**АРХАНГЕЛЬСКИЙ КОЛЛЕДЖ ТЕЛЕКОММУНИКАЦИЙ (ФИЛИАЛ) СПбГУТ**

**(АКТ (ф) СПбГУТ)**

**Отчеты по лабораторным и практическим работам**

**по МДК.02.01**

Студент: Кузьмин Андрей

Преподаватель: Маломан Юлия Сергеевна

Архангельск 2024

**Лабораторная работа №1**

**Инспекция программного кода на предмет соответствия стандартам кодирования**

1. **Цель работы**
   1. Изучить процесс выполнения инспекции программного кода.
2. **Контрольные вопросы**
   1. Инспекция программного кода — это процесс систематического анализа исходного кода программы с целью выявления ошибок, несоответствий стандартам кодирования и улучшения качества кода. Инспекция может проводиться как вручную, так и с использованием автоматизированных инструментов.
   2. Code review (ревью кода) — это процесс, в ходе которого один или несколько разработчиков проверяют код, написанный другим разработчиком, с целью выявления ошибок, улучшения качества кода и обеспечения соответствия стандартам. Ревью кода может быть формальным или неформальным и часто включает обсуждение и обратную связь.
   3. Критерии оценки качества ПО могут включать:

* Функциональность (соответствие требованиям)
* Надежность (устойчивость к сбоям)
* Удобство использования (интерфейс и опыт пользователя)
* Эффективность (производительность и использование ресурсов)
* Поддерживаемость (легкость внесения изменений и исправлений)
* Портативность (возможность работы на различных платформах)
  1. Преимущества парного программирования включают:
* Улучшение качества кода за счет совместной работы
* Быстрое выявление и исправление ошибок
* Обмен знаниями между разработчиками
* Ускорение процесса разработки
* Повышение командной сплоченности и коммуникации
  1. Типы инспекций кода могут включать
* Формальные инспекции (с четко определенными этапами и документацией)
* Неформальные инспекции (более свободный процесс, без строгих правил)
* Ревью кода (как часть процесса разработки)
* Автоматизированные инспекции (с использованием инструментов статического анализа)
  1. ПО для автоматизации инспекции программного кода включает:
* Статические анализаторы кода (например, SonarQube, ESLint, Checkstyle)
* Инструменты для проверки стиля кода (например, Prettier, Pylint)
* Системы контроля версий с функциями ревью (например, GitHub, GitLab, Bitbucket)

1. **Вывод**

Изучен процесс выполнения инспекции программного кода

**Лабораторная работа №2**

**Разработка тестового сценария**

1. **Цель работы**
   1. Изучить процесс разработки тестовых сценариев.
2. **Контрольные вопросы**
   1. Тест-кейс — это документ, который описывает условия, входные данные, действия и ожидаемые результаты для проверки конкретной функции или аспекта программного обеспечения. Тест-кейсы помогают систематически проверять, соответствует ли ПО заданным требованиям и спецификациям.
   2. Минимальный набор тестов (или минимальный набор тест-кейсов) — это группа тестов, которая обеспечивает достаточное покрытие функциональности приложения для выявления основных ошибок и проблем. Этот набор тестов должен быть достаточно малым, чтобы его можно было быстро выполнить, но при этом достаточно полным, чтобы гарантировать, что критические функции работают корректно.
   3. Тестирование методом «черного» ящика — это подход, при котором тестировщик не имеет доступа к внутренней структуре или коду приложения. Тестирование сосредоточено на входных данных и ожидаемых выходных данных. Этот метод обычно проводится тестировщиками, которые фокусируются на функциональности и пользовательском опыте, а не на технических деталях реализации.
   4. Тестирование методом «белого» ящика — это подход, при котором тестировщик имеет доступ к внутренней структуре и коду приложения. Тестирование включает анализ логики, алгоритмов и структуры кода. Этот метод обычно проводится разработчиками или тестировщиками с техническим опытом, которые могут оценить, как код реализует функциональность.
   5. Чтобы разрешить использование интернета в манифесте Android, необходимо добавить следующий элемент в файл AndroidManifest.xml: <uses-permission android:name="android.permission.INTERNET"/>
3. **Вывод**

Изучен процесс разработки тестовых сценариев.

