**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СЕВЕРО-КАВКАЗСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ»**

**СРЕДНЕПРОФЕССИОНАЛЬНЫЙ КОЛЛЕДЖ**

**ОТЧЕТ**

**О ПРОХОЖДЕНИИ УЧЕБНОЙ ПРАКТИКИ**

**ПМ.03. РЕВЬЮИРОВАНИЕ ПРОГРАММНЫХ ПРОДУКТОВ**

Обучающийся\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Специальность 09.02.07 Информационные системы и программирование

Место прохождения практики СПК ФГБОУ ВО «СевКавГА»

Время прохождения практики:

прибыл «15» июня 2024 г.

выбыл «28» июня 2024 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики от СПК ФГБОУ ВО «СевКавГА»:

Чагарова Зухра Нуржигидовна

г. Черкесск, 2024 г.

**СОДЕРЖАНИЕ**

Введение (описать цели и задачи)

1. Системы контроля версий

2. Создание и изучение возможностей репозитория проекта

(<https://smartiqa.ru/courses/git/lesson-1> Перейти по ссылки пройти обучение в процессе обучения все действия конспектировать в файл Word оформить в отчет).

3. Экспорт настроек в командной среде разработки (Выполнить конспект в файл Word оформить в отчет)

4. Сравнительный анализ офисных пакетов (Изучить виды офисных пакетов и провести сравнительный анализ сделать выводы и оформить все в отчете).

5.Сравнительный анализ браузеров (Изучить виды браузеров и провести сравнительный анализ сделать выводы и оформить все в отчете)

6.Сравнительный анализ средств просмотра видео(ПЗ)

7. Обратное проектирование алгоритма

8. Планирование code-review

9. Проверки на стороне клиента (ПЗ)

10. Проверки на стороне сервера (ПЗ)

11. Настройки доступа к репозиторию (ПЗ)

12. Подготовка дневника и отчета по практике.

**ВВЕДЕНИЕ**

Учебная практика по ревьюированию программных продуктов является важным элементом обучения в области информационных технологий. Она направлена на развитие навыков анализа, оценки и улучшения качества программного обеспечения. Ревьюирование программных продуктов представляет собой процесс систематического и структурированного обзора и оценки кода, документации, архитектуры и других аспектов программ для выявления ошибок, уязвимостей, а также обеспечения их соответствия требованиям качества.

Цели:  
1. Ознакомление с основными методиками и принципами ревьюирования программного кода и других аспектов программ.  
2. Приобретение практического опыта в проведении ревью программных продуктов различной сложности.  
3. Развитие навыков обнаружения и исправления ошибок, улучшения структуры и архитектуры программного кода.  
4. Повышение профессионализма в области оценки качества программного обеспечения и технического письма.  
Задачи:  
1. Изучение методов и инструментов ревьюирования программного кода и других элементов программных продуктов.  
2. Проведение ревью различных программных продуктов с учетом специфических требований качества.  
3. Оценка и анализ качества программного кода, архитектуры и процесса разработки.  
4. Подготовка отчетов о результатах ревью и предложение рекомендаций по улучшению программного продукта.  
5. Участие в дискуссиях и обсуждениях с коллегами по поводу обнаруженных проблем и способов их решения.   
  
Таким образом, учебная практика по ревьюированию программных продуктов предоставляет возможность не только практически освоить методики оценки качества программ, но и развить коммуникативные навыки и способности к критическому мышлению, что является важным для успешной работы в сфере разработки программного обеспечения.

**ПРАКТИЧЕСКАЯ РАБОТА 1**

**СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ**

**Система управления версиями (от англ. Version Control System, VCS)** — программное обеспечение для облегчения работы с изменяющейся информацией. 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 многое другое.

Понятно, что проект для команды - это бесценные знания, накопленные за все время его существования, и никому бы не хотелось, чтобы с ними что-то случилось. Система контроля версий защищает эти накопления от человеческого фактора, непредвиденных последствий и разных форс-мажоров.Обычно, работая в команде, каждый разработчик трудится над какой-то своей частью проекта: создает новые функции, оптимизирует уже существующие, исправляет ошибки в уже написанном коде. Сам проект обычно организован в виде дерева файлов. Контроль версий помогает сотрудникам одновременно работать над проектом, отслеживать, кто какие изменения внес, и избегать при этом различных конфликтов (например, когда два человека изменят один и тот же файл по-разному).

Существует три основных разновидности архитектур систем контроля версий: локальная, централизованная и распределенная. Рассмотрим их все.

**Локальная система контроля версий**

Одной из самых популярных локальных систем контроля версий на сегодняшний день (не считая Git) остается система **RCS**. Она работает по принципу сохранения изменений в ваших файлах. То есть она хранит не целую новую версию, а только указания к изменению первоначального файла. Например, "добавить к предыдущей версии строку **import math**". Таким образом, последовательно изменяя файл, система воссоздает любую из его версий.

