Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Медіасистем та технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження підходів до управління версіями тривимірних моделей під час проектування мультимедійних видань
(тема)

Виконав:
студент 2 курсу, групи ТЕМВм-22-1

 Зуєвський Д.Р.

Спеціальності 186 Видавництво та поліграфія

Освітня програма
Технології електроних мультимедійних видань

Керівник  проф. Кулішова Н. Є.

Допускається до захисту
Зав. кафедри МСТ _____ Дейнеко Ж.В.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Медіасистем та технологій
Рівень вищої освіти другий (магістерський)
Спеціальність 186 Видавництво та поліграфія
Тип програми Освітньо-професійна
Освітня програма Технології електронних мультимедійних видань
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри МСТ_____
(підпис)
«30» жовтня 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Зуєвському Дмитру Романовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження підходів до управління версіями тривимірних моделей під час проектування мультимедійних видань*

Затверджена наказом по університету від 27.10.2023 р. № 1248 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18.01.2024 р

3. Вихідні дані до роботи
Мови розробки: Python; середовище розробки – Blender;

4. Перелік питань, що потрібно опрацювати в роботі
Вступ; Аналітичний огляд; Розробка методики; Практична реалізація підходу до збереження версій моделей; Експериментальна частина; Економічна частина; Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п. 5 включається до завдання за рішенням випускової кафедри)
Перелік слайдів презентації: Актуальність та мета роботи; Аналітичний огляд; Проблеми при використанні загальних рішень; Розробка методики; Практична реалізація; Експерементальна частина; Економічна частина; Висновки.

6. Консультанти розділів роботи (п. 6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п. 1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	проф. Кулішова Н. Є.		17.01.24
Економічна частина	ас. Помогалова Н.В.		13.01.24

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналітичний огляд	27.10	виконано
2	Розробка методики	10.11	виконано
3	Практична реалізація підходу	25.11	виконано
4	Експериментальна частина	10.12	виконано
5	Економічна частина	25.12	виконано
6	Оформлення пояснювальної записки	10.01	виконано
7	Оформлення графічної частини	03.06	виконано
8	Попередній захист роботи	21.01	виконано
9	Захист роботи	24.01	виконано

Дата видачі завдання 30.10.2023 р.

Студент _____ Зуєвський Д.Р.
(підпис)

Керівник роботи _____ проф. Кулішова Н. Є.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи містить: 56 с., 9 табл., 13 рис., 15 джерел.

СИСТЕМИ КОНТРОЛЮ ВЕРСІЙ, ПЛАГІН, МУЛЬТИМЕДІЙНІ ПРОЕКТИ, ОПТИМІЗАЦІЯ.

Метою роботи є підвищення продуктивності процесу розробки електронних видань.

В ході виконання кваліфікаційної роботи магістра вирішено такі завдання:

– проведено аналіз інформаційних джерел, що охоплюють предметну область застосування систем контролю версій;

– визначено проблему наявних систем контролю версій при використанні у мультимедійних проектах;

– реалізувано альтернативний метод зберігання версій та оцінити його ефективність на рівні прототипу порівняно з загальними рішеннями, провести аналіз отриманих результатів;

– виконано оцінку ефективності розробленого підходу.

Крім цього, було проведено економічне обґрунтування доцільності проведення даної кваліфікаційної роботи.

Практична значущість кваліфікаційної роботи полягає в застосуванні отриманих знань для розробки та впровадження плагіну, який реалізує збереження версій об’єктів всередині файлу для програмного забезпечення Blender3d, яке використовується для створення тривимірної графіки.

ABSTRACT

The explanatory note to the qualification work contains: 56 p., 9 tabl., 13 pic., 15 sources.

VERSION CONTROL SYSTEMS, PLUGINS, MULTIMEDIA PROJECTS, OPTIMIZATION.

The purpose of the work is to increase the productivity of the process of developing electronic publications.

In the course of the master's qualification work, the following tasks were solved:

– an analysis of information sources covering the subject area of application of version control systems was carried out;

– identified the problem of existing version control systems when used in multimedia projects;

– implemented an alternative method of storing versions and evaluated its effectiveness at the prototype level compared to general solutions, analyzing the obtained results;

– an evaluation of the effectiveness of the developed method was performed.

In addition, an economic justification of the feasibility of this qualification work was conducted.

The practical significance of the qualification work lies in the application of the acquired knowledge to the development and implementation of a plug-in that implements the saving of versions of objects inside the file for the Blender3d software, which is used to create three-dimensional graphics.

ЗМІСТ

С.

ВСТУП

Розробка мультимедійних видань – це комплексний процес, який проходить значну кількість етапів проектування, на кожному з етапів створюються різні версії, використовуються різні підходи, виникає значна кількість різноманітних файлів та їх різні версії, які слід зберігати для забезпечення можливості повернутися до попередніх станів або використати їх у майбутній розробці. Це важливий елемент забезпечення якості та продуктивності в процесі проектування та розробки.

Організацію зберігання файлів та їх версій можна реалізувати різноманітними способами, від самих елементарних, таких як використання певних правил найменування файлів та вбудованих інструментів операційної системи, до більш складних методів, які включають в себе використання спеціалізованих програм та сервісів таких, як системи контролю версій [1].

Проблема використання загальних методів та систем контролю версій полягає у обмеженості їх функціоналу при роботі зі складними типами файлів, якими здебільшого і представленні мультимедійні проекти.

У роботі пропонується обійти недостатність функціоналу, реалізуючи систему контролю версій всередині програми редактора файлу за допомогою додатку, що допомагає безпосередньо всередині файлу зберігати версії окремих об’єктів, та за допомогою вже наявних інструментів програми виконувати операції з версіями об’єктів.

Метою магістерського кваліфікаційного дослідження є підвищення продуктивності процесу розробки електронних видань.

Гіпотеза полягає в тому, що запропонований метод зберігання версій всередині файлу є більш ефективним, ніж використання загальних методів, які широко використовуються в індустрії.

Актуальність теми кваліфікаційного дослідження полягає в тому, що ефективний контроль версій під час розробки зменшує час розробки, дає змогу розробити більше версій між якими зручніше обрати кращу.

Об'єктом дослідження є процес розробки електронних мультимедійних проектів.

Предмет дослідження – хронометраж робіт під час розробки електронних мультимедійних проектів.

Для досягнення поставленої мети в ході виконання кваліфікаційної роботи магістра слід вирішити такі завдання:

– провести аналіз інформаційних джерел, що охоплюють предметну область застосування систем контролю версій;

– провести аналіз особливостей застосування систем контролю версій;

– визначити проблему наявних систем контролю версій при використанні у мультимедійних проектах;

– реалізувати альтернативний метод зберігання версій та оцінити його ефективність на рівні прототипу порівняно з загальними рішеннями, провести аналіз отриманих результатів;

– виконати оцінку ефективності розробленого підходу.

Практична значущість кваліфікаційної роботи полягає в застосуванні отриманих знань для розробки та впровадження додатку, який реалізує збереження версій об’єктів всередині файлу для програмного забезпечення Blender3d, яке використовується для створення тривимірної графіки. Результати роботи за проектом були описані в публікації «Розробка внутрішньопрограмої системи контролю версій» [2].

1 АНАЛІТИЧНИЙ ОГЛЯД

Під час розробки проекту виникає значна кількість різноманітних файлів та їх різні версії, які слід зберігати для забезпечення можливості повернутися до попередніх станів або використати їх у майбутній розробці. Це важливий елемент забезпечення якості та продуктивності в процесі проектування та розробки.

Організацію зберігання файлів та їх версій можна реалізувати різноманітними способами, від самих елементарних, таких як використання певних правил найменування файлів та вбудованих інструментів операційної системи, до більш складних методів, які включають в себе використання спеціалізованих програм та сервісів таких як системи контролю версій.

Системи контролю версій (СКВ) – це інструменти та платформи, які використовуються для відстеження та керування змінами в програмному коді, текстових документах, а також, як у нашому випадку, в тривимірних графічних ресурсах. Основною метою СКВ є збереження історії роботи над проектом, забезпечення можливості колективної роботи різних учасників команди та відстеження змін для уникнення конфліктів.

Збереження історії проекту: СКВ дозволяють вам зберігати детальну історію всіх змін у проекті. Це означає, що розробники можуть переглядати і повертатися до будь-якої попередньої версії проекту, що допомагає відстежувати розвиток та виявляти помилки.

Колективна робота над проектом: у галузі тривимірної графіки часто працюють команди з декількох учасників. СКВ надають можливість одночасно працювати над проектом, не завдаючи шкоди для інших членів команди. Кожен може працювати в своїй окремій гілці (branch), і пізніше об'єднати свої зміни.

Відстеження змін і конфліктів: СКВ автоматично відстежують всі зміни, що робляться в проекті. Це допомагає виявляти конфлікти, які можуть виникнути, коли два або більше учасники роблять зміни в одному файлі.

1.1 Історія систем контролю версій

Системи контролю версій (СКВ) стали невід'ємною частиною розробки програмного забезпечення і допомагають зберігати, відслідковувати та управляти змінами в коді проекту. Вони дозволяють розробникам працювати спільно, вносити зміни у проект, відстежувати історію змін, відновлювати попередні версії та вирішувати конфлікти між різними версіями файлів.

Історія їх розвитку охоплює довгий період і відображає зростання складності та обсягу програмних проектів.

Історія виникнення систем контролю версій (СКВ) датується з моменту, коли розробники і інженери почали відчувати необхідність в керуванні версіями свого програмного коду та інших текстових ресурсів. Початково ця необхідність виникла в контексті розробки програмного забезпечення, а з часом інструменти СКВ розширили свій вплив на різні галузі та сфери діяльності. Ось кілька ключових моментів у розвитку історії систем контролю версій.

Виникнення систем контролю версій: перші спроби керування версіями програмного коду з’явилися в 1970-х роках. Один із ранніх СКВ, відомий як SCCS (Source Code Control System), був розроблений в AT&T Bell Labs. SCCS дозволяв зберігати версії файлів та відстежувати їхні зміни, але він був обмеженим та не підтримував розподілену розробку.

Початкові етапи поєднали локальні системи контролю версій, які використовувались індивідуальними розробниками. Один із ранніх прикладів – система RCS (Revision Control System), яка з'явилася в 1982 році. RCS дозволяла зберігати різні версії файлів та фіксувати зміни.

Зі зростанням команд розробників, виникла необхідність у централізованих системах контролю версій. CVS (Concurrent Versions System), представлений у 1986 році, дозволяв кільком розробникам одночасно працювати над проектом. Пізніше з'явився SVN (Subversion), який вдосконалив попередні концепції та вводив нові функції. CVS був особливо популярним та став стандартом для керування версіями коду. Ці системи були централізованими, що означало, що весь код та історія змін зберігалися на центральному сервері.