**Лабораторная работа №3**

**Построение структурных диаграмм UML**

1. **Цель работы**
   1. Изучить процесс проектирования структурных диаграмм UML.
2. **Контрольные вопросы**
   1. UML-диаграмма классов отображает структуру системы, показывая классы, их атрибуты, методы и отношения между ними. Основные элементы диаграммы классов включают классы, ассоциации, обобщения, агрегации и композиции.
   2. UML-диаграмма компонентов отображает архитектуру системы на уровне компонентов, показывая, как они взаимодействуют друг с другом. Основные элементы диаграммы компонентов включают компоненты, интерфейсы, порты и зависимости.
   3. UML-диаграмма пакетов отображает организацию системы на уровне пакетов, группируя классы и компоненты в логические единицы. Основные элементы диаграммы пакетов включают пакеты, классы, зависимости между пакетами и видимость.
   4. UML-диаграмма развертывания отображает физическую архитектуру системы, показывая, как программные компоненты разворачиваются на аппаратных узлах. Основные элементы диаграммы развертывания включают узлы, компоненты, артефакты и связи между узлами.
   5. UML-диаграмма объектов отображает конкретные экземпляры классов и их взаимосвязи в определенный момент времени. Основные элементы диаграммы объектов включают объекты, их атрибуты и ссылки на другие объекты.
   6. UML-диаграмма состава (кооперации) отображает структуру системы, показывая, как объекты взаимодействуют друг с другом для выполнения определенной задачи. Основные элементы диаграммы состава включают объекты, их связи, роли и интерфейсы.
3. **Вывод**

Изучен процесс проектирования структурных диаграмм UML.

**Лабораторная работа №4**

**Построение поведенческих диаграмм UML**

1. **Цель работы**
   1. Изучить процесс проектирования поведенческих диаграмм UML.
2. **Контрольные вопросы**
   1. UML-диаграмма вариантов использования отображает функциональные требования системы с точки зрения пользователей (актеров) и их взаимодействия с системой. Основные элементы диаграммы вариантов использования включают актеров, варианты использования (use cases), границы системы и отношения между ними (например, ассоциации, обобщения и включения).
   2. UML-диаграмма последовательности отображает взаимодействие между объектами в виде последовательности сообщений, которые они обмениваются для выполнения определенной задачи. Основные элементы диаграммы последовательности включают объекты (или экземпляры), временные линии, сообщения (вызовы методов) и активации (периоды активности объектов).
   3. UML-диаграмма состояний отображает жизненный цикл объекта, показывая различные состояния, в которых он может находиться, и переходы между этими состояниями. Основные элементы диаграммы состояний включают состояния, переходы, события, действия и начальные и конечные состояния.
   4. UML-диаграмма деятельности отображает поток управления и данные в процессе или алгоритме, показывая последовательность действий и их взаимосвязи. Основные элементы диаграммы деятельности включают действия, переходы, начальные и конечные узлы, решения (ветвления) и параллельные процессы.
3. **Вывод**

Изучен процесс проектирования поведенческих диаграмм UML.

**Лабораторная работа №5**

**Построение диаграмм потоков данных**

1. **Цель работы**
   1. Изучить процесс проектирования диаграмм потоков данных.
2. **Контрольные вопросы**
   1. Диаграмма DFD (Data Flow Diagram) отображает поток данных в системе, показывая, как данные перемещаются между процессами, хранилищами данных и внешними сущностями. Она помогает визуализировать, как информация обрабатывается и передается в рамках системы.
   2. Элемент «процесс» на DFD показывает действия или функции, которые обрабатывают входные данные и производят выходные данные. Он изображается в виде круга или овала, внутри которого указано название процесса.
   3. Элемент «внешняя сущность» на DFD представляет собой объекты или системы, которые взаимодействуют с системой, но не являются частью ее. Внешние сущности могут быть пользователями, другими системами или организациями. Они изображаются в виде прямоугольников.
   4. Элемент «хранилище данных» на DFD показывает, где данные хранятся в системе. Это может быть база данных, файл или любой другой тип хранилища. Хранилище данных изображается в виде открытого прямоугольника или параллелограмма.
   5. Элемент «поток данных» на DFD показывает движение данных между процессами, хранилищами данных и внешними сущностями. Поток данных изображается в виде стрелок, указывающих направление движения данных, с подписями, описывающими тип данных.
   6. Для построения DFD используются следующие нотации:

* Процесс: круг или овал
* Внешняя сущность: прямоугольник
* Хранилище данных: открытый прямоугольник или параллелограмм
* Поток данных: стрелка
* Подписи для потоков данных, процессов и хранилищ для пояснения их назначения.

