процесс.

**Основные классы ошибок в программах:**

1. ***Ошибки вычислений.***

Ошибки содержатся в закодированных математических выражениях или в получаемых с их помощью результатах. Пример ошибок является неверное преобразование типов переменных, неверный знак операции, ошибка в выражении индекса.

1. ***Логические ошибки.***

Являются причиной искажения алгоритма решения задачи.

Ошибки возникают в связи с неверным заданием диапазона изменениям параметрам циклам, неверным условием.

3.***Ошибки ввода-вывода.***

Связано с действами, как управлением ввода-вывода, формирование выходных записей, определение размеров записей и др.

Примерами ошибок является неправильная форма ввода (вывода), отсутствие признака конца файла.

4.***Ошибки совместимости.***

Связаны с отсутствием совместимости ОС или с другими прикладными программами, используемой в данной программе.

5.***Ошибки сопряжений.***

Вызывают неверное взаимодействие программы с другими программами (подпрограммами), с системными программами, устройствами ЭВМ, входными данными и др.

1. **Качество программного продукта. Понятие, характеристики качества программного обеспечения.**

**Качество программного обеспечения** – это совокупность характеристик ПО, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

**Характеристики качества ПО:**

1. Функциональность – определяется способность ПО решать задачи, которые соответствует зафиксированным и предполагаемым потребностям пользователя, при заданных использованием ПО.
2. Надежность – способность ПО выполнять требуемые задачи в обозначенных условиях на протяжения заданного промежутка время или указанное количество операций.
3. Удобство использования – возможность лёгкого понимания, изучения, использования и привлекательность ПО для пользователя.
4. Эффективность – способность ПО обеспечивать требуемый уровень производительности соответствия с выделенными ресурсами, временем и другими обозначенными условиями.
5. Удобство сопровождения – легкость, с которой ПО может анализироваться, тестироваться, изменяться для исправления дефектов, для реализации новых требований, для облегчения дальнейшего обслуживания.
6. **Опишите основные критерии обеспечения легкости применения программного средства.**

Основные критерии обеспечения легкости применения программного средства включают:

1. Интуитивный интерфейс: Программное средство должно иметь понятный и удобный интерфейс пользователя, который позволяет легко освоиться с программой даже неопытным пользователям.
2. Простота использования: Программное средство должно быть простым в использовании без необходимости глубоких знаний технических аспектов или сложных действий для выполнения задач.
3. Ясная документация: Программа должна быть сопровождена подробной документацией, которая объясняет, как использовать программное средство и решить возможные проблемы.
4. Понятные сообщения и ошибки: Если в процессе работы возникают проблемы или ошибки, программное средство должно предоставлять четкие и понятные сообщения и подсказки, чтобы пользователь мог легко понять причину и исправить ошибку.
5. Консистентность: Программное средство должно использовать привычные и консистентные элементы интерфейса для выполнения задач, чтобы пользователи могли легко применить свои знания и опыт из других программ.
6. Гибкость: Программное средство должно быть гибким и настраиваемым, чтобы пользователи могли настроить его под свои собственные нужды и предпочтения.
7. Быстрый доступ к функциям: Программное средство должно предоставлять быстрый доступ к основным функциям и командам для удобства пользователей.
8. Эффективность: Программное средство должно быть эффективным в использовании ресурсов системы, таких как память и процессорное время, чтобы не создавать замедления и проблем со стабильностью работы.
9. Обратная связь: Программное средство должно предоставлять пользователю обратную связь о его действиях и результате операций, чтобы пользователь мог быть уверен в правильности выполненных операций.
10. Поддержка: Программное средство должно иметь доступную и отзывчивую службу поддержки, чтобы пользователи могли получить помощь при необходимости.
11. **Система программирования. Понятие, программные компоненты, примеры систем программирования. Описание компонентов дописать**

**Система программирования** — это система для разработки новых программ на конкретном языке программирования.

**Система программирования обычно включает в себя следующие компоненты:**

1. *Компилятор или интерпретатор.*

Компилятор — это особый вид транслятора, который переводит тексты с языка программирования высокого уровня.

Интерпретатор — это исполняемый файл, который поэтапно читает программу, а затем обрабатывает, сразу выполняя ее инструкции.