З приходом Інтернету та розширенням спільної розробки коду, централізовані системи стали обмежені. Розподілені СКВ, такі як Git і Mercurial, виникли в реакції на це. Git, створений Лінусом Торвальдсом у 2005 році, став вельми популярним через свою швидкодію та розподіленій природі, яка дозволяє кожному розробнику мати свій власний репозиторій та працювати навіть офлайн. Mercurial - ще одна розподілена система, виникла в той самий період.

Наступним етапом стали хмарні платформи, такі як GitHub та Bitbucket, вони значно спростили спільну роботу розробників. Ці платформи пропонують інтегроване середовище для зберігання та спільної роботи над кодовою базою, вони надають можливість зберігати репозиторії в хмарі, ведення проблем, забезпечують інструменти для співпраці та автоматизації процесів розробки.

Зараз відбувається постійний розвиток і вдосконалення систем контролю версій. Це включає в себе впровадження концепцій DevOps, автоматизацію CI/CD процесів, а також використання розумного аналізу коду та машинного навчання для оптимізації розробки.

Усі ці етапи інновацій свідчать про постійний пошук більш ефективних та зручних інструментів для розробників, що відображає динаміку росту та зростання складності програмних проектів на протязі десятиліть.

1.2 Застосування систем контролю версій

Системи контролю версій стали необхідним інструментом для розробки програмного забезпечення, веб-розробки, наукових досліджень та управління проектами. Вони допомагають розробникам вести історію змін, впроваджувати нові функції, виправляти помилки та спільно працювати над проектами.

З часом СКВ стали необхідним інструментом для будь-якої галузі, де важливо ведення історії змін, керування версіями та спільна робота над текстовими ресурсами. Їхні можливості постійно розширюються, і вони залишаються незамінними для розробників і інших професій.

Нижче наведено галузі та зразки програмного забезпечення, які часто використовуються в них:

– розробка програмного забезпечення: Git, Mercurial, SVN. В розробці програмного забезпечення СКВ використовуються для збереження та відстеження змін вихідного коду. Кожна зміна зберігається у вигляді коміту [3], і розробники можуть легко відмічати версії, працювати паралельно та об'єднувати зміни;

– веб-розробка: GitHub, GitLab, Bitbucket. Веб-розробка включає в себе створення веб-сайтів чи веб-додатків. Це може включати клієнтську частину (фронтенд) та серверну частину, яка обробляє дані та логіку (бекенд). СКВ платформи, такі як GitHub, GitLab та Bitbucket, дозволяють командам веб-розробників спільно працювати над проектами, відстежувати проблеми та робити пул-реквести;

– керування проектами: Jira, Trello. Керування проектами охоплює планування, виконання та контроль проектів. СКВ може використовуватися для відстеження змін у завданнях, проектах та документах у системах керування проектами.

Інтеграція між платформами управління проектами та системами контролю версій є досить поширеною. Це дозволяє забезпечити більш

ефективну та згуртовану роботу розробників і проектних менеджерів. Наприклад GitHub має інтеграцію з Jira, що дозволяє зв'язувати коміти та гілки з конкретними задачами в Jira. Це полегшує відстеження прогресу роботи та забезпечує єдиний інтерфейс для розробників та менеджерів проекту;

– документація: Markdown, LaTeX. Документація включає у себе створення письмових матеріалів для пояснення функціональності, коду, процесів або вирішення проблем. Деякі системи, як Git, дозволяють вести документацію у текстовому форматі, такому як Markdown. Кожна зміна документа може бути відстежена, а історія може бути переглянута. Редактори документів наприклад, Google Docs, Microsoft Word мають вбудований функціонал для відслідковування змін;

– наукові дослідження: Git, Mercurial. В університетах та дослідницьких групах СКВ використовується для відстежування змін та версій даних, спрощуючи виправлення помилок та відновлення попередніх станів;

– графічний дизайн та мультимедіа: Git, SVN. Графічний дизайн охоплює створення візуальних елементів, таких як логотипи, афіші, веб-дизайн. Мультимедіа включає в себе обробку графіки, відео та аудіо. Для великих файлів, таких як графічні або відео файли, використовуються СКВ для зручного управління та відстеження змін;

– конфігураційне управління системами: Ansible, Puppet, Chef. В галузі IT СКВ використовуються для відстеження конфігураційних змін у системах та автоматизації розгортання інфраструктури [4]. Система конфігураційного управління веде журнал змін в конфігураціях, що дозволяє відстежувати, хто, коли і які зміни вносив до конфігурації системи. Конфігураційне управління дозволяє автоматично відновлювати систему до бажаного стану у випадку виявлення помилок або відхилень від конфігураційних правил.

Кожна з цих галузей використовує різні інструменти та методології, але використання систем контролю версій часто є спільним елементом для відстеження змін та спільної роботи.

1.3 Типи систем контролю версій

Системи контролю версій (СКВ) розвивалися з часом. Зараз серед них можна виділити централізовані та розподілені системи.

Централізовані СКВ – це системи, в яких весь код та історія змін зберігаються на центральному сервері. Інші користувачі та розробники звертаються до центрального сервера для отримання останньої версії коду та зберігання своїх внесків [5].

Ось ключові риси централізованих СКВ.

Центральний сервер: у централізованих системах контролю версій весь репозиторій (робоча директорія проекту, що містить всі ресурси), розташований на центральному сервері.

Кожен розробник взаємодіє з цим центральним сервером для отримання нових версій коду та зберігання своїх змін (рис. 1.1).

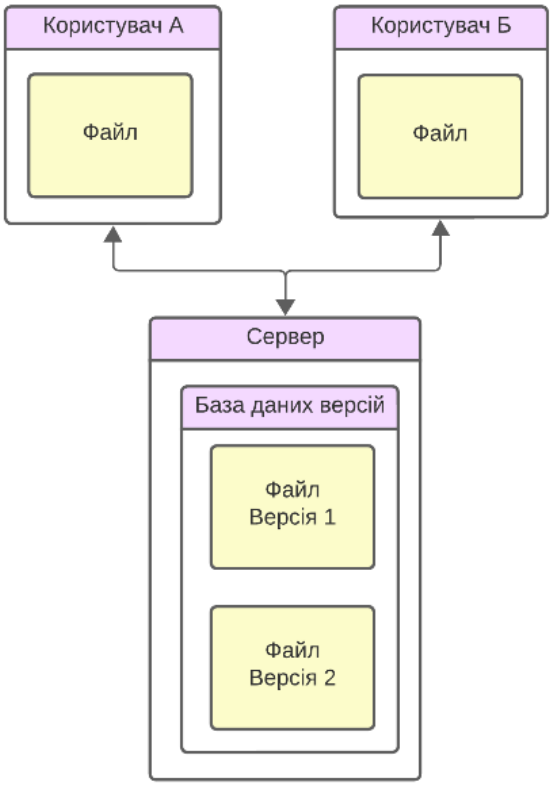

Рисунок 1.1 – Схема централізованої СКВ

Доступ до сервера: розробники мають доступ до сервера, щоб завантажувати свої зміни, отримувати оновлення та вносити зміни.

Під час одночасної роботи над одним файлом можуть виникати конфлікти злиття, які потребують ручного вирішення.

Відомими представниками централізованих рішень є CVS (Concurrent Versions System) та Subversion (SVN), які використовуються для керування версіями коду та інших ресурсів.

Серед централізованих систем окремо можна виділити локальні СКВ – це системи, в яких історія версій та файли проекту зберігаються локально на вашому комп'ютері [5].

Вони є простими та ефективними інструментами для управління версіями сутностей, особливо для індивідуальних розробників чи невеликих проектів. Часто функціонал локальних систем контролю версій представлений як частина розподільних СКВ.

Ось ключові риси локальних СКВ.

Локальні СКВ дуже прості для початку роботи.

На відміну від централізованих аналогів, вони не вимагають підключення до центрального сервера. Весь репозиторій розташований локально на комп'ютері розробника. Це робить процес роботи з контролем версій надзвичайно простим та інтуїтивним.

Локальні СКВ мають обмеження в тому, що вони не надають можливості легко спільно працювати над проектом з іншими користувачами.

Вони найбільше підходять для індивідуальної роботи або невеликих робочих команд (рис. 1.2).

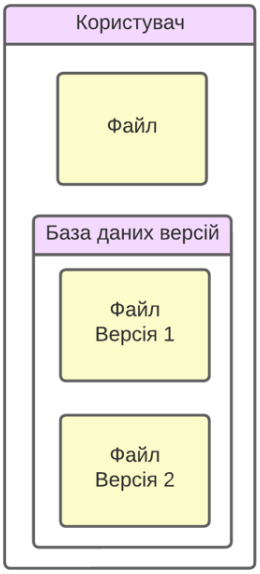

Рисунок 1.2 – Схема локальної СКВ

Розподілені СКВ вирізняються тим, що кожен розробник має свій власний репозиторій, що містить повну історію змін (рис. 1.3). Вони можуть працювати офлайн та зберігати власні версії файлів [5]. Ключові риси розподілених СКВ:

– кожен розробник має свій власний репозиторій, що містить повну копію коду та історію змін (аналогічно локальним СКВ);

– розподілені СКВ дозволяють кільком користувачам одночасно працювати над проектом та обмінюватися змінами. Кожен користувач може створювати свої гілки для роботи над окремими функціями або завданнями що полегшує спільну роботу команд;

– розподілені СКВ зазвичай працюють швидше, оскільки розробники можуть працювати офлайн і використовувати свої власні локальні репозиторії.

Відомими представниками розподілених систем Git і Mercurial, які набули популярності завдяки їхній швидкості та ефективності.

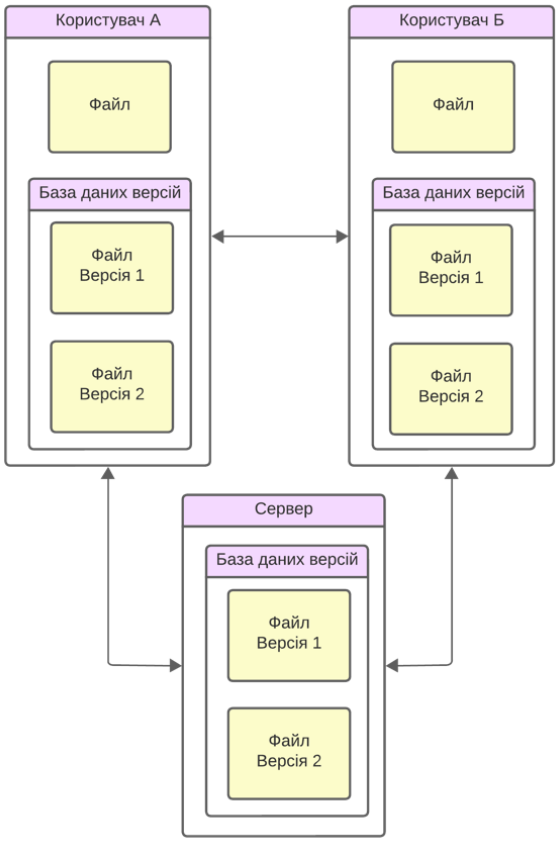

Рисунок 1.3 – Схема розподіленої СКВ

Централізовані та розподілені системи контролю версій мають свої переваги та недоліки, і вибір між ними залежить від конкретних потреб розробки та вимог проекту.