1. **Вывод**

Изучен процесс проектирования диаграмм потоков данных.

**Практическая работа №1**

**Анализ предметной области**

1. **Цель работы**
   1. Научиться проводить сбор и анализ требований к программному обеспечению.
2. **Контрольные вопросы**
   1. Существуют различные способы сбора требований пользователей, включая интервью, анкетирование, фокус-группы, наблюдение, анализ документации, прототипирование и мозговые штурмы.
   2. Преимущества и недостатки интервью:

* Преимущества: позволяют получить глубокое понимание потребностей и ожиданий пользователей, возможность задавать уточняющие вопросы и получать развернутые ответы, устанавливают личный контакт, что может повысить доверие и открытость респондентов.
* Недостатки: времязатратность, особенно если интервью проводятся с большим количеством пользователей, возможность предвзятости со стороны интервьюера или респондента, сложность в анализе и обобщении полученных данных.
  1. Преимущества и недостатки анкетирования:
* Преимущества: позволяют собрать данные от большого числа респондентов за короткое время, стандартизированные вопросы облегчают анализ и сравнение ответов, анонимность может способствовать более честным ответам.
* Недостатки: ограниченность в глубине получаемой информации, невозможность задавать уточняющие вопросы, риск низкой степени вовлеченности респондентов, что может привести к неполным или некачественным ответам.
  1. Диаграмма вариантов использования используется для визуализации функциональных требований системы с точки зрения пользователей (актеров) и их взаимодействия с системой. Она помогает понять, какие функции необходимы, и как пользователи будут взаимодействовать с системой.
  2. Прототипирование на этапе анализа требований используется для создания предварительных версий системы, которые позволяют пользователям визуализировать и взаимодействовать с функциональностью. Это помогает выявить недочеты и уточнить требования до начала разработки, что снижает риски и улучшает конечный продукт.
  3. Wireframe (каркасный макет) — это упрощенная визуализация интерфейса пользователя, которая показывает структуру и элементы дизайна без детальной графики. Wireframe помогает определить расположение элементов, навигацию и функциональность интерфейса, служа основой для дальнейшего проектирования.

1. **Вывод**

Научились проводить сбор и анализ требований к программному обеспечению.

**Практическая работа №2**

**Разработка и оформление технического задания**

1. **Цель работы**
   1. Научиться разрабатывать и оформлять техническое задание.
2. **Контрольные вопросы**
   1. Существуют следующие этапы разработки программного обеспечения: анализ требований, проектирование, реализация (кодирование), тестирование, внедрение, поддержка и сопровождение.
   2. Техническое задание (ТЗ) — это документ, который описывает требования к разрабатываемому программному обеспечению, включая его функциональные и нефункциональные характеристики. ТЗ служит основой для разработки и согласования между заказчиком и исполнителем.
   3. Основные разделы, входящие в техническое задание, могут включать: введение, цели и задачи проекта, функциональные требования, нефункциональные требования, ограничения и допущения, этапы и сроки выполнения, критерии приемки.
   4. В каждом из разделов технического задания указывается следующее: введение — общая информация о проекте, его значимость и контекст; цели и задачи проекта — конкретные цели, которые необходимо достичь, и задачи, которые нужно решить; функциональные требования — описание функций, которые должно выполнять программное обеспечение; нефункциональные требования — требования к производительности, безопасности, удобству использования и другим аспектам; ограничения и допущения — условия, которые могут повлиять на реализацию проекта, и предположения, на которых основывается проект; этапы и сроки выполнения — план работ, включая ключевые этапы и сроки их выполнения; критерии приемки — условия, при которых заказчик примет выполненную работу.
   5. Отличие между техническим заданием, эскизным проектом и техническим проектом заключается в уровне детализации и цели. Техническое задание определяет требования и цели проекта на высоком уровне. Эскизный проект представляет собой предварительное проектирование, которое включает общую архитектуру и основные компоненты системы, но не содержит детальной информации. Технический проект — это более детализированное описание, которое включает конкретные решения, архитектуру, технологии и методы реализации, а также детализированные планы тестирования и внедрения.
3. **Вывод**

Научились разрабатывать и оформлять техническое задание.