1. *Интегрированная среда разработки.*

Интегрированная среда разработки — это набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку

1. *Средства создания и редактирования текстов программ.*
2. *Библиотеки стандартных программ и функций.*
3. *Отладочные программы, помогающие находить и устранять ошибки.*
4. *Диалоговая среда.*
5. *Многооконный режим работы.*
6. *Мощные графические библиотеки.*
7. *Утилиты для работы с библиотеками.*
8. *Ассемблер.*
9. *Справочная служба.*

**Примеры систем программирования:**

1. Eclipse — свободная интегрированная среда разработки модульных кроссплатформенных приложений, которая часто используется как платформа для разработки расширений. Eclipse является платформой с особым фундаментом для построения и запуска интегрированных инструментов разработки сквозного программного обеспечения. В силу бесплатности и высокого качества, Eclipse во многих организациях является корпоративным стандартом для разработки приложений.
2. IntelliJ IDEA — мощная универсальная среда программирования, поддерживающая язык Java. Она позиционирует себя как умная и удобная система программирования для Java (и других языков) с поддержкой всех последних технологий и фреймворков. У этой среды разработки есть мощные аналитические возможности. Система включает в себя набор инструментов для изменения внутренней структуры программы, который позволяет быстро реорганизовывать исходные тексты. Дизайн среды ориентирован на продуктивность работы программистов и позволяет оптимизировать простые рутинные задачи, чтобы дать возможность специалистам сконцентрироваться на достижение функциональных целей.
3. Delphi — среда разработки прикладных программ, предназначенных для запуска в ОС Windows, MacOS, а также в мобильных операционных системах — iOS и Android. Delphi отличается простотой и может использоваться в учебных целях. Программы в Delphi пишутся на языке Object Pascal, который является преемником и развитием языка Turbo Pascal. Программа предназначена, в первую очередь, для разработки приложений в архитектуре клиент-сервер.
4. **Системное программное обеспечение. Понятие, классификация, примеры.**

**Системное программное обеспечение (СПО)** — совокупность программ с узкой специализацией, направленной на взаимосвязь между физическими устройствами, управление компьютером, поддержание его в рабочем состоянии. Занимается его разработкой системный программист.

**Основные функции системного ПО в информатике:**

1. Поддержание эффективной работы любой вычислительной машины.
2. Выполнение фоновых процессов работы с файловой системой, проверка на наличие вредоносных скриптов и вирусов.
3. Осуществление диагностики и предотвращение выхода из строя компьютера, ноутбука и другого цифрового устройства.
4. **Прикладное программное обеспечение. Понятие, классификация, примеры.**

**Прикладное программное обеспечение** — совокупность программных средств и сопроводительных документов, которые предназначены для решения задач разной сложности с помощью компьютера



**Примеры прикладного ПО:**

***К примеру, популярностью пользуются текстовые редакторы:***

1. Блокнот Windows;
2. AkelPad;
3. Notepad++;
4. KeyPad+;
5. Geany.

***Текстовые процессоры:***

1. Microsoft Word;
2. AbiWord;
3. WordPerfect.

***Электронные таблицы:***

1. Microsoft Excel;
2. LibreOffice Calc;
3. OpenOffice Calc.

***В качестве СУБД используют:***

1. MySQL;
2. PostgreSQL;
3. MS SQL Server.

***Для редактирования презентаций подходят:***

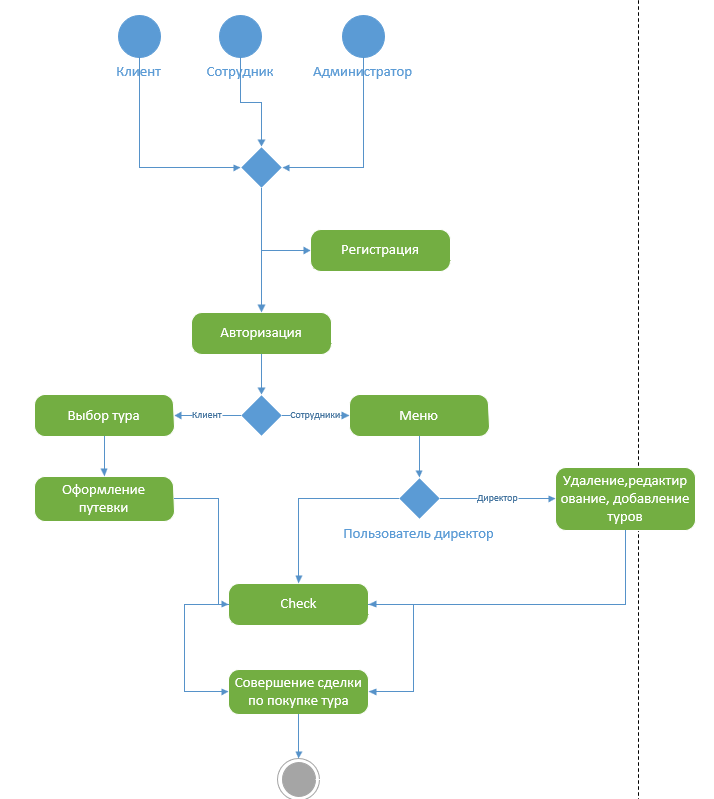
1. Microsoft PowerPoint;
2. Apple Keynote;
3. Google Slides;
4. LibreOffice Impress.

