**Теоретические вопросы:**

**1. Объясните разницу между каскадной (водопадной) и спиральной моделью разработки ПО.**

Каскадная модель или «водопад». Данная модель предполагает строго последовательное (во времени) и однократное выполнение всех фаз проекта с жестким (детальным) предварительным планированием.

Спиральная разработка — это семейство процессов разработки программного обеспечения, для которых характерно повторение набора элементарных процессов разработки и управление рисками, с целью уменьшения рисков.

повторяющуюся последовательность циклов разработки с непрерывным контролем рисков.

**2. Перечислите этапы разработки ПО и соответствующие им методики анализа и**

**проектирования.**

* Постановка и анализ задачи
* Проектирование
* Разработка
* Тестирование программы
* Документирование
* Сопровождение и эксплуатация

**Анализ и планирование**: Графический метод. Диаграммы сущность-связь (ER диаграммы), диаграммы потока данных (DFD), структурные схемы, деревья решений, метод SADT и пр.

Объектно-ориентированная методология. Объектный подход к разработке систем следует итеративному процессу с наращиванием возможностей. Единая модель конкретизируется на этапах **анализа, проектирования и реализации** – в результате успешных итераций добавляются новые детали, при необходимости вводятся изменения и усовершенствования. (UML) Диаграмма прецедентов использования (Use Case Diagram), Диаграммы взаимодействия (Interaction Diagram), Диаграмма деятельности (Activity Diagram), Диаграмма классов (Class Diagram).

**3. Дайте определение понятию CASE-средства. Приведите примеры.**

CASE-средства (Computer - Aided Software Engineering) - это методы и технологии, которые позволяют проектировать различные информационные системы (в частности, базы данных) и автоматизировать их создание.

CASE средства верхнего уровня. Эти CASE средства ориентированы на начальные этапы построения информационной системы. Они связаны с анализом и планированием. Все CASE средства верхнего уровня содержат графические инструменты построения диаграмм, таких как диаграммы сущность-связь (ER диаграммы), диаграммы потока данных (DFD), структурные схемы, деревья решений и пр.

CASE средства нижнего уровня. Эти CASE средства больше сфокусированы на последних этапах разработки информационной системы – проектирование, разработка программного кода, тестирование и внедрение. ERwin (Logic Works),

Интегрированные CASE средства (I – CASE). Эти CASE средства охватывают полный жизненный цикл разработки информационной системы. Они позволяют обмениваться данными между инструментами верхнего и нижнего уровня.

**4. Опишите назначение методологии IDEF0, основные составляющие графические обозначения**

**нотации, правила именования блоков, направление и назначение стрелок.**

IDEF0 — методология моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Данная модель позволяет описать все основные виды **процессов**, как административные, так и организационные. Функциональная модель IDEF0 представляет собой набор блоков со входами (вводные, которые ставят определенную задачу) и выходами (выводящие результат деятельности), управлением (механизмы управления (положения, инструкции)) и механизмами (что используется для того, чтобы произвести необходимую работу), которые детализируются (декомпозируются).

1. В левом верхнем углу всегда – главный элемент.
2. Управляющие стрелки называют именами существительными, блоки – глаголами
3. Все элементы должны иметь входящие и исходящие стрелки, так как для выполнения необходимо что-то получить на входе (заказ, поставленную задачу), а после обработки на выходе необходимо передать готовый продукт. Входящие стрелки всегда слева, исходящие – справа.
4. Сверху – управляющие элементы, снизу – механизмы, необходимые для выполнения процесса.
5. Если на одном листе (экране) располагается несколько блоков, каждый последующий располагается справа и ниже предыдущего.
6. Необходимо стремиться создавать схемы таким образом, чтобы пересечение стрелок было сведено к необходимому минимуму.

**5. Опишите назначение методологии DFD, основные составляющие графические обозначения нотации.**

DFD (Data Flow Diagram - Диаграмма Потоков Данных) методология а так же графическая нотация, предназначенная для моделирования информационный систем с точки зрения хранения, обработки и передачи **данных**.

**Процесс (англ. Process)**, т.е. функция или последовательность действий, которые нужно предпринять, чтобы данные были обработаны.