Обираючи між локальними та розподіленими СКВ, важливо враховувати розмір команди та складність проекту. Локальні СКВ підходять для індивідуальної роботи або малих проектів, тоді як розподілені СКВ рекомендуються для більших команд та складних проектів.

1.4 Використання СКВ при роботі над мультимедійними проектами

Системи контролю версій грають важливу роль в розробці мультимедійної продукції, а також в управлінні різноманітними типами файлів, що використовуються в цьому процесі. Ці системи допомагають зберігати історію роботи над проектом, спрощують колективну роботу

команди, та дозволяють відстежувати зміни у всіх типах файлів, включаючи тривимірні графічні ресурси, аудіо- та відео файли, текстові документи та інші мультимедійні компоненти.

Основні аспекти використання СКВ у мультимедійних проектах:

– збереження та відстеження змін: перша і основна ціль СКВ, мультимедійні проекти мало відрізняються у цьому від будь яких інших і можливість зберігати версії файлів та відстежувати їх змін корисно для виявлення помилок, відміни неправильних змін, а також для повернення до попередніх версій файлів, які працювали коректно. Порівняно з іншими галузями, де використовуються СКВ, з файлами, що використовуються в мультимедіа, працювати складніше, це спричинено багаточисленними типами файлів, які зберігають різні типи даних та призначені для різного програмного забезпечення;

– колективна робота: у мультимедійних проектах, де багато різних типів файлів (графіка, аудіо, відео, мета, програмні файли) і розробників, ефективна спільна робота стає ключовою. СКВ дозволяють кожному члену команди працювати над своєю частиною проекту, не переймаючись конфліктами версій;

– використання гілок: мультимедійні проекти можуть бути великими та складними, іноді з великою кількістю різних функціональностей, які розробляються паралельно. Гілки у системах контролю версій дозволяють розробникам працювати над різними версіями та аспектами проекту, не впливаючи один на одного, і пізніше об'єднати свою роботу. Це часто використовується при тестуваннях для перевірки різних варіантів реалізації продукту (A/B тестування, мультиваріантне тестування, спліт-тестування);

– відстеження залежностей: в мультимедійних проектах часто є залежності між різними ресурсами. СКВ дозволяють відстежувати ці залежності і забезпечувати правильну інтеграцію різних елементів проекту;

– робота з великими файлами: мультимедійні проекти можуть включати великі файли, такі як відео, звукові треки чи великі графічні

ресурси. Деякі СКВ підтримують ефективне керування великими файлами [6], що дозволяє ефективно працювати з ними, не втрачаючи швидкість роботи.

Однією з головних переваг використання СКВ в мультимедійній розробці є можливість зберігати структуровану історію всіх змін у проекті, незалежно від типу файлу. Це робить процес розробки більш відстежуваним і дозволяє розробникам повертатися до попередніх версій проекту, а також виявляти та виправляти помилки на ранніх стадіях.

Системи контролю версій широко використовуються у різних мультимедійних проектах, включаючи галузі відеоігор, анімації, веб-розробки, виробництва фільмів та інші [7]. Ось деякі приклади:

– розробка відеоігор: в індустрії використовують СКВ для керування версіями гри, її коду, графічних ресурсів, анімації та інших елементів. Кожен член команди розробки може вносити зміни та відстежувати їхню історію, що дозволяє спільно працювати;

– виробництво фільмів: у створенні анімаційних фільмів або спеціальних ефектів для фільму СКВ використовуються для керування версіями анімаційних моделей, сценаріїв та графічних активів. Аніматори та художники можуть вносити зміни до сцен і персонажів, а потім зберігати їхню історію та злити різні версії у фінальний продукт. Часто використовується Perforce, який має вбудовану підтримку для роботи з великими об'ємами мультимедійних даних;

– аудіо продукція: розробка аудіо контенту для ігор, фільмів чи рекламних матеріалів. Використовуються SVN (Subversion) чи Git, залежно від розміру та специфіки проекту;

– веб-розробка: розробка веб-сайтів із великою кількістю мультимедійного контенту. Використовуються Git для збереження HTML, CSS, графіки та інших файлів проекту;

– віртуальна реальність та розширена реальність: розробка VR- чи AR-додатків з великою кількістю 3D-моделей, текстур та аудіо контенту.

Використовуються Git (іноді з використанням Git LFS для керування великими 3D-моделями);

– мультимедійні архіви: створення та управління мультимедійними архівами (зображення, аудіо, відео) для бібліотек чи музейних проектів. Використовуються Git, Mercurial, або інші, в залежності від конкретних потреб проекту.

Ці приклади показують, що СКВ стали важливим інструментом у сферах, де робота з мультимедійними ресурсами є складною та вимагає спільної роботи багатьох фахівців. Вони спрощують процес розробки та дозволяють зберігати структуровану історію змін, що може бути важливим для великих та складних мультимедійних проектів. З розвитком систем контролю версій функціонал для роботи з мультимедіа розширявся, та деякі обмеження закладені безпосередньо у типах файлів, що контролюються, залишаються. Це яскраво проявляється при роботі з специфічними форматами файлів різних редакторів мультимедіа.

1.5 Проблеми при використанні стандартних рішень при роботі над мультимедійними проектами

Проблеми при роботі виникають на основі файлів, з якими необхідно працювати, розробляючи мультимедіа.

Створення мультимедіа часто зав’язано на конкретному програмному забезпеченні. В залежності від виробничого процесу набір ПЗ відрізняється, але можна виділити дві категорії, що визначають ціль програмного забезпечення у виробничому процесі:

а) Creator Software (Програмне забезпечення для створення) – це програми, які призначені для створення нових контентів, об'єктів або проектів. Вони дозволяють користувачам створювати щось з нуля;

б) Editor Software (Редакторське програмне забезпечення) – це програми, які призначені для редагування і обробки вже існуючих контентів або об'єктів.

Отже, основна відмінність між ними полягає в тому, що "creator software" допомагає створювати новий контент або об'єкти, тоді як "editor software" служить для редагування та обробки існуючих контентів або об'єктів.

Яскравим прикладом такого розподілення може послужити використання ігрового рушія – об’єкти були створенні та експортовані до проекту, рушій зберігає лише інформації про те як він використовує ці об’єкти.

Фіксування окремих змін в проекті залежить від особливостей програмного забезпечення файли якого ми відслідковуємо.

Найліпше це розподільне збереження файлів у програмах редакторах коли існує сутність та існує файл властивостей цієї сутності (мета файли). В такому випадку використання загальних систем контролю версій дає можливість відслідковувати зміну самої сутності та зміну ролі цієї сутності у певному проекті.

Інший варіант – коли програмне забезпечення зберігає всі зміни всередині одного файлу, у такому випадку відслідкувати зміну сутності та її ролі вже неможливо, що ускладнює повернення окремих змін.

Якщо використовувати зовнішню сиситему контролю версій то вона розглядає зміну всередені файлу як зміну всього файлу (рис 1.4).

Тому за допомогою неї можна виконати контроль версій всього файлу, а не окремих сктнностей всередені файлу.

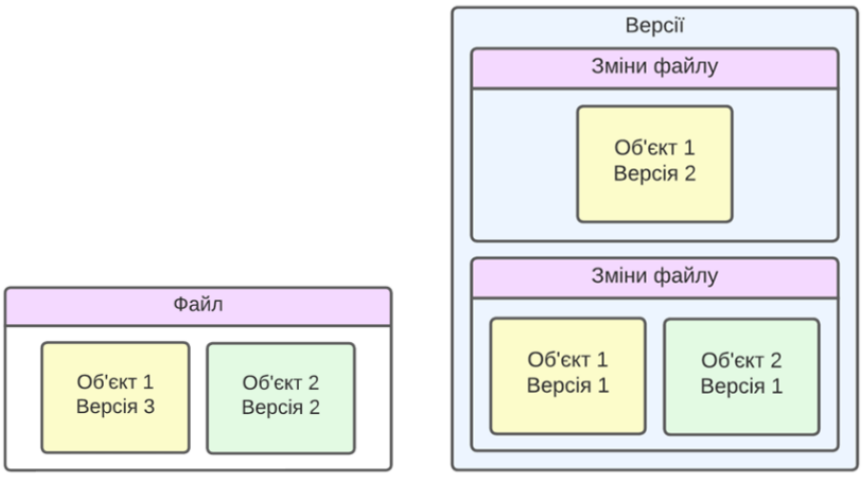

Рисунок 1.4 – Збереження змін об’єктів всередині файлів зовнішньою системою контролю версій

Виникає ряд додаткових дій коли необхідно виконати операції працюючи з версіями (рис 1.5). Це ускладнює роботу, збільшує ймовірність помилки при виконанні дій, так страждає ефективність роботи з системою контролю версій.

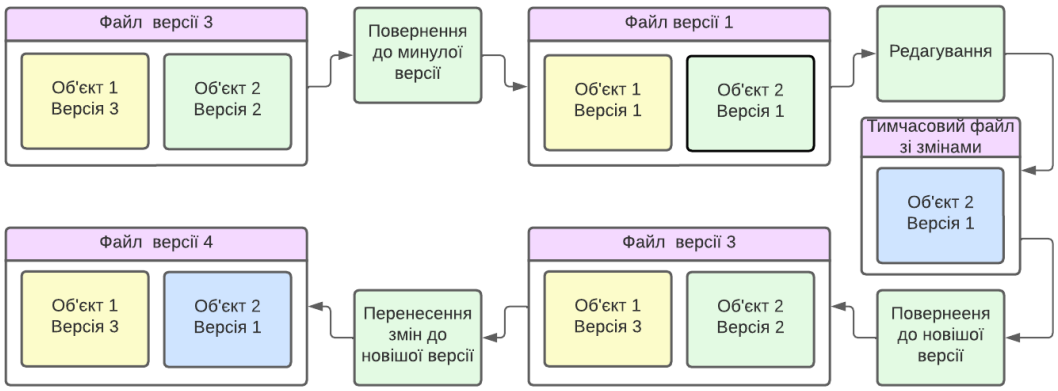

Рисунок 1.5 – Основні кроки при перенесенні змін використовуючи зовнішню СКВ

Частково проблему неможливості фіксувати окремі зміни всередині файлу за допомогою СКВ можна вирішити, ускладнюючи структуру всього проекту, для мультимедіа найчастіше це використання посилань між об’єктами в різних файлах.