***Известными браузерами являются:***

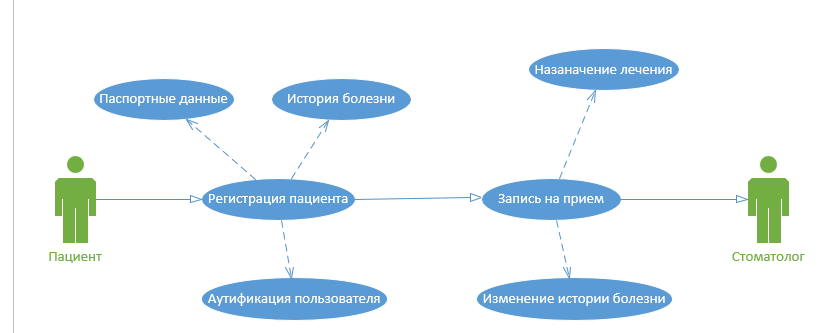
1. Internet Explorer;
2. Microsoft Edge;
3. Google Chrome;
4. Mozilla Firefox;
5. Opera;
6. Яндекс.Браузер;

***Практические задания:***

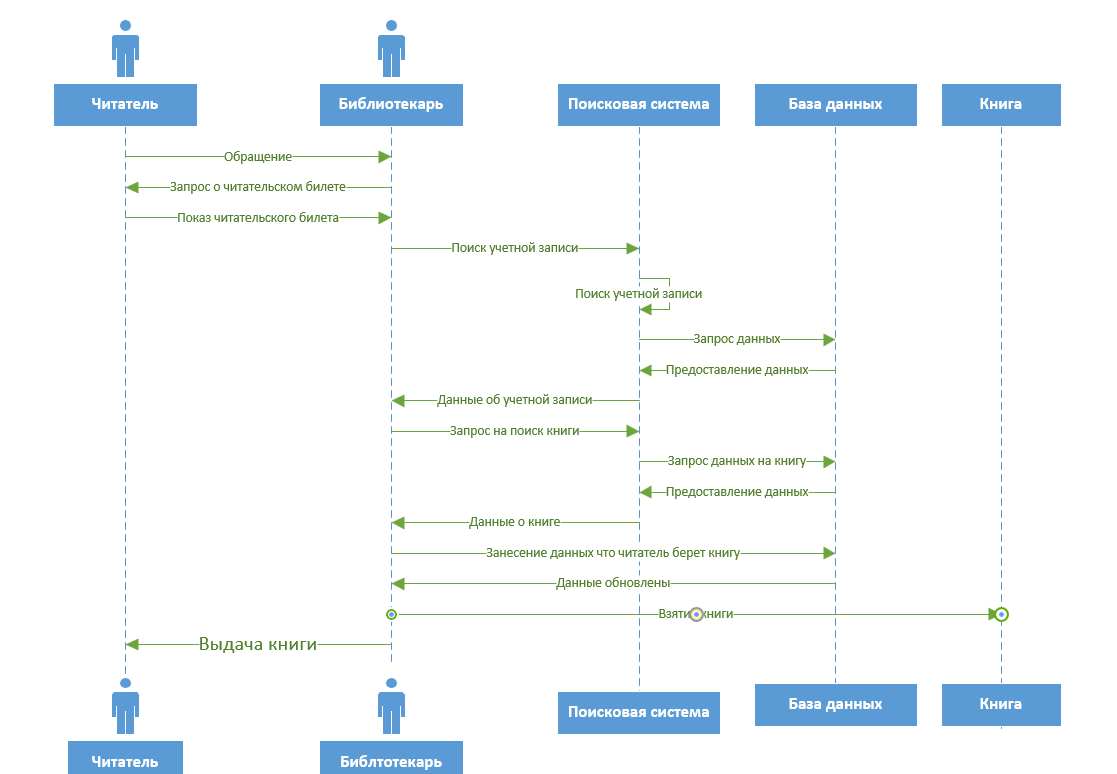
1. ***Создайте диаграмму Деятельности по описанию предметной области «Автоматизированная информационная система для туристического агентства».***