**Внешние сущности** (англ. External Entity). Это любые объекты, которые не входят в саму систему, но являются для нее источником информации либо получателями какой-либо информации из системы после обработки данных.

**Хранилище данных** (англ. Data store). Внутреннее хранилище данных для процессов в системе. Поступившие данные перед обработкой и результат после обработки, а также промежуточные значения должны где-то храниться.

**Поток данных** (англ. Data flow). В нотации отображается в виде стрелок, которые показывают, какая информация входит, а какая исходит из того или иного блока на диаграмме.

Правила построения DFD диаграмм

* **Каждый процесс должен иметь хотя бы один вход и один выход.** Смысл процессов здесь заключается в обработке данных, а потому процесс должен получить данные (входящая стрелка) и отдать куда-то после обработки (исходящая стрелка);
* **Процесс обработки данных должен иметь внешнюю входящую стрелку** (данные от внешней сущности). Для того, чтобы любой подобный процесс начал работать, мало использовать данные из хранилища, должна поступить новая информация для последующей обработки;
* **Стрелки не могут связывать напрямую хранилища данных, все связи идут через процессы.** Нет смысла просто перемещать данные из одного места в другое, а именно так читается прямая связь двух хранилищ стрелкой. Данные поступают для того, чтобы производились какие-то действия, в нашем примере – осуществлялся процесс продажи. А это возможно только посредством обработки (процесса);
* **Все процессы должны быть связаны либо с другими процессами, либо с другими хранилищами данных.** Процессы не существуют сами по себе, а потому результат должен куда-то передаваться;
* **Декомпозиция. В DFD-диаграммах предусмотрена возможность создавать крупные процессы и декомпозировать их на подпроцессы с подробным описанием действий.** Например, мы можем создать процесс «создание заявки», который потом декомпозировать на последовательность действий, например, на получение заявки, отдельно – проверку и получение данных клиента, если товар в интернет-магазине продается под заказ, то также при формировании заявки потребуется получить данные от поставщика о наличии нужных наименований и т.д. И тогда на верхней диаграмме у нас будет блок «обработка заявки», а при декомпозировании мы получим диаграмму с подробной последовательностью действий на этом этапе. При этом ни на одном этапе у нас не будет условий и ветвления. Будет процесс и его декомпозиция глубиной до 3-4 уровней.

**6. Опишите основные этапы проектирования модели данных в соответствие с методологией IDEF1x.**

1.1. Basic Entity Relationship Diagram (Basic ERD) - представляет собой модель данных верхнего уровня. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Такая диаграмма не слишком детализирована, в нее включаются основные сущности и связи между ними, которые удовлетворяют основным требованиям, предъявляемым к информационной системе (ИС). Basic ERD может включать связи многие-ко-многим (не реализуемые в СУБД) и не включать описание ключей.

1.2. Key Based Model (KBM) - более подробное представление данных. Она включает описание всех сущностей и первичных ключей и предназначена для представления структуры данных и ключей, которые соответствуют предметной области.

1.3. Fully Attributed model (FAM) – наиболее детальное представление структуры данных: включает все сущности, атрибуты и связи.

**7. Опишите назначение универсального языка моделирования UML.**

UML означает Унифицированный язык моделирования. Это язык для моделирования программных решений, структур приложений, поведения систем и бизнес-процессов.

Структурные диаграммы

* Диаграмма классов
* Диаграмма компонентов
* Диаграмма развертывания
* Диаграмма объектов
* Диаграмма пакета
* Диаграмма профиля
* Диаграмма композитной структуры

Поведенческие диаграммы

* Диаграмма вариантов использования (use case)
* Диаграмма деятельности (Activity)
* Диаграмма машины состояний
* Диаграмма последовательности (Sequence)
* Диаграмма связи
* Диаграмма обзора взаимодействия
* Временная диаграмма

**8. Опишите тип и назначение диаграммы** **активностей (activity) UML.**

Диаграмма активностей. Это поведенческая диаграмма, которая иллюстрирует поток деятельности через систему. Также могут быть использованы для отображения потока событий в бизнес-процессе. С помощью диаграммы активностей мы описываем последовательность действий для каждого прецедента, необходимая для достижения поставленной цели.