Такий спосіб ускладнює роботу з структурою самого проекту, збільшується кількість файлів, у них складніше орієнтуватись, є вірогідність виникнення вкладених і змішаних конструкцій посилань.

Метою магістерського кваліфікаційного дослідження є підвищення продуктивності процесу розробки електронних видань.

Гіпотеза полягає в тому, що запропонований метод зберігання версій всередині файлу є більш ефективним, ніж використання загальних методів, які широко використовуються в індустрії.

Актуальність теми кваліфікаційного дослідження полягає в тому, що ефективний контроль версій під час розробки зменшує час розробки, дає змогу розробити більше версій між якими зручніше обрати кращу.

Об'єктом дослідження є процес розробки електронних мультимедійних проектів.

Предмет дослідження – метод хронометражу робіт під час розробки електронних мультимедійних проектів.

Для досягнення поставленої мети в ході виконання кваліфікаційної роботи магістра слід вирішити такі завдання:

– провести аналіз інформаційних джерел, що охоплюють предметну область застосування систем контролю версій;

– провести аналіз особливостей застосування систем контролю версій;

– визначити проблему наявних систем контролю версій при використанні у мультимедійних проектах;

– реалізувати альтернативний підхід до зберігання версій та оцінити його ефективність на рівні прототипу порівняно з загальними рішеннями, провести аналіз отриманих результатів;

– виконати оцінку економічної ефективності дослідження.

2 РОЗРОБКА МЕТОДИКИ

Розроблена методика має вирішити проблему неможливості відслідковувати зміни об’єктів всередині файлу.

Найпростішим способом буде реалізувати власну внутрішню організацію версій об’єктів всередині файлу. Використовуючи певну систему неймінгу можна зберігати версії одних і тих самих сутностей (рис 2.1), а використовуючи функціонал програми, виконувати дії над ними.

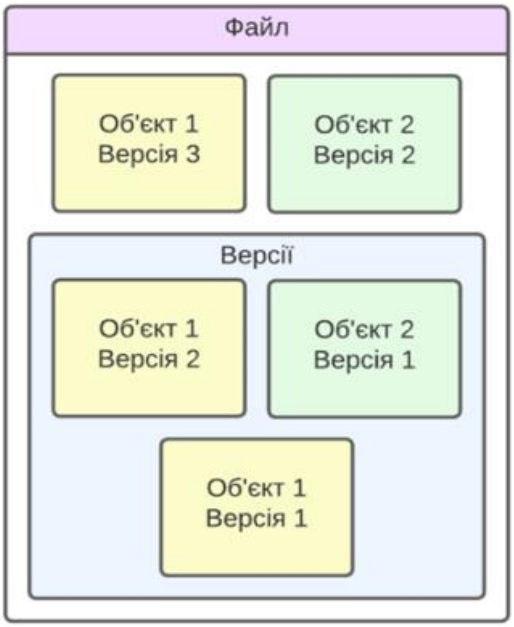

Рисунок 2.1 – Збереження змін об’єктів всередині файлів внутрішньою системою контролю версій

Ручне збереження можна полегшити, автоматизувавши окремі дії за допомогою скриптів, об’єднаних у підпрограму для вжитого ПЗ. Структурно додаток має автоматизувати дії, які можуть бути виконані користувачем, а також надати інтерфейс для роботи з версіями. Для збереження версії сутність дублюється, перейменовується та переноситься в окремий простір, де вона не буде заважати подальшій роботі. Сутності додається інформація про створений дублікат, його назву та властивості, спираючись на ці «мета»

дані буде працювати скрипт, що повертатиме об’єкт до попередньої версії [2]. Це полегшить виконнаня дій, які при використанні зовнішніх систем контролю версій потребували більшої кількості операцій (рис 2.2).

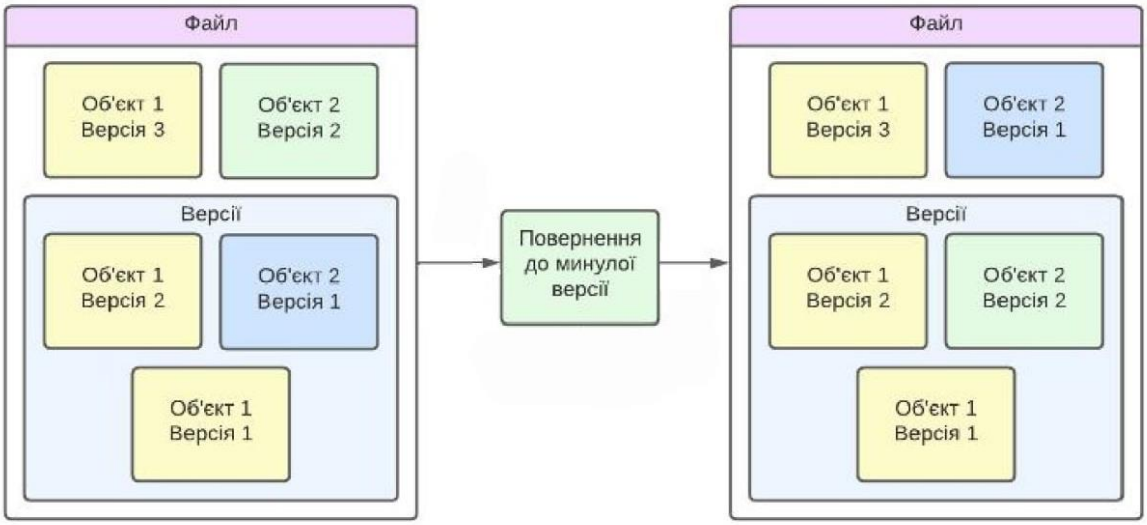

Рисунок 2.2 – Основні кроки при перенесенні змін всередині файлу

Спираючись на функціонал ПЗ можна реалізувати сценарії, що будуть переносити обрані зміни сутності або об’єднувати обрані зміни різних версій. Виключення необхідності використовувати зовнішню систему контролю версій зменшує загальну кількість дій, необхідних для початку роботи з версіями, виключаються тимчасові файли, внаслідок чого зменшується і негативний вплив людського фактору на швидкість роботи з версіями.

Одна з переваг запропонованого рішення – це незалежність створених версій від аддону, адже користувач в ручному режимі може повернутись до створених версій і виконати операції над ними, так як вони наявні у файлі і можуть бути легкодоступні для нього.

Автоматизоване збереження, реалізоване подібним чином, зменшує кількість дій та часу для створення ітерацій, спрощує об’єднання та перенесення змін.

Такий внутрішній контроль версій в поєднанні з зовнішньою системою контролю версій надає функціонал для відстеження змін сутностей всередині файлу та зміну файлу загалом.

Методика має обмеження, так як спирається на програмне забезпечення, що використовується для редагування, інструменти автоматизації або підтримка скриптів є основною вимогою для реалізації методики.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПІДХОДУ ДО ЗБЕРЕЖЕННЯ ВЕРСІЙ МОДЕЛЕЙ

3.1 Вибір програмного забезпечення для реалізації підходу

Для практичної реалізації підходу обрано пакет для 3D-моделювання Blender. Це потужний і безкоштовний пакет для 3D-моделювання, анімації, рендерингу, композитингу, редакції відео та багато іншого. Це відкрите програмне забезпечення, яке активно розробляється та має велику спільноту.

Blender став невід'ємним інструментом для створення різноманітних мультимедійних контентів. Основні можливості Blender 3D:

– 3D-моделювання: Blender дозволяє створювати складні 3D-моделі з нуля або редагувати існуючі. Від простих об'єктів до складних сцен – Blender надає розширені інструменти для творчого моделювання;

– анімація: програма дозволяє створювати анімації з використанням ключових кадрів, шляхів руху та інших методів;

– рендеринг: Blender має вбудовані двигуни рендерингу, такі як Cycles та Eevee, які дозволяють створювати реалістичні зображення та анімацію;

– композитинг: Blender включає в себе інструменти для обробки та редагування зображень та відео;

– симуляції: за допомогою Blender можна створювати різноманітні симуляції, такі як симуляція рідини, диму, вогню, тканин і т. ін.;

– спільнота і розширення: завдяки активній спільноті користувачів та розширенням (аддонам), Blender може бути розширений функціональністю в залежності від потреб користувача.

Остання можливість є ключовою для практичної реалізації запропонованого підходу. Blender має вбудовану підтримку мови програмування Python. Користувачі можуть писати скрипти для

автоматизації різних завдань. Це дозволяє створювати власні аддони, плагіни та скрипти для вирішення конкретних завдань.

Blender надає розширену Python API (Application Programming Interface), яка дозволяє отримувати доступ до практично всіх функцій та об'єктів Blender [8]. Можна маніпулювати об'єктами, створювати нові об'єкти, анімувати їх, налаштовувати параметри матеріалів і багато іншого, використовуючи Python. Є можливість створювати власні аддони і плагіни для Blender, які додають нові функції та можливості до програми. Це дозволяє розширити функціональність Blender за власними потребами і вимогами. Python дозволяє автоматизувати рутинні завдання в Blender. Наприклад, коли необхідно написати скрипт для створення повторюваних об'єктів, анімації, обробки текстур, рендерингу і багатьох інших операцій. За допомогою Python можна змінювати існуючі об'єкти, параметри матеріалів, камери, світло та багато інших параметрів.

Використовуючи Python, можна взаємодіяти з іншими програмами та сервісами. Наприклад, можна імпортувати дані з зовнішніх джерел, таких як бази даних або веб-сервіси, і використовувати їх у проекті. Подібним чином влаштовані плагіни – мости між різними програмами, що дозволять виконувати швидке перенесення даних з одного ПЗ до іншого.

Python є дуже популярною мовою для розробки аддонів та скриптів в Blender, оскільки вона має зручний та зрозумілий синтаксис і багату екосистему бібліотек та ресурсів. Багато ресурсів, документації та прикладів Python Scripting для Blender доступні на офіційному веб-сайті Blender та інших веб-ресурсах.

3.2 Інструменти розробки

Для розробки використано внутрішній редактор коду всередині Blender, який називається "Text Editor" або "Scripting Editor". Цей редактор надає зручний інтерфейс для написання, редагування та виконання Python-

скриптів безпосередньо всередині програми. Користувач може створювати та зберігати свої скрипти в цьому редакторі, а також запускати їх безпосередньо у Blender. Є і простір для тестування та відладки "Python Console" який дозволяє відлажувати будь які операції що виконуються у Blender. У редакторів є і звичні функції для редакторів коду такі як підсвічування синтаксису та автодоповнення.