******

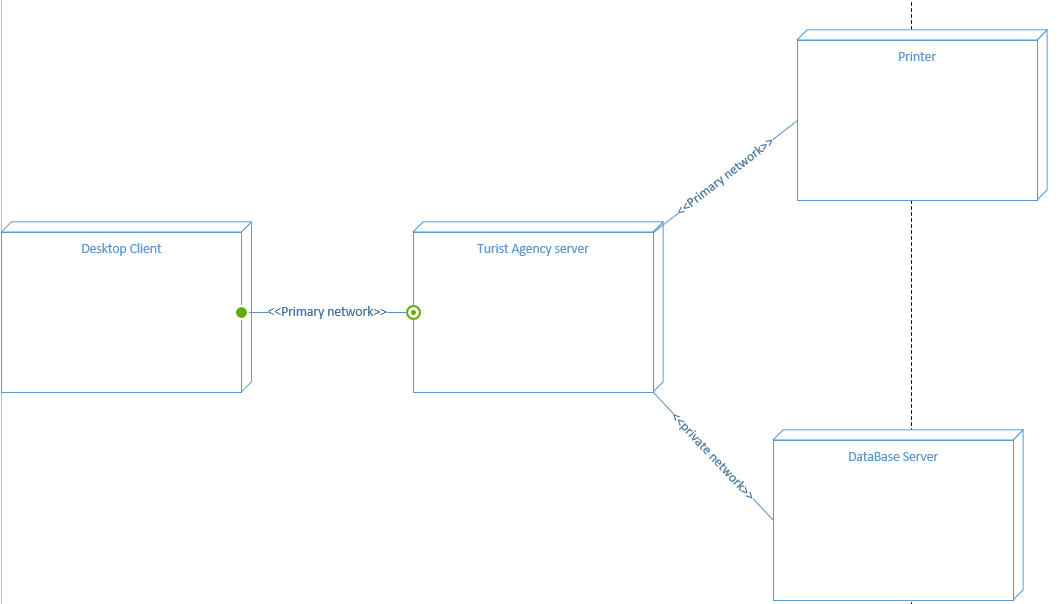
1. ***Создайте диаграмму Вариантов использования по описанию предметной области «Автоматизированная информационная система для стоматологической поликлиники»***

******

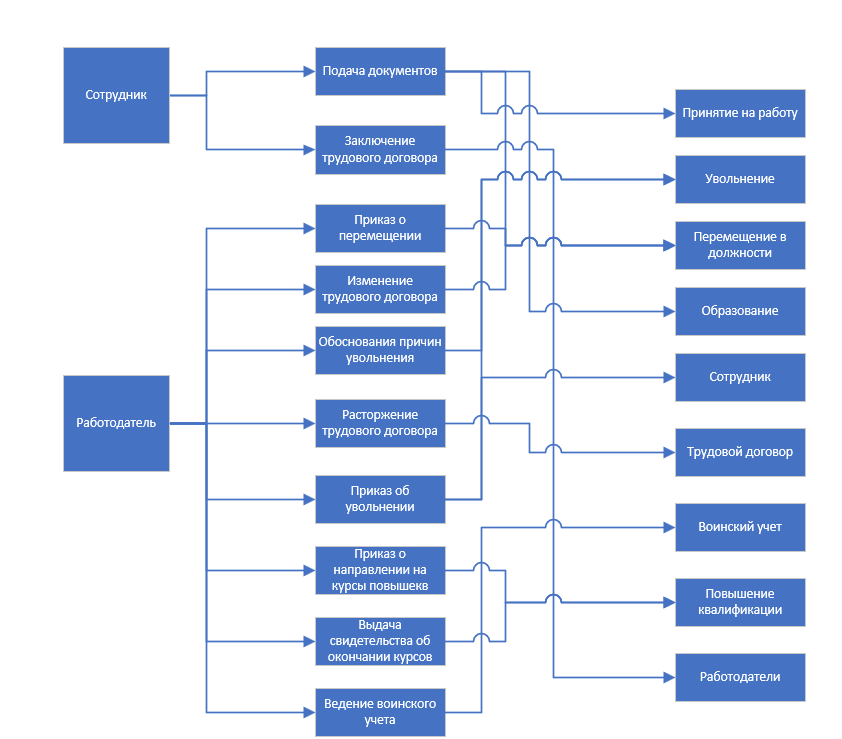
1. ***Создайте диаграмму Последовательностей по описанию предметной области «Автоматизированная информационная система для библиотеки»***