Диаграмма представляет собой блок-схему, которая наглядно показывает, как поток управления переходит от одной деятельности к другой.

Активности на диаграмме “разбросаны” по беговым дорожкам, каждая из которых соответствует поведению одного из объектов (например, клиента, менеджера, веб-сервера, сервера БД и т.п.). Благодаря этому легко определить, каким из объектов выполняется каждая из активностей. Дорожка - часть области диаграммы деятельности, на которой отображаются только те активности, за которые отвечает конкретный объект. Предназначены дорожки для разбиения диаграммы в соответствии с распределением ответственности за действия. Имя дорожки может означать роль или объект, которому она соответствует.

**9. Опишите тип и назначение диаграммы последовательности (sequence) UML.**

Диаграмма последовательности предназначена для моделирования взаимодействия объектов системы во времени, а также обмена сообщениями между ними.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники), вертикальные линии (англ. lifeline), отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами. На данной диаграмме объекты располагаются слева направо.

**10. Опишите тип и назначение диаграммы классов UML.**

Диаграмма классов предназначена для представления внутренней структуры программы в виде классов и связей между ними.

Графически класс изображается в виде прямоугольника, разделенного на 3 блока горизонтальными линиями:

* имя класса
* атрибуты (свойства) класса
* операции (методы) класса.

Для атрибутов и операций может быть указан один из трех типов видимости:

* — private (частный)
* # — protected (защищенный)
* + — public (общий)

Существует четыре типа связей в UML:

* Зависимость
* Ассоциация
* Обобщение
* Реализация

**11. Опишите тип и назначение диаграммы вариантов использования (use case) UML.**

Диаграммы Use case дают графический обзор участников системы, различных функций, необходимых этим участникам, и того, как эти различные функции взаимодействуют. С помощь диаграммы прецедентов (вариантов использования) выявляются основные пользователи системы и задачи, которые данная система должна решать.

**12. Опишите архитектурный паттерн MVC, его назначение, применимость и основные**

**компоненты.**

Model, View, Controller (MVC) — шаблон (паттерн) программирования, разделяющий архитектуру приложения на три модуля: модель (Model), представление (View), контроллер (Controller). Этот паттерн разработки нужен для того, чтобы разделить логические части приложения и создавать их отдельно друг от друга. То есть писать независимые блоки кода, которые можно как угодно менять, не затрагивая другие.

**Модель** (Model). Это основная логика приложения. Отвечает за данные, методы работы с ними и структуру программы.

**Представление** (View). Задача компонента — визуализация информации, которую он получает от модели. Отвечает за взаимодействие с пользователем.

**Контроллер** (Controller). Он обеспечивает взаимодействие с системой: обрабатывает действия пользователя, проверяет полученную информацию и передает ее модели. Контроллер определяет, как приложение будет реагировать на действия пользователя. Также контроллер может отвечать за фильтрацию данных и авторизацию.

**13. Дайте определение понятию ORM, укажите ORM, с которыми вы работали и применимость объектно-реляционного отображения в разработке.**

Объектно-реляционное отображение (ORM = object-relational mapper) - это библиотека языка программирования, выполняющая отображение объектов реляционной модели (отношения, строки и атрибуты) на объекты языка программирования (классы, экземпляры, методы, атрибуты).

Таблица (отношение, включая виртуальные таблицы - JOIN, VIEW) обычно соответствуют классу, строки таблицы - экземплярам этого класса, колонки таблицы ("реляционные атрибуты") при этом отображаются на атрибуты объекта или вызовы методов чтения/записи.

ОРП - теоретически позволяет программисту работать с таблицами, полями и связями реляционной БД, как с объектами, свойствами и коллекциями (массивами), не отвлекаясь на подробности более низкого уровня, такими, например, как порядок выборки и сохранения модифицированных данных, вопросы переносимости и особенностей диалекта SQL конкретной СУБД, генерации уникальных первичных ключей, заполнения полей ссылок для моделирования связей.