3.3 Розробка

Додатки для Blender мають стандартні структури, які використовуються для загального опису додатку, імпорту бібліотек та модулів [9]. Так можна весь додаток поділити на декілька основних частин:

– словник для опису додатку;
– імпорт необхідних модулів;
– функціональна частина;
– інтерфейс додатку;
– реєстрація та скасування реєстрації додатку.

Словник для опису розширення bl_info – це спеціальний словник у форматі Python, який використовується для опису розширення (аддона) для Blender. Цей словник містить інформацію про додаток, таку як його назва, версія, автор та інші параметри. Цей словник дозволяє Blender коректно розпізнавати та взаємодіяти з вашим додатком, і він використовується при активації, встановленні та управлінні додатком через інтерфейс Blender [10].

Імпорт модулів – це процес завантаження та використання коду, який знаходиться в інших файлових модулях або бібліотеках у вашому програмному проекті. Використання імпортів дозволяє використовувати функції, класи, змінні та інші ресурси, які знаходяться в інших модулях, без необхідності повторювати код.

Основним модулем, що імпортується, є модуль «bpy», який надає доступ до API Blender [11]. Весь функціонал Blender доступний через цей

модуль, і через нього можна взаємодіяти з об'єктами, сценами, матеріалами та іншими аспектами 3D-середовища.

Часто імпортуються лише частини модуля «bpy», які дозволяють працювати лише з конкретним функціоналом Blender. Основні з них це:

а) bpy.types що містить класи, які представляють різні типи об'єктів у Blender, такі як Mesh, Armature, Camera тощо. Ці класи використовують для створення і редагування об'єктів;

б) bpy.ops, який містить набір операторів (функцій), що виконують різні дії у Blender: створення об'єктів, видалення об'єктів, переміщення камери тощо;

в) bpy.data, який дозволяє доступатися до даних про об'єкти, такі як матеріали, текстури, ключові кадри анімації тощо;

г) bpy.contex, який надає доступ до поточного контексту в Blender. Контекст представляє собою інформацію про поточну сцену, об'єкти та редагування, які виконують в момент виконання скрипта або команди в Blender. Зазвичай bpy.context використовується для отримання доступу до різних об'єктів та параметрів, таких як поточний вибраний об'єкт, поточна сцена, налаштування відображення та інші аспекти поточного стану Blender.

Функціональна частина являє собою набір створених функцій класів та операторів. Для реалізації методики, що забезпечує контроль версій, створений додаток має реалізувати ряд базових функцій, які надають можливість роботи з версіями. Всі створені функції базуються вже на існуючих можливостях Blender.

Основний ряд функцій, які необхідно реалізувати: ініціалізація об’єкта для відслідковування, створення варіанту, видалення варіанту, перехід між різними варіантами. Для зручності роботи також створено ряд додаткових функцій: визначення основного варіанту, створення нового об’єкту з обраного варіанту, видалення всіх варіантів.

Для початку необхідно підготувати простір для роботи функцій, визначити, як зберігаються інформація про версії об’єктів.

Фактично, необхідно зберігати інформацію про об’єкт та колекцію варіантів об’єкту, яка належить йому.

Створимо оператор для додавання нового варіанту. Основна мета цього оператора – створити новий варіант мешу (Mesh Variant) на основі поточного стану об'єкта. Він буде виконувати функції ініціалізації відстежування для початку роботи з версіями.

При виконанні буде виконано ряд дій над вибраним об’єктом:

– виконання перевірок назви варіанту;

– створення нового мешу на основі поточного стану активного об'єкта;

– позначення створеного мешу як захищеного від видалення (щоб він не видалявся, коли здійснюється очищення проекту);

– генерація унікального ідентифікатора;

– встановлення атрибуту для нового варіанту мешу для ідентифікації мешу як варіанту (рис 3.1);

– якщо це ініціалізація додавання запису про базову версію;

– додавання запису про колекцію для об’єкту (рис 3.2);

– додавання нового запису до колекції варіантів, який представляє собою новий варіант мешу (рис 3.2);

– додавання ідентифікатора про поточний стан об’єкта (рис 3.2).

Створимо оператор для видалення варіанту. Основна мета оператора – це видалити дані варіанта та очистити інформацію про його існування.

При виконанні буде виконано ряд дій над вибраним об’єктом:

– буде перевірено, чи є відповідний запис про варіант в колекції;

– буде перевірено, чи є варіант реальним;

– буде видалено запис про варіант;

– буде видалено відповідний варіант.

Для операції переходу між різними варіантами скористаємось колекцією варіантів вибраного об’єкту. Вибираючи варіант зі списку, будемо замінювати активну версію на обрану.

При виконанні буде виконано ряд дій над вибраним об’єктом, а також робочим простором:

– буде перевірятись, чи є у об’єкту версії;

– відображатись список всіх версій для об’єкту;

– при зміні активного елементу у списку буде змінювати версію на обрану;

– буде виведено інформацію про те, що було обрано певну версію об’єкту.

Рисунок 3.1 – Атрибут, що позначає об’єкт як версію

Рисунок 3.2 – Атрибути, що надаються для об’єкту, який має версії

Для зручності додамо операцію перейменування для варіантів. Вона має спростити менеджмент версій.

При виконанні буде виконуватись стандартний оператор перейменування. Виконання операцій спирається на унікальні ідентифікатори, тому імена можна міняти без шкоди для системи версій.

Додамо операцію для визначення базової версії. Під час ініціалізації перша версія за замовчуванням позначається як базова. Для програми вона служить ярликом, до якого прив’язується інформація про колекцію, для користувача вона позначає актуальну версію.

При виконанні буде виконано ряд дій над вибраним об’єктом:

– зміна атрибуту старої базової версії;

– зміна атрибуту навої базової версії.

Додамо операцію для створення нового об’єкту на основі вибраної версії. Функція має аналогічну мету як гілки в системах контролю версій.

При виконанні буде виконано ряд дій над вибраним об’єктом:

– створюється новий об’єкт;

– у створений об’єкт дублюється інформація про обрану версію;

На даному етапі користувач вже може використовувати створені операції через командний рядок всередині Blender, або використовуючи пошук по всім доступним операціям. Для цих операцій необхідно створити інтерфейс через який користувач матиме зручний доступ них.

Для створення інтерфейсу скористаємось вже існуючим класом «bpy.types.Panel» [12], який використовується для створення панелі в користувацькому інтерфейсі Blender. Ця панель буде відображатися в області додаткових властивостей. Всередині можна додати необхідні кнопки та призначити відповідні дії при активації (рис 3.3, рис. 3.4).

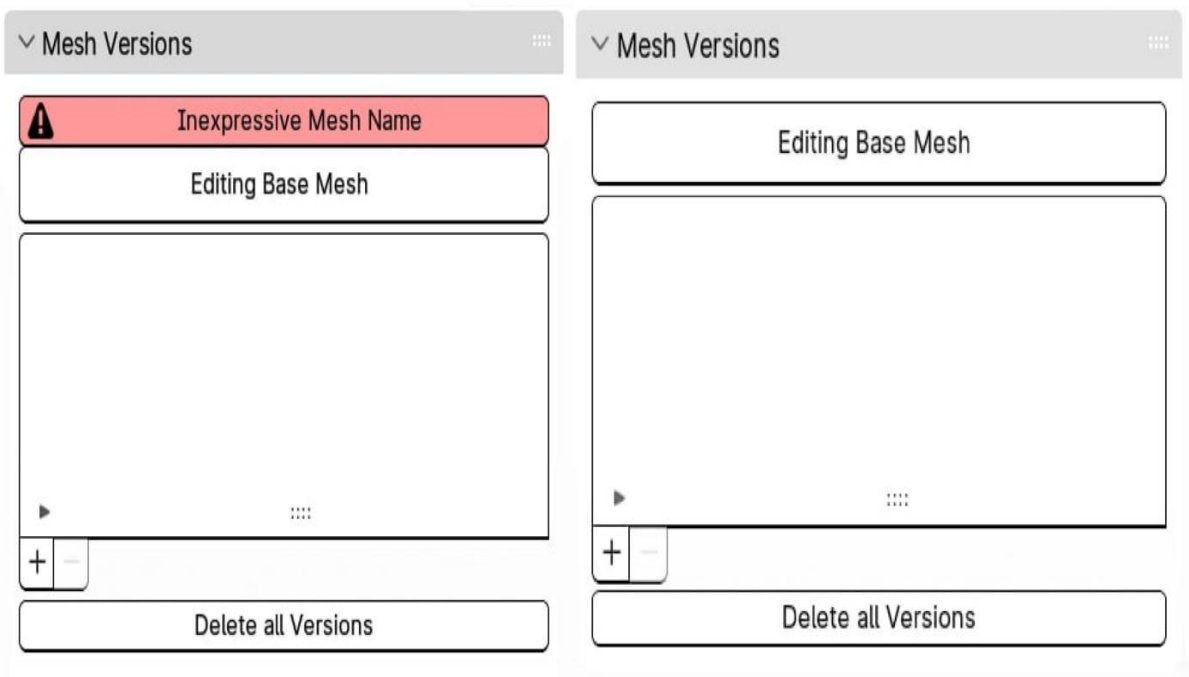

Рисунок 3.3 – Інтерфейс розробленого додатку

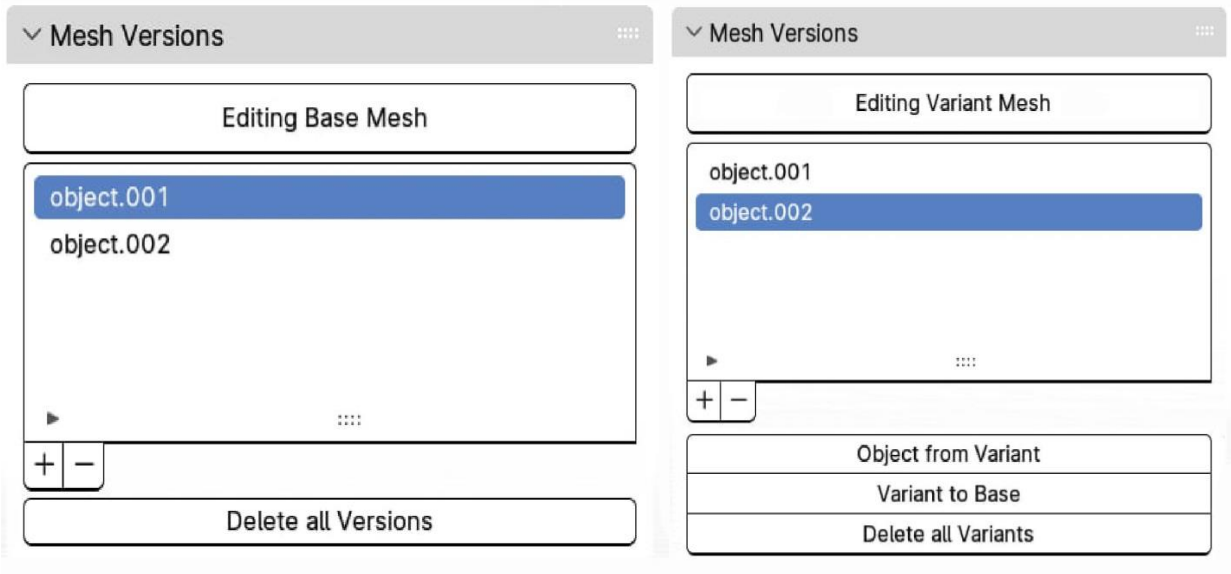

Рисунок 3.4 – Додаткові розроблені функції

3.4 Інтеграція та тестування

Написаний код додатку можна запускати в Blender, при цьому він буде створювати відповідну панель та надавати всі свої можливості. Для того, щоб додаток запускався разом з Blender, його необхідно встановити.