******

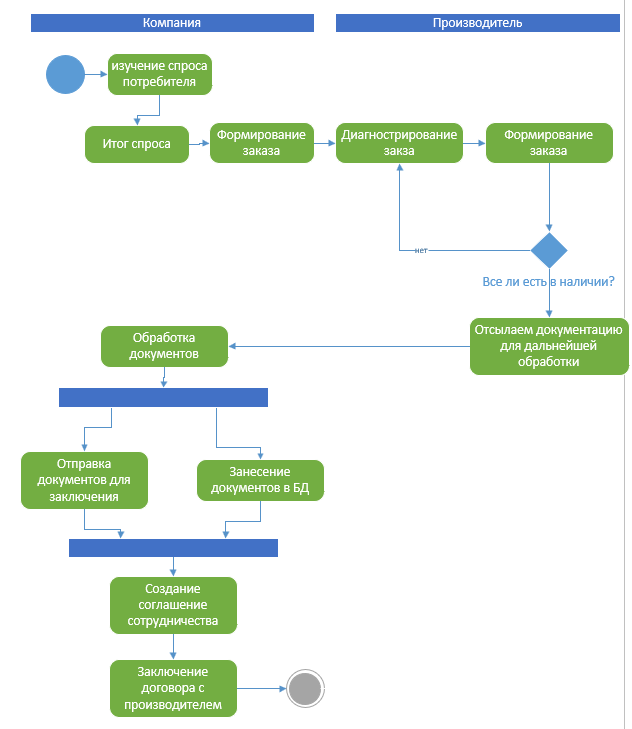
1. ***Создайте диаграмму Развертывания по описанию предметной области «Автоматизированная информационная система для туристического агентства».***

******

1. ***Создайте диаграмму Потоков данных по описанию предметной области «Автоматизированная информационная система для отдела кадров»***