**14. Дайте определение Web API, опишите назначение и применимость разработки API по сравнению с монолитными приложениями.**

API –это интерфейс обработки приложений между веб-сервером и веб-браузером.

Максимальное разделение фронтенда и бэкенда.

Повышает безопасность. API позволяет вынести в отдельное приложение функциональность, которая должна быть защищена. Так снижается вероятность некорректного использования этих функций другими программами.

API — это набор правил, по которым приложения или части программы общаются друг с другом. Его можно встретить везде — от операционных систем до веб-приложений. API позволяет разработчикам использовать готовые инструменты и не переживать за их реализацию. А ещё он делает приложения безопаснее и помогает связывать разные программы между собой.

**15.1. Опишите назначение платформы NET, ее основные компоненты и возможности при разработке.**

NET - (Dotnet, рис.1 ), платформа компании Microsoft. Главной её особенностью является , что на этой платформе можно писать различными языками программирования такие как : C Sharp, F Sharp , C++, Visual Basic и другие. Dotnet делится на .NET Core - модульная платформа, для создания кроссплатформенных приложений с открытым исходным кодом..NET Framework - платформа для операционной системы Windows , позволяющая пользователю писать программное обеспечение.

В .NET Framework есть три основных компонента: CLR, CTS и CLS. В этой статье рассмотри их подробнее.

CLR

Основная задача CLR – автоматическое обнаружение, загрузка и управление типами .NET. В данном случае типами управляет .NET, а не программист. Также среда CLR заботится о ряде низкоуровневых деталей – управление памятью, обработка потоков, выполнение разных проверок, связанных с безопасностью.

CTS

Общая система типов (Common Type System) или система CTS предоставляет полное описание всех возможных типов данных и программных конструкций, которые поддерживаются исполняющей средой, а также способов, как все эти сущности могут взаимодействовать друг с другом. Нужно помнить, что любая возможность CTS может не поддерживаться в отдельно взятом языке, совместимом с .NET Framework.

CTS представляет собой формальную спецификацию, в которой описано то, как должны быть определены типы для того, чтобы они могли обслуживаться в CLR-среде. Существует пять типов CTS. Рассмотрим их подробнее:

CLS

Третий компонент (Common Language Specification) или спецификация CLS. В ней описано лишь то подмножество общих типов и программных конструкций, каковое способны воспринимать все .NET языки.

CLS можно считать просто подмножеством всех функциональных возможностей, определенных в CTS. CLS является набором правил, которые должны придерживаться создатели компиляторов, если они хотят чтобы их продукты могли без особых проблем функционировать в мире .NET Framework.

Кроме среды CLR и спецификаций CTS и CLS, в составе платформы .NET существует библиотека базовых классов. В этой библиотеке содержатся определения примитивов (потоки, файловые I/O, системы графической визуализации, механизмы для взаимодействия с разными внешними устройствами), предоставляет поддержку ряда служб, которые нужны в большинстве реальных приложений.

**15.2. Опишите назначение платформы ASP.NET, ее основные компоненты и возможности при разработке.**

ASP.NET работает поверх протокола HTTP и использует команды и политики HTTP для настройки двусторонней связи и сотрудничества между браузером и сервером.

для написания мощных клиент-серверных интернет-приложений.

Типичное приложение ASP.NET состоит из множества элементов: файлы веб-материалов (.aspx), исходные файлы (файлы .cs), сборки (файлы .dll и .exe), файлы источников данных (файлы .mdb), ссылки, значки, пользовательские элементы управления и другие файлы и папки.

**16. Опишите фреймворк Blazor, его назначение и преимущества при разработке.**

Blazor Server: позволяет создавать серверные приложения

и на стороне клиента, и на стороне сервера при определении кода в качестве языка программирования применяется C#, вместо JavaScript. А для описания визуального интерфейса используются стандартные HTML и CSS.

Blazor — это фреймворк, с помощью которого можно создавать приложения для браузера. Он написан на .NET и запускается с WebAssembly, поэтому разработчики могут работать как со стороной клиента, так и со стороной сервера.