Можна скористатись ручним способом, перемістивши файл з кодом до відповідної теки програми, або створити архів з файлами додатку та встановити його як додаток.

Для плагіну виконано функціональне тестування. Функціональне тестування плагіну для Blender проводиться на протязі всього процесу розробки, а також перед його випуском. Основна мета тестування полягає в перевірці коректності роботи всіх компонентів плагіну. Під час тестування аналізується правильність функціонування, а також реакція плагіну на некоректні або нестандартні дії з боку користувача. Для розробленої системи версій такими діями є взаємодія з версіями у ручному режимі, їх редагування, зміна імен та властивостей не використовуючи плагін. Важливою також є реакція при імпортуванні та посиланні на версії у інших проектах. Функціональних помилок під час тестування не знайдено, при імпорті коректно переноситься вся історія версій, а при посиланні – лише актуальна версія.

4 ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

4.1. Визначення мети та завдань експериментального дослідження

Використання систем контролю версій є важливим, оскільки це дозволяє розробникам та командам ефективно відслідковувати, контролювати та управляти змінами у ресурсах проекту. Але неможливість відслідковувати зміни всередині складних типів файлів накладає обмеження та спричиняє зниженню ефективності використання СКВ. Використання додатку, який уможливлює контроль версій безпосередньо всередині програми редактора, в свою чергу, може бути більш ефективним методом.

Необхідно оцінити різні підходи до контролю версій, та порівняти ефективність їх використання з запропонованим методом під час проектування тривимірних моделей.

4.2. Основні етапи експериментального дослідження

За підсумками аналізу літературних джерел, складання критеріїв порівняльного оцінювання ефективності підходів до контролю версій. Було запропоновано наступний план проведення експерименту, за яким вимагалося здійснити таке:

– розробити прототип додатку для програмного забезпечення, що реалізує запропоновану методику збереження версій всередині файлу;

– зібрати дані про процеси, які необхідні для виконання часто використовуваних операцій зі змінними методами збереження, але при однакових умовах;

– оцінити ефективності методів збереження з запропонованим методом.

Очікується, що ефективність запропонованого підходу буде більша, що підтвердить гіпотезу дослідіження.

4.3 Опис запропонованого експерименту

Для оцінки ефективності скористаємось кількісним методом оцінки ефективності інтерфейсів. Це зумовлено тим, що ефективність роботи напряму залежить від інтерфейсів, які використовуються для роботи. Основним критерієм, що визначає якість інтерфейсу користувача, є зручність його взаємодії з елементами управління програми. Ефективним, з точки зору часу роботи користувача, прийнято вважати метод, що забезпечує виконання завдань предметної області за найменший проміжок часу.

Від того, наскільки призначений для користувача інтерфейс буде функціональним, зрозумілим і зручним кінцевому користувачеві, багато в чому залежить успішність вирішення поставленого завдання при проектуванні інтерфейсу.

Для визначення середнього часу, необхідного користувачу для виконання завдань, інженери і дослідники в галузі інтерфейсів використовують методи GOMS [13].

Ці методи дозволяють створити модель виконання завдань користувачем і на основі цих моделей оцінити якість інтерфейсу, визначивши час виконання завдань як один із основних критеріїв якості.

Найпоширенішим залишається метод KLM (Keystroke - level - Model). Він є швидким і ефективним способом оцінки часу виконання завдань досвідченими користувачами.

Підходить для порівняння безлічі альтернативних рішень в плані завдань на великій кількості користувачів [14].

Оцінка якості інтерфейсу за методом KLM полягає в розкладанні виконуваного завдання на типові складові та обчисленні часу, який в середньому витрачається користувачем на виконання цього завдання. Інтерфейс вважається кращим, коли час виконання завдання, визначений у моделі, співпадає з реальним часом виконання, і, звісно, коли час виконання завдання за допомогою інтерфейсу є меншим.

KLM – варіант методу GOMS, в якому використовуються тільки оператори, немає цілей, методів або правил вибору. Аналітик просто перераховує натиснення на клавіші, рухи мишею, які повинен зробити користувач, щоб вирішити завдання, а потім використовує декілька простих евристичних правил, щоб розставити оператори M (оператори ментальної підготовки) [15].

Таким чином, KLM надає можливість об'єктивно оцінити ефективність інтерфейсу та зрозуміти, наскільки він відповідає потребам та очікуванням користувачів. Однак важливо визначити, які саме сценарії використовувати для таких розрахунків.

Для повторюваності умов задач для різних методів контролю версій введено певні умовності. Вважається, що для експерименту підготовлена директорія, в якій знаходиться файл проекту, над яким виконуються дії. Вважається що програми систем контролю версій та програма-редактор відкриті. Будь які дії починаються у вікні редактора проекту, так як у ньому виконується основна робота у ньому і закінчуються виконання задач.

В результаті ретельного лабораторного дослідження вдалося отримати середній час виконання різних жестів [15]:

– K = 0.2 с – час, необхідний для натиснення клавіші;

– P = 1.1 з – час, необхідний для переміщення покажчика миші до певної позиції на моніторі;

– H = 0.4 – час, необхідний для переміщення руки користувача з клавіатури на мишу;

– M = 1.35 – ментальна пауза, тобто час, необхідний користувачеві на обдумування наступного кроку.

Обрано ряд операцій, які є часто використовуються при роботі з системами контролю версій:

– інтеграція системи контролю версій до проекту;

– створення версії;

– повернення до минулої версії об’єкту за умови, що він один у файлі;

– повернення до минулої версії об’єкту за умови наявності інших об’єктів у файлі;

– часткове перенесення змін з минулої версії до актуальної.

4.4Проведення експериментів

1. Інтеграція системи контролю версій.

Менеджер файлів. При використанні файлової системи операційної системи для збереження версії жодних додаткових дій перед початком роботи не потрібно.

GitHub Desktop. Для початку для інтеграції системи контролю версій GIT потрібно створити новий репозиторій. Для цього виконуються наступні дії: перехід до файлового менеджера (МРК), вибір директорії з проектом (МРК) та переміщення її у вікно GIT (МРК), налаштування репозиторію потребує назви (МРК+НМКККККН+МРК), після необхідно підтвердити створення репозиторію. Після створення репозиторію необхідно ініціалізувати його файли (створення першої версії): потрібно дати назву (МРК+НМКККККН+ МРК) для змін та підтвердити зміни. Після завершення переходимо назад до редактора (МРК).

Збереження Snowtrack. Для початку роботи з Snowtrack необхідно створити новий проект для цього перейдемо до програми (МРК), виберемо меню створення нового проекту (МРК), ввиберемо шлях дислокації проекту (МРК+НМКККККН) та назву проекту (МРК+НМКККККН+МРК), підтвердимо створення. Ініціалізація файлів автоматична і додаткових дій для початку роботи не потребується, переходимо назад до редактора (МРК).

Blender addon. Як і для GitHub Desktop, для початку роботи необхідно створити 1-у версію об'єкта (базового). Для цього необхідно вибрати об'єкт (МРК), перейти до панелі додатку (МРК), натиснути кнопку створення нової ітерації (МРК), після чого треба вибрати створену ітерацію у списку та

присвоїти статус базової (МРК) та надати відповідну назву (МРК+НМКККККН+МРК).

Результати визначення кількості та тривалості дій при інтеграції методів контролю версій наведено в табл. 4.1.

Таблиця 4.1 – Оцінка ефективності методів при інтеграції методів контролю версій

	Дії	Кліки	Переходи	Час	Час $_{розрах.}$
Manual	-	-	-	-	-
GitHub Desktop	6	8	3	28	26,7
Snowtrack	4	6	2	25	22,2
Blender addon	5	6	0	20	18,65

2. Збереження версій.

Збереження вручну. При використанні файлової системи як системи контролю версій для створення нової версії найпростішим підходом буде файл з змінами зберегти під новим ім'ям. Для збереження версії оберемо меню файл виберемо пункт зберегти як (МРК+МРК) змінимо назву (МРК+НМКККККН+МРК) та збережемо файл (МРК). У результаті кожної версії буде новий файл.

Збереження GitHub Desktop. Після внесення змін до файлу необхідно зберегти файл (МРК+МРК) перейти до GIT (МРК) у полі ввести назву для змін (МРК+НМКККККН+МРК) та натиснути кнопку (МРК) підтвердження внесення змін. Після чого повернутися до редактора (МРК).

Збереження Snowtrack. Для збереження версії файлу необхідно зберегти файл (МРК+МРК), перейти до Snowtrack (МРК), натиснути кнопку збереження (МРК) та перейменувати збережену версію (МРК+НМКККККН+МРК) (або залишити автоматично створену назву), після необхідно перейти назад до редактора (МРК).

Збереження Blender addon. Для збереження версії використовуючи додаток необхідно вибрати об'єкт (МРК) перейти до панелі додатку (МРК) та натиснути кнопку створення версії (МРК) після чого додати відповідну назву (МРК+НМКККККН+МРК).

Результати визначення кількості та тривалості дій при збереженні версій наведено в табл. 4.2.

Таблиця 4.2 – Оцінка ефективності методів при збереженні версій

	Дії	Кліки	Переходи	Час с.	Час $_{розрах.}$ с.
Manual	3	5	0	17	16
GitHub Desktop	5	7	2	23	21,30
Snowtrack	5	7	2	23	21,7
Blender addon	4	5	0	10	10,7

3. Повернення до минулої версії об’єкту за умови що у нього окремий файл.

Збереження вручну. Для повернення минулої версії необхідно перейти до менеджера файлів (МРК) у директорії проекту вибрати файл необхідної версії (МРК), відкрити його у програмі редакторі (МРК), відкрити меню файл, вибрати опцію зберегти як та перезаписати актуальну версію файлу (МРК+МРК+МРК+МРК). Після повернутися до редактору (МРК).