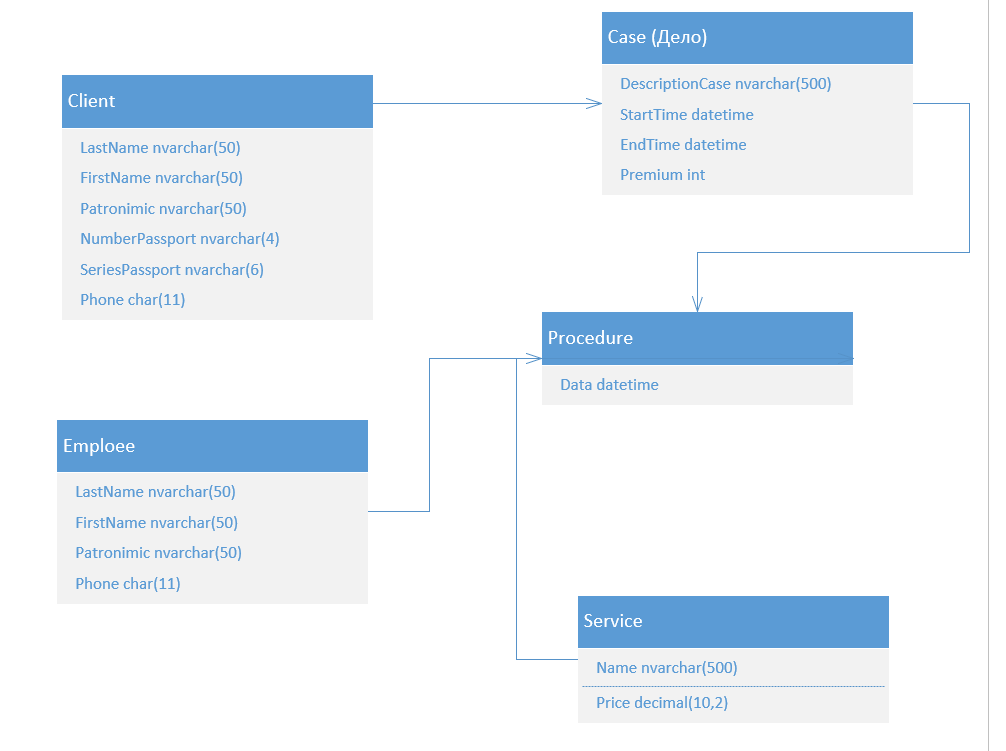
******

1. ***Создайте диаграмму Состояний по описанию предметной области «Автоматизированная информационная система для торговой базы»***

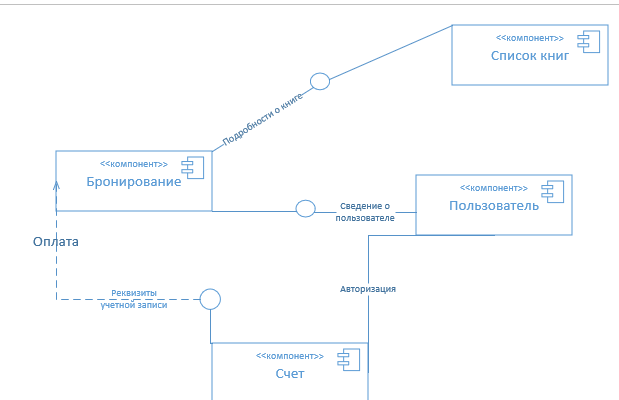
******

**Для описания заключения договора.**

1. ***Создайте диаграмму Классов по описанию предметной области «Автоматизированная информационная система для юридической фирмы»***

******

1. ***Создайте диаграмму Компонентов по описанию предметной области «Автоматизированная информационная система для книжного магазина»***

******

1. ***Разработайте класс для работы с турами в информационной системе «Туристического агентства». Реализуйте проверку правильности вводимых пользователем значений при добавлении нового тура.***
2. ***Разработайте класс для работы с книгами в информационной системе «Библиотеки». Реализуйте проверку правильности вводимых пользователем значений при добавлении новой книги.***
3. ***Разработайте страницу добавления товара для информационной системы «Торговая база». Реализуйте проверку на отсутствие пустых полей при добавлении нового товара.***
4. ***Разработайте окно для вывода списка товаров и цен к ним для приложения «Книжный магазин».***
5. ***Разработайте страницу записи на услугу в информационной системе «Салон красоты».***
6. ***Разработайте класс для работы с товарами в информационной системе «Магазин бытовой техники». Реализуйте проверку правильности вводимых пользователем значений.***
7. ***Разработайте класс для работы с украшениями в информационной системе «Ювелирный салон». Реализуйте проверку правильности вводимых пользователем значений при добавлении нового товара.***
8. ***Разработайте страницу добавления товара для информационной системы «Мебельный салон. Реализуйте проверку на отсутствие пустых полей при добавлении нового товара.***
9. ***Разработайте с вывода списка клиентов для информационной системы «Юридическая фирма».***
10. ***Разработайте окно для работы с услугами в информационной системе «Салона сотовой связи».***
11. ***Разработайте класс для хранения данных клиента в информационной системе «Фирма по оказанию бухгалтерских услуг».***
12. ***Разработайте страницу добавления нового товара для информационной системы «Магазина оргтехники».***
13. ***Разработайте страницу для добавления нового сотрудника в информационной системе «Отдел кадров». Реализуйте проверку на отсутствие пустых полей.***
14. ***Разработайте страницу для записи на лечение пациента в информационной системе «Лечебное медицинское учреждение.***
15. ***Разработайте страницу заказов рекламных баннеров в информационной системе «Рекламное агентство».***
16. ***Разработайте класс для хранения данных автомобилей в информационной системе «Автостоянка». Реализуйте проверку правильности вводимых пользователем значений при добавлении нового автомобиля.***
17. ***Разработайте страницу для записи на лечение пациента в информационной системе «Стоматологическая поликлиника».***
18. ***Разработайте страницу вывода списка спортивного инвентаря для информационной системы «Прокат спортивного оборудования».***
19. ***Разработайте класс для хранения данных водителя в информационной системе для Каршеринг. Реализуйте проверку правильности вводимых пользователем значений.***