**17. Перечислите, из каких основных компонентов состоит IDE, назначение инструментов линтер и профилировщик.**

Интегрированная среда разработки (IDE) - это программное приложение, которое предоставляет программистам комплексные возможности для разработки программного обеспечения.

Компоненты:

* Текстовый редактор;
* Компилятор и / или интерпретатор;
* Автоматизация сборки инструментов;
* Отладчик.

К числу возможностей современных интегрированных сред относятся: текстовый редактор, система сборки, отладчик, профайлер, генератор unit- тестов, инструменты поддержки коллективной разработки, инструменты связи с системой управления версиями, обфускатор, средства создания различных видов проектов и их визуализации, плагины, инструменты моделирования на языке UML.

Линтер (от англ. слова lint) – статический анализатор кода, который указывает на “подозрительные” участки программы и тем самым помогает программисту писать более качественный код. Подозрительными участками может быть любой код, в котором есть ошибка). Обычно, линтер устанавливается на компьютер программиста, а точнее в IDE.

Профилировщик — это специализированный программный инструмент, который выполняет процесс профилирования. Профилирование — это процесс, при котором собираются характеристики работоспособности какой-то программы. Характеристики работы программы нужны для того, чтобы оценить насколько эффективно работает исследуемая программа и выявить ее «слабые» участки.

Простыми словами, профилировщик — это программа, которая следит за другими программами, во время их исполнения. Процесс профилирования связан не только с программированием, но и с другими сферами, где нужно «отслеживать» определенные показатели определенных программ. Например, профилирование активно используется в трейдерстве, когда специальные профилировщики отслеживают работу других программ, участвующих в построении графиков торговых серверов.

**18. Опишите различия между функциональным и модульным тестированием.**

Главное отличие между модульным тестированием и функциональным тестированием является то, что модульное тестирование проверяет отдельные модули или блоки системы, в то время как функциональное тестирование проверяет функции или функциональные возможности системы.

Модульное тестирование - это тип функционального тестирования.

Модульное тестирование(Unit-testing) — уровень тестирования, на котором тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция. Функциональный тест-он же интеграционный тест, тестирующий фрагмент функциональности в системе. Это проверит многие методы и может взаимодействовать с зависимостями, такими как базы данных или веб-службы.

Интеграционное тестирование(Integration testing) – уровень тестирования, на котором отдельные программные модули объединяются и тестируются в группе.

**19. Расскажите о паттерне тестирования AAA.**

Arrange (настройка) — в этом блоке кода мы настраиваем тестовое окружение тестируемого юнита;  
Act — выполнение или вызов тестируемого сценария;  
Assert — проверка того, что тестируемый вызов ведет себя определенным образом.  
Этот паттерн улучшает структуру кода и его читабельность, однако начинать писать тест нужно всегда с элемента Act.

Тесты, структурированные с использованием AAA ("Организовать, действовать, утверждать" (Arrange, Act, Assert)), имеют три четко определенные области. Первая часть теста - это этап организации. Здесь вы подготавливаете объекты и данные, необходимые для теста. Шаг Действия содержит поведение, которое проверяется. Шаг Утверждения предоставляет утверждения, которые докажут, что код работает должным образом, или выявят проблемы.

**20. Опишите назначение репозитория проекта, систем контроля версий.**

Репозитории - хранилище информации, связанной с проектом разработки программного продукта в течение всего его жизненного цикла.

Системы управления версиями (контроля версий)–программное обеспечение, предназначенное для отслеживания изменений между различными версиями файловых документов и разделения доступа к ним. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости, возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение и т.д.

**21. Опишите систему контроля версий Git, её характерные особенности.**

Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии.

Системы контроля версий, как правило, отслеживают только изменения (дельты) между версиями файлов (а не сами полные версии), что позволяет эффективно использовать дисковое пространство.

Часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, то один из них может случайно отменить изменения, сделанные другим. Системы управления версиями отслеживают такие конфликты и предлагают средства их решения. Большинство систем может автоматически объединить (слить) изменения, сделанные разными разработчиками. Однако, такое автоматическое объединение изменений, обычно, возможно только для текстовых файлов и при условии, что изменялись разные (непересекающиеся) части этого файла. Если автоматическое объединение выполнить не удалось, система может предложить решить проблему вручную.