**Практическая работа №3**

**Построение архитектуры программного средства**

1. **Цель работы**
   1. Научиться создавать приложения с различной архитектурой.
   2. Научиться выбирать оптимальную архитектуру программного обеспечения
2. **Контрольные вопросы**
   1. Архитектура программного обеспечения — это структурное описание системы, которое определяет ее компоненты, их взаимодействия и принципы организации. Она служит основой для разработки, поддержки и масштабирования программного обеспечения.
   2. Клиент-серверная архитектура — это модель, в которой клиентские приложения запрашивают услуги или ресурсы у серверов. Серверы обрабатывают запросы и возвращают результаты клиентам. Эта архитектура позволяет разделить функции и ресурсы между клиентами и серверами.
   3. Микросервисная архитектура — это подход к разработке программного обеспечения, при котором приложение состоит из множества небольших, независимых сервисов, каждый из которых выполняет определенную функцию. Эти сервисы взаимодействуют друг с другом через API и могут разрабатываться, развертываться и масштабироваться независимо.
   4. MVC (Model-View-Controller) — это архитектурный паттерн, который разделяет приложение на три основных компонента: модель (данные и бизнес-логика), представление (интерфейс пользователя) и контроллер (логика обработки пользовательских запросов). Это разделение упрощает разработку и поддержку приложения.
   5. REST API используется для создания интерфейсов, которые позволяют различным приложениям взаимодействовать друг с другом через HTTP-протокол. Он обеспечивает стандартизированный способ обмена данными и позволяет создавать распределенные системы.
   6. Форматы, обычно применяемые для обмена данными между приложениями, включают JSON (JavaScript Object Notation), XML (eXtensible Markup Language) и YAML (YAML Ain't Markup Language). Эти форматы позволяют структурировать данные и легко их передавать.
   7. Преимущества микросервисной архитектуры включают: независимость разработки и развертывания сервисов, возможность использования различных технологий для разных сервисов, легкость масштабирования и улучшение устойчивости системы.
   8. Недостатки микросервисной архитектуры могут включать: сложность управления и оркестрации множества сервисов, необходимость в надежной системе мониторинга и логирования, а также потенциальные проблемы с производительностью из-за сетевых взаимодействий.
   9. Особенности архитектуры «peer-to-peer» (P2P) заключаются в том, что все узлы в сети равноправны и могут как предоставлять, так и запрашивать ресурсы. В этой архитектуре отсутствует централизованный сервер, что повышает устойчивость и распределенность системы, но может усложнять управление и безопасность.
3. **Вывод**

Научились создавать приложения с различной архитектурой.

Научились выбирать оптимальную архитектуру программного обеспечения

**Практическая работа №4**

**Изучение работы в системе контроля версий**

1. **Цель работы**
   1. Изучить основы работы с Git
   2. Получить навыки работы с Git
2. **Контрольные вопросы**
   1. Чтобы избежать «лишних» файлов при инициализации репозитория, следует использовать файл .gitignore, в котором перечисляются файлы и директории, которые не должны отслеживаться системой контроля версий. Это позволяет исключить временные файлы, конфигурации среды разработки и другие ненужные элементы.
   2. Ясные наименования коммитам требуются для того, чтобы другие разработчики (и вы сами в будущем) могли быстро понять, какие изменения были внесены в код. Хорошие сообщения коммитов облегчают отслеживание истории изменений и помогают в процессе ревью кода.
   3. HEAD указатель в Git — это ссылка на текущую ветку или последний коммит в текущей ветке. Он указывает на то, где вы находитесь в истории проекта и позволяет Git знать, какие изменения вы собираетесь внести.
   4. Отличие между git reset --soft и git reset --hard заключается в том, что git reset --soft перемещает указатель HEAD на указанный коммит, но сохраняет изменения в индексе и рабочем каталоге, тогда как git reset --hard также перемещает HEAD, но при этом удаляет все изменения в индексе и рабочем каталоге, возвращая их к состоянию указанного коммита.
   5. Файл .gitignore нужен для того, чтобы указать Git, какие файлы и директории не следует отслеживать. Это помогает избежать добавления временных, конфиденциальных или ненужных файлов в репозиторий.
   6. Файл readme.md нужен для предоставления информации о проекте, его назначении, установке, использовании и других важных аспектах. Он служит первым источником информации для новых пользователей и разработчиков, которые хотят понять, как работать с проектом.
3. **Вывод**

Изучили основы работы с Git

Получили навыки работы с Git