### Централизованная система контроля версий

Такими системами были очень популярные в свое время **CVS**, **Subversion** и **Perforce**. Долгое время такой тип системы контроля версий считался стандартом. Такая система довольно удобна с точки зрения руководства компании. Она позволяет им следить, кто и чем занимается в текущий момент, позволяет настроить: кому какие файлы можно редактировать, а кому - нельзя. Тем не менее, централизованная система контроля версий имеет множество **недостатков**. Если сервер отключится на несколько часов, то работа целой компании будет парализована: никто не сможет сохранять версии файлов с кодом. Если будет поврежден жесткий диск сервера, на котором хранятся данные, то восстановить их не удастся (если только не были сделаны копии), и вся работа будет потеряна.

### Распределенная система контроля версий

**Распределенная система контроля версий** решает все описанные выше проблемы. К этой группе систем относится **Git**, **Mercurial**, **Bazaar** и некоторые другие. Особенность этой архитектуры в том, что клиенты не хранят у себя отдельные файлы, они хранят **полную копию** всех версий проекта. Поэтому, если что-то случится с сервером, работа не остановится, а продолжится, как ни в чем не бывало. Работники будут сохранять версии у себя на компьютере, а как только сервер восстановится, они загрузят все эти версии на него.

**ПРАКТИЧЕСКАЯ РАБОТА 2**

Git - это самая популярная на сегодня система контроля версий. Это развитый проект с открытым кодом, активно поддерживаемый и совершенствуемый. Как я уже говорил выше, Git устроен по принципу распределенной архитектуры, но поддерживает и локальную работу (в таком случае ваш компьютер будет единственным сервером, где хранятся все версии проекта). Чтобы в полной мере понять, в чем заключаются все преимущества Git, надо уметь работать в Git. Ну а пока мы не умеем, просто пробежимся по основным возможностям этой системы. Не пугайтесь новых непонятных слов, все термины будут подробно описаны позднее. Сейчас наша задача понять, что Git - это самая крутая VCS на сегодняшний день.  
Итак, Git обладает следующими **преимуществами.**

### Полная копия репозитория лежит у вас на машине.

### Отсюда вытекает **два** серьезных**преимущества:** все работает очень быстро, и вы получаете полный контроль над репозиторием:

### Чтобы создать репозиторий нужна всего одна команда -**git init**.

1. Все файлы VCS хранятся только в одной папке**.git**. Никаких **.svn** в каждой директории.
2. Вам не нужен постоянный и бесперебойный интернет. Утром - скачали данные с сервера к себе на машину, днем - поработали у себя, вечером - залили данные обратно на север. Проще, чем заливать каждый файл после изменения.
3. В локальном репозитории вы можете создавать дополнительные ветки, тестировать что-то новое и делать все, что угодно. Никто не увидит этого, ведь репозиторий только ваш.

**2. Контроль**

В Git можно делать что угодно с **коммитами**:

1. Удалить
2. Изменить
3. Поменять местами
4. Объединить несколько коммитов в один
5. Разделить один коммит на несколько
6. Перетаскивать коммиты между ветками

**3. Ветки**

**Ветки** в Git - это настолько мощный и функциональный инструмент, что все выполняется в них. От маленьких задач до релиза.

1. Создать ветку, переключиться между ветками, слить ветки, удалить ветку - рутинные операции.
2. За исключением релизных, ветки живут 1-3 дня. Создали ветку, написали новую функцию, протестировали, убедились, что все работает, слили с основной и удалили.

**4. Коммиты**

Чтобы сделать **коммит** (фиксацию изменений), нужно указать, какие изменения в него необходимо добавить (именно изменения, а не файлы). Преимущество этого в следующем.

1. Перед тем, как делать коммит, можно посмотреть и настроить, какие файлы в него попадут. Например, если вы хотите зафиксировать изменения только **в одном файле** из целого проекта - Git позволит вам это сделать.
2. Вы можете занести в коммит только **часть изменения** в файле. А остальные изменения откатить, или положить в другой коммит.
3. Просмотр **истории** коммитов и различий в файлах
4. Можно посмотреть, какие изменения были внесены в файл в разных коммитах.
5. Можно посмотреть историю коммитов всей ветки, чтобы проследить, как менялись файлы.

**5. Stash**

**Stash** - это очень удобная функция Git. Она позволяет заморозить текущие изменения и переключиться на другую ветку.

Например, коллега попросил вас срочно помочь ему с его работой, а у вас множество изменений в файлах, которые еще рано класть в коммит. Вы просто прописываете **git stash**, после чего переключаетесь на ветку коллеги и помогаете ему. Вам не придется создавать бесполезный коммит, только чтобы сохранить изменения в своих файлах, но при этом вы их и не потеряете при смене ветки.

**6. Работа в команде**

Вся работа выполняется **атомарно** в соответствующих ветках. После завершения работы, она отправляется на **ревью**, и только после этого ветка может быть слита с основной. Это позволяет не допустить присутствие непроверенного кода на основной ветке.

Очень удобно проводить ревью задачи, которая выполнена в отдельной ветке. Посмотреть различия коммитов, что-то исправить, прокомментировать, отправить обратно на доработку, а потом объединить отдельные коммиты и слить в основную ветку.

**7. GitHub, BitBucket etc.**

**Гитхаб**, **битбакет** и другие бесплатные удаленные репозитории очень удобны и часто используются для **open-source** проектов. Там делают форки этих проектов, обсуждают их и развивают.