Многие системы управления версиями предоставляют ряд других возможностей:

1. Позволяют создавать разные варианты одного документа, т. н. ветки, с общей историей изменений до точки ветвления и с разными – после неё.

2. Дают возможность узнать, кто и когда добавил или изменил конкретную строку кода в файле.

3. Ведут журнал изменений, в который пользователи могут записывать информацию о том, что и почему они изменили в данной версии.

4. Контролируют права доступа пользователей, разрешая или запрещая чтение или изменение информации, в зависимости от того, кто запрашивает это действие.

Системы контроля версий бывают локальными, централизованными или распределёнными.

Локальная система хранит файлы на одном устройстве, централизованная использует общий сервер, а распределённая — общее облачное хранилище и локальные устройства участников команды. В локальной системе удобно работать с большими проектами, но сложно взаимодействовать с удалённой командой.

В централизованной системе налажена удалённая работа, но всё привязано к одному серверу. Любой сбой или взлом может повредить файлы проекта.

В распределённой системе налажена удалённая работа. Если с файлами основного репозитория что-то случится — проект легко восстановить из копии любого участника команды.

**22. Опишите различия между понятиями staging, commit и stash в системе контроля версий Git.**

- Область подготовленных изменений (staging area) - область куда попадают изменения(файлы), которые надо включить в коммит. git staging — область, отслеживаемая git, в которой находятся файлы еще не попавшие в commit.

- Команда git commit делает для проекта снимок текущего состояния изменений, добавленных в раздел проиндексированных файлов. Такие подтвержденные снимки состояния можно рассматривать как «безопасные» версии проекта — Git не будет их менять, пока вы явным образом не попросите об этом. Перед выполнением команды git commit необходимо использовать команду git add, чтобы добавить в проект («проиндексировать») изменения, которые будут сохранены в коммите.

Коммиты — основные конструктивные элементы временной шкалы проекта Git. Их можно рассматривать как снимки состояния или контрольные точки на временной шкале проекта Git. Коммиты создаются с помощью команды git commit, которая делает снимок состояния проекта на текущий момент времени.

- Добавить изменения в staging area - git add <имя файла> или git add \* если надо включить все изменения.

Команда git stash позволяет на время «сдать в архив» (или отложить) изменения, сделанные в рабочей копии, чтобы вы могли применить их позже. Откладывание изменений полезно, если вам необходимо переключить контекст и вы пока не готовы к созданию коммита.

**23. Перечислите команды для отображения истории коммитов, создания коммита, ветки.**

Команда для отображения истории коммитов - git log

создания коммита - git commit

ветки - git branch (git checkout – для перехода по веткам)

**24. Перечислите команды для перехода к определённому коммиту и его отмены в системе контроля версия Git.**

команды для перехода к определённому коммиту - git checkout

Отмена коммита - git revert (Git создаст новый коммит с операцией, обратной последнему коммиту. В текущую историю ветки будет добавлен новый коммит)

История коммитов будет сброшена до указанного коммита - git reset --hard

**25. Опишите процесс подключения удалённого репозитория к локальному репозиторию в качестве origin в Git.**

Git remote add origin <https://github.com/>... (url) - Добавление удаленного репозитория к существующему локальному

Чтобы подключаться по **HTTPS** , вместо пароля нужно использовать персональный **токен**.