Збереження GitHub Desktop. Необхідно перейти до GitHub Desktop (МРК), вибрати меню історія та вибрати у ньому пункт необхідної версії (МРК), після чого на основі неї створити нову гілку (МРК+МРК+МРК+НМКККККН+МРК).

Після створення гілки необхідно перейти до редактора (МРК) та відкрити оновлений файл через меню файл, пункт відкрити (МРК). Після перейти до GIT (МРК), перейти в основну гілку (МРК+МРК+МРК). Після повернутися до редактору (МРК).

Збереження Snowtrack. Для повернення збереженої версії необхідно перейти до Snowtrack (МРК), вибрати необхідну версію в історії та натиснути

кнопку для повернення (МРК). Після повернутися до редактора та перевідкрити оновлений файл (МРК+МРК+МРК).

Збереження Blender addon. Для повернення збереженої версії за допомогою додатку необхідно вибрати об'єкт (МРК), перейти до панелі додатку (МРК) та вибрати у списку версій необхідну попередньо збережену версію (МРК).

Результати визначення кількості та тривалості дій при поверненні до минулої версії об’єкту наведено в табл. 4.3.

Таблиця 4.3 – Оцінка ефективності методів при поверненні до минулої версії об’єкту

	Дії	Кліки	Переходи	Час с.	Час $_{розрах.}$ с.
Manual	5	8	2	23	21,2
GitHub Desktop	8	13	4	37	34,95
Snowtrack	4	5	2	15	13,25
Blender addon	3	3	0	9	7,95

4. Повернення до минулої версії за умови наявності декількох об'єктів у файлів.

Збереження вручну. У редакторі необхідно видалити наявну версію об'єкта (МРК+МРК), версію якого ми хочемо замінити, після чого імпортувати з попередньо збережених версій файлів необхідну нам версію об’єкту (МРК+МРК+ МРК+МРК).

Збереження GitHub Desktop. Необхідно перейти до GitHub Desktop (МРК), вибрати меню історія (МРК) та вибрати у ньому пункт необхідної версії (МРК), після чого на основі неї створити нову гілку (МРК+МРК+МРК+НМКККККН+МРК). Після повернутися до редактору (МРК), видалити наявну версію об'єкта (МРК), та імпортувати з тимчасового файлу стару версію (МРК+МРК+МРК+МРК). Після перейти до GIT (МРК) перейти в основну гілку (МРК+МРК+МРК) та повернутися до редактору (МРК).

Збереження Snowtrack. Для повернення збереженої версії окремого об’єкту необхідно перейти до Snowtrack (МРК), вибрати необхідну версію в історії та натиснути кнопку для повернення (МРК). Після повернутися до редактору (МРК) видалити наявну версію об'єкта (МРК) та імпортувати з тимчасового файлу стару версію (МРК+МРК+МРК+МРК).

Збереження Blender addon. Для повернення збереженої версії за допомогою додатку необхідно вибрати об'єкт (МРК) перейти до панелі додатку (МРК) та вибрати у списку версій необхідну попередньо збережену версію (МРК).

Результати визначення кількості та тривалості дій при поверненні до минулої версії одного з багатьох об’єктів у файлі наведено в табл. 4.4.

Таблиця 4.4 – Оцінка ефективності методів при поверненні до минулої версії одного з багатьох об’єктів у файлі

	Дії	Кліки	Переходи	Час с.	Час $_{розрах.}$ с.
Manual	2	6	0	17	15,9
GitHub Desktop	10	18	3	56	50,85
Snowtrack	5	8	2	20	21,12
Blender addon	3	3	0	9	7,95

5. Часткове перенесення змін минулих версій до актуальної.

Збереження вручну. Необхідно імпортувати версію об'єкта (МРК+МРК+МРК+МРК) з якого будуть переноситись зміни на актуальну, виконати перенесення (МРК) та видалити імпортовану версію (МРК).

Збереження GitHub Desktop. Необхідно перейти до GIT (МРК) вибрати меню історія (МРК) та вибрати у ньому пункт необхідної версії (МРК) після чого на основі неї створити нову гілку (МРК+МРК+МРК+НМКККККН+МРК).

Після створення гілки необхідно перейти до директорії проекту (МРК), та створити тимчасовий дублікат файлу (МРК). Після перейти до GitHub

Desktop (МРК) перейти в основну гілку (МРК). Після повернутися до редактору (МРК) з тимчасового файлу імпортувати версію об’єкта з якої будуть переноситись дані (МРК+МРК+МРК+МРК). Перенести дані (МРК). Видалити імпортовану версію (МРК).

Збереження Snowtrack. Необхідно перейти до Snowtrack(МРК), вибрати необхідну версію в історії та натиснути кнопку для повернення(МРК). Після повернутися до редактору (МРК) з тимчасового файлу імпортувати версію об’єкта з якої будуть переноситись дані (МРК+МРК+МРК+МРК). Перенести дані (МРК). Видалити імпортовану версію (МРК).

Збереження Blender addon. Необхідно вибрати об'єкт (МРК) перейти на панель додатку(МРК) у списку версії вибрати версію донора (МРК) вибрати дані для перенесення(МРК) перейти на актуальну версію (МРК) та перенести дані (МРК).

Результати визначення кількості та тривалості дій при частковому перенесенні змін наведено в табл. 4.5.

Таблиця 4.5 – Оцінка ефективності методів при частковому перенесенні змін

	Дії	Кліки	Переходи	Час с.	Час $_{розрах.}$ с.
Manual	3	6	0	17	15,9
GitHub Desktop	12	18	4	55	50,85
Snowtrack	6	9	2	23	23,85
Blender addon	6	6	0	17	15,9

4.5 Результати експерименту

Для дослідження гіпотези було проведено збір даних про час необхідний для виконання популярних операцій при роботі з системами контролю версій.

Зібрану інформацію з результатами досліджень наведено в табл. 4.6 та на рис. 4.1, рис. 4.2. На основі даних можна стверджувати, що використання додатку для програмного забезпечення редактора на 66,85% ефективніше, ніж використання GitHub Desktop для аналогічних дій, на 40.12% ефективніше, ніж використання Snowtrack та на 11.38% ефективніше, ніж ручний менеджмент версій у файловому редакторі.

Таблиця 4.6. Результати порівняння ефективності методів

	Дії	Кліки	Переходи	Час с.	Час $_{розрах.}$ с.
Manual	13	25	2	74	69
GitHub Desktop	41	64	16	199	184,65
Snowtrack	24	35	10	106	102,12
Blender addon	21	23	0	65	61,15

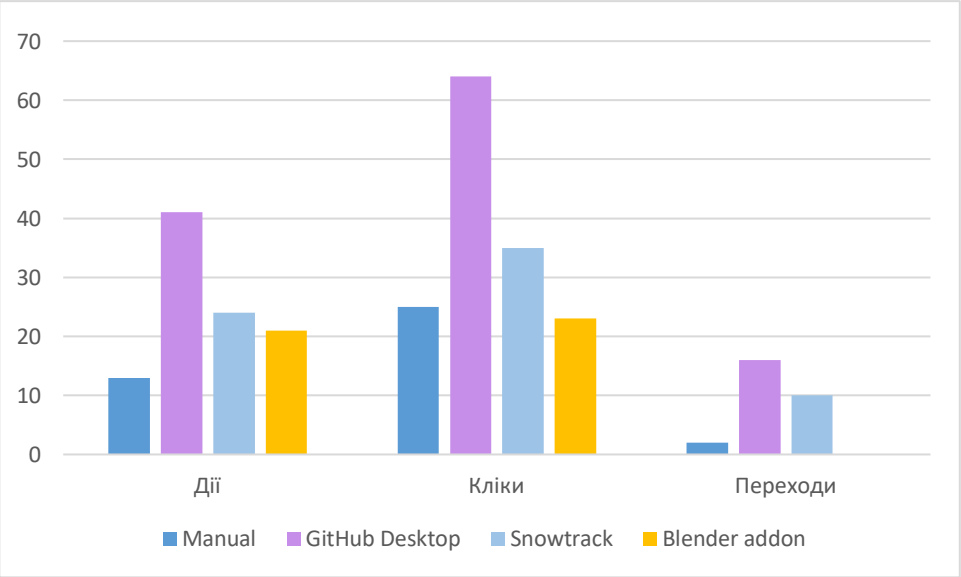

Рисунок 4.1 – Порівняння кількості необхідних дій

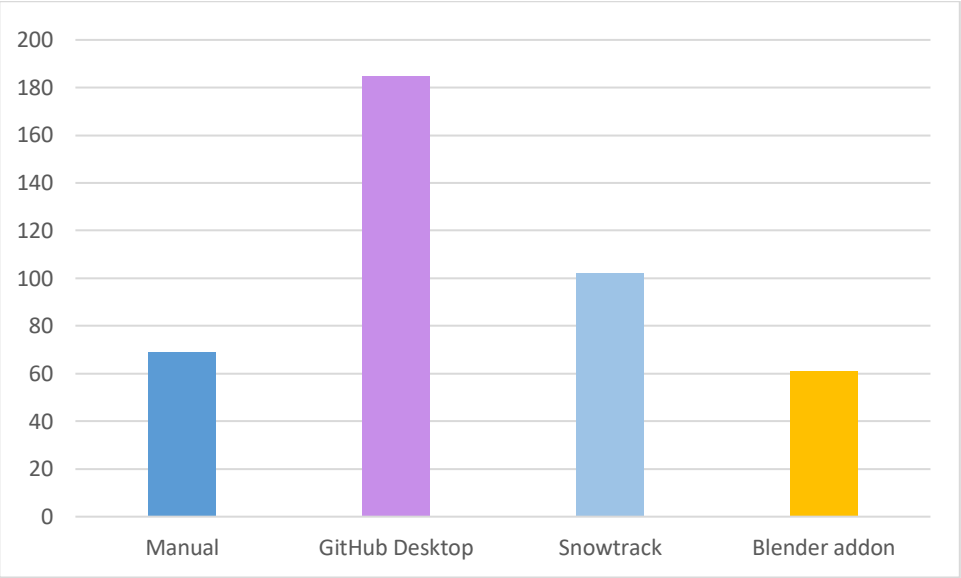

Рисунок 4.2 – Порівняння часу необхідного для виконання операцій

Як висновок, можна зазначити, що запропонована методика, реалізована, як додаток для Blender3d, ефективніша, ніж зовнішні системи контролю версій та ручний метод збереження версій, що підтверджує гіпотезу дослідження.

Також при використанні вбудованого інструменту контролю версій зменшується до нуля необхідність переходу між програмним забезпеченням, що зменшує і час, необхідний на проведення операцій, і вірогідність помилок.

Проте, слід відмітити, що ручний менеджмент версій показує себе достатньо ефективним, хоча така тенденція може не зберегтися при великій кількості версій.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Характеристика науково-дослідної роботи

В економічній частині кваліфікаційної роботи обґрунтовано економічну доцільність розробки внутрішньої системи контролю версій.