Git pull origin master – получить данные

Git push -u origin master – отправка данных

**26. Перечислите команды, которые используются для синхронизации локального и удаленного**

**репозитория в Git.**

Git pull origin master – получить данные

Git push -u origin master – отправка данных

**Практическое задание:**

1. Опишите в нотации UML логику работы лифта в многоэтажном доме. Выбор типа диаграммы производится по усмотрению студента.
2. Опишите в нотации UML логику работы электрического чайника. Выбор типа диаграммы производится по усмотрению студента.
3. Опишите в нотации UML логику включения «умной» лампы в квартире через мобильное приложение. Выбор типа диаграммы производится по усмотрению студента.
4. Опишите в нотации UML работу курьера с мобильным приложением ресторана. Выбор типа диаграммы производится по усмотрению студента.
5. Опишите в нотации UML работу терминала для оформления заказа в заведении общепита. Выбор типа диаграммы производится по усмотрению студента.
6. Опишите в виде диаграммы активностей (Activity diagram) работу приложения сервиса доставки еды. На диаграмме должна быть продемонстрирована связь между приложением, удаленным сервером и БД.
7. Опишите в виде диаграммы активностей (Activity diagram) работу приложения для приема товаров на складское хранение. На диаграмме должна быть продемонстрирована связь между приложением, удаленным сервером и БД.
8. Опишите в виде диаграммы активностей (Activity diagram) работу приложения для внесения поставок книг в библиотеку. На диаграмме должна быть продемонстрирована связь между приложением, удаленным сервером и БД.
9. Опишите в виде диаграммы прецедентов (Use case diagram) актеров и прецеденты в АИС кафе. На диаграмме должно быть минимум 3 актера, 5 прецедентов и использована связь типа <<extend>> и <<include>> по крайней мере один раз.
10. Опишите в виде диаграммы прецедентов (Use case diagram) актеров и прецеденты в АИС колледжа. На диаграмме должно быть минимум 3 актера, 5 прецедентов и использована связь типа <<extend>> и <<include>> по крайней мере один раз.
11. Опишите в виде диаграммы прецедентов (Use case diagram) актеров и прецеденты в АИС онлайн-курсов. На диаграмме должно быть минимум 3 актера, 5 прецедентов и использована связь типа <<extend>> и <<include>> по крайней мере один раз.
12. Опишите в виде диаграммы последовательности (Sequence diagram) работу приложения сервиса объявлений. На диаграмме должна быть продемонстрирована связь между приложением, удаленным сервером и БД.
13. Опишите в виде диаграммы последовательности (Sequence diagram) работу приложения для покупки билетов на фестиваль. На диаграмме должна быть продемонстрирована связь между приложением, удаленным сервером и БД.
14. Опишите в виде диаграммы последовательности (Sequence diagram) работу приложения для записи к врачу. На диаграмме должна быть продемонстрирована связь между приложением, удаленным сервером и БД.
15. Опишите в нотации IDEF0 работу библиотеки. Декомпозиция IDEF0 должна содержать 3-4 блока.
16. Опишите в нотации IDEF0 работу супермаркета. Декомпозиция IDEF0 должна содержать 3-4 блока.
17. Опишите в нотации IDEF0 работу кинотеатра. Декомпозиция IDEF0 должна содержать 3-4 блока.
18. Опишите в нотации IDEF0 работу музея. Декомпозиция IDEF0 должна содержать 2-3 блока.
19. Опишите в нотации IDEF0 работу автосервиса. Декомпозиция IDEF0 должна содержать 3-4 блока.
20. Опишите в нотации IDEF0 работу фотостудии. Количество уровней должно быть не менее 3-х.
21. Опишите в нотации DFD иерархическую структуру компании – разработчика ПО. Количество уровней должно быть не менее 3-х.
22. Опишите в нотации DFD потоки данных в системе библиотеки. Диаграмма должна содержать по крайней мере 3 хранилища данных и 3 процесса. Выбор нотации производится по усмотрению студента.
23. Опишите в нотации DFD потоки данных в системе супермаркета. Диаграмма должна содержать по крайней мере 3 хранилища данных и 3 процесса. Выбор нотации производится по усмотрению студента.
24. Опишите в нотации DFD потоки данных в системе кинотеатра. Диаграмма должна содержать по крайней мере 3 хранилища данных и 3 процесса. Выбор нотации производится по усмотрению студента.
25. Опишите в нотации DFD потоки данных в системе музея. Диаграмма должна содержать по крайней мере 3 хранилища данных и 3 процесса. Выбор нотации производится по усмотрению студента.
26. Опишите в нотации DFD потоки данных в системе автосервиса. Диаграмма должна содержать по крайней мере 3 хранилища данных и 3 процесса. Выбор нотации производится по усмотрению студента.