У роботі досліджено використання систем контролю версій при розробці мультимедійних проектів, розглянуто проблеми які виникають при використання загальних рішень та розроблено і реалізовано альтернативну методику збереження версій, проведено розрахунок економічної ефективності.

Реалізація розробленої методики дозволяє:

– спростити інтеграцію системи контролю версій до проекту;

– зменшити час необхідний для виконання задач, пов’язаних з контролем версій.

5.2 Етапи виконання НДР, їх трудомісткість та заробітна плата

Під час виконання науково-дослідної роботи (НДР) було проведено огляд існуючих систем контролю версій, розглянуто основні типи систем контролю версій (СКВ) та підходи, які в них використовуються, розглянуто як використовуються системи контролю версій у мультимедійних проектах та проблеми, що виникають при роботі з ними. Після було розроблено методику для реалізації контролю версій, що обходить розглянуті проблеми, та розроблено додаток до програмного забезпечення, який використовує розроблену методику. Для оцінки розробленої методики було проведено експеримент для оцінки часу необхідного для виконання задач користувачем в системі контролю версій. На основі отриманих даних зроблено висновки про ефективність розробленого рішення порівняно з існуючими СКВ.

Умовно НДР можна розділити на такі етапи: підготовчий, основний заключний.

Для підготовки проаналізовано використання систем контролю версій, визначено які складнощі виникають при використанні загальних рішень при роботі над мультимедійними проектами.

В основній частині розроблено методику і її реалізація як додаток для програмного забезпечення.

У заключній частині оцінено ефективність розробленого додатку, складено та захищено звіт з НДР.

Для виконання роботи було залучено 3 особи: технічний художник, заробітна плата – 56 700,00 грн/міс.; 3d художник, заробітна плата – 30 240,00 грн/міс.; керівник роботи, заробітна плата – 9 000,00 грн/міс.

Проведемо розрахунок трудовитрат і заробітної плати виконавців робіт. Середньоденна заробітна плата виконавця робіт ($З_{ср.дн.}$) розраховується за формулою:

$$З_{ср.дн.} = \frac{З_{ср.міс.}}{n}, \quad (5.1)$$

де Зср.міс. – середньомісячна зарплата виконавця роботи;

n – число робочих днів у місяці, (n = 22).

Етапи виконання НДР, перелік і зміст робіт, трудомісткість їх виконання, заробітна плата виконавців робіт представлені в табл. 5.1.

Таким чином, сума витрат на заробітну плату в межах виконання НДР складе 18114,51 грн.

5.3 Розрахунок одноразових витрат на розробку НДР

Калькуляція собівартості розраховується відповідно до існуючих нормативних актів України. До складу калькуляції входять такі статті витрат:

– матеріальні витрати;

– витрати на оплату праці;
– єдиний соціальний внесок;
– амортизація основних засобів (вартість машинного часу);
– витрати на спожиту електроенергію;
– інші витрати.

Таблиця 5.1 – Розрахунок трудовитрат і заробітної плати виконавців робіт

Перелік робіт	Кіль-кість вико-навців	Посада виконавця	Трудо-місткість робіт, люд.-днів	Середньо-денна заробітна плата, грн	Сума заробітної плати, грн
1. Підготовчий етап					
1.1. Розробка та затвердження ТЗ	1	Керівник роботи	1	409,09	409,09
1.2 Підготовка довідкових довідкових матеріалів та даних	1	Керівник роботи	1	409,09	409,09
2. Основний етап					
2.1 Постановка задачі	1	Керівник роботи	1	409,09	409,09
2.2 Розробка методики	2	3d художник	1	1374,54	1374,54
		Технічний художник	1	2577,27	2577,27
2.3 Розробка додатку	1	Технічний художник	3	2577,27	7731,81
2.4 Формування методики експерименту	1	Керівник роботи	1	409,09	409,09
2.5 Проведення експериментів	1	3d художник	2	1374,54	2749,08
2.6 Обробка результатів експерименту	1	Керівник роботи	1	409,09	409,09
3. Заключний етап					
3.1 Оцінка використання розробленої методики	1	Керівник роботи	1	409,09	409,09
3.2 Технічне оформлення звіту про виконання НДР	1	Керівник роботи	3	409,09	1227,27
Усього			15		18114,51

Витрати на оплату праці розраховуються, виходячи з необхідного для виконання робіт складу й кількості працівників, а також із середньомісячної заробітної плати. Відповідно до проведених розрахунків витрати на оплату праці виконавців роботи дорівнюють 18114,51 грн.

Єдиний соціальний внесок (ЄСВ) є об'єднаним внеском, який регулярно і обов'язково сплачується до системи загальнообов'язкового

державного соціального страхування. Цей внесок має на меті забезпечити соціальний захист у випадках, визначених законодавством, та гарантувати право на страхові виплати для застрахованих осіб та членів їхніх сімей у рамках різних видів державного соціального страхування.

Ставка єдиного соціального внеску складає 22 % від витрат на оплату праці, тобто розмір ЄСВ дорівнює 3985,19 грн.

Витрати на електроенергію розраховуються, виходячи зі споживаної потужності пристрою і тарифу на електроенергію. У даному випадку передбачається використання 3-ох комп'ютерів потужністю 0,7 кВт/год. Вартість однієї кВт/год електроенергії прийнято у розмірі 2,64 грн.

Витрати на використану обладнанням електроенергію (Ве) розраховуються за формулою:

$$B_e = M \cdot t \cdot T_{кВт}, \tag{5.2}$$

де М – потужність устаткування, тобто кількість енергії, споживаної за одиницю часу (кВт/година);

t – кількість годин використання устаткування за період проведення науково-дослідницької роботи;

$T_{кВт}$ – тариф, тобто вартість використання 1 кВт електроенергії.

Підставивши значення у формулу (5.2), визначимо величину витрат (Ве) на спожиту електроенергію:

$$B_e = (0{,}7 \times 72 \times 2{,}64) + (0{,}7 \times 32 \times 2{,}64) + (0{,}7 \times 24 \times 2{,}64) = 236{,}54 \text{ грн.}$$

Витрати на обслуговування ЕОМ визначаються з вартості ЕОМ і часу її експлуатації, після закінчення якого, вона підлягає заміні (звичайно цей час не перевищує 3-х років), протягом року ЕОМ використовується 254 робочих дні. Отже амортизація основних засобів розраховується за формулою:

$$AB = \sum_{k=1}^{L} \frac{BO_k}{TE_k} \times T \quad , \tag{5.3}$$

де АВ – сума амортизаційних відрахувань, нарахованих під час проведення НДР;

BO_k – вартість основних засобів k-го виду;

TE_k – термін експлуатації основних засобів k-го виду, днів;

Т – термін НДР, днів;

L – кількість видів обладнання.

Загальна вартість обладнання, що використовується під час виконання НДР, дорівнює 48 000,00 грн.

Підставивши відомі значення у (5.3), визначимо величину амортизаційних відрахувань:

$$AB = \frac{48000{,}00 \times 15}{762} = 944{,}88 \text{ грн.}$$

До інших статей витрат відносяться такі:

– адміністративні витрати: (водопостачання, водовідведення, освітлення, опалення), які прийнято у розмірі 20 % від витрат на оплату праці;

– вартість оплати послуг зв’язку.

Адміністративні витрати складатимуть 20 % від витрат на оплату праці, тобто дорівнювати 3622,90 грн.

Вартість оплати послуг зв’язку, а саме Інтернет – 150 грн за 15 днів виконання НДР.

За період виконання НДР витрати на відрядження, аутсорсинг, інформаційні послуги та маркетингові заходи не мали місця. Протягом розробки матеріальні витрати також не мали місця.

Для виконання НДР використовувалося ряд програмного забезпечення та онлайн платформ. Для розробки додатку використовувався Blender, в

експериментальній частині використовувались сторонні системи контролю версій такі як GitHub Desktop та Snowtrack, для обробки зібраних даних та оформлення супутніх текстових матеріалів GoogleDocs. Все використане програмне забезпечення та сервіси безкоштовні.

Результати розрахунку кошторису витрат, тобто одноразових витрат, на виконання НДР, наведені в табл. 5.2.

Таблиця 5.2 − Кошторис витрат на розробку НДР

№ з/п	Стаття витрат	Сума, грн.
1	Заробітна плата	18114,51
2	Єдиний соціальний внесок (22,0 % від п.1)	3985,19
3	Матеріальні витрати	–
4	Амортизація основних засобів	944,88
5	Витрати на спожиту електроенергію	236,54
6	Інші витрати	
6.1	Адміністративні витрати (20,0 % від п.1)	3622,90
6.2	Вартість послуг зв’язку	150,00
7	Усього витрати	27054,02

Таким чином, кошторис витрат на виконання даної НДР складає 27054,02 грн.

5.4 Оцінка результатів науково-дослідної роботи

Результат – це наслідок послідовності дій, виконаних під час НДР, виражений якісно або кількісно. В загальному випадку оцінка результатів НДР – це визначення ефективності отриманих рішень порівняно з сучасним науково-технічним рівнем.

Відповідно до теми даного дослідження у якості результату впровадження НДР визначено зменшення часу при використанні розробленої методики реалізованої у додатку.

Результат від впровадження НДР визначається за формулою:

$$\Delta P_j = \left| Хб_j - Хн_j \right|, \tag{5.4}$$

де ΔP_j – покращення j-ої характеристики досліджуваного процесу за рахунок впровадження результатів НДР (j = 1,m);

m – кількість досліджуваних характеристик;

$Хб_j$ – базове значення j-ої характеристики;

$Хн_j$ – нове значення j-ої характеристики після впровадження НДР.

У експериментальній частині розглянуто час, необхідний для виконання операцій у системі контролю версій якості. Розглянуто ручний спосіб менеджменту версій, дві СКВ (GitHub Desktop, Snowtrack) та розроблений додаток. Отримані результати тестування наведено у табл. 5.3.

Таблиця 5.3 – Час, необхідний для виконання задач у СКВ

Показник	Manual	GitHub Desktop	Snowtrack	Blender addon
Час, секунд	69	184,65	102,12	61,15

Підставивши відповідні значення до формули (5.4), визначимо результат від впровадження НДР у чисельному вигляді:

$$\Delta P_{Manual} = |69\text{-}61{,}15| = 7{,}85 \text{ с},$$
$$\Delta P_{\text{GitHub}} = |184{,}65\text{-}61{,}15| = 123{,}5 \text{ с},$$
$$\Delta P_{\text{Snowtrack}} = |102{,}12\text{-}61{,}15| = 40{,}97 \text{ с}.$$

На основі обчислень можна стверджувати, що використання додатку для програмного забезпечення редактора (Blender addon) на 123 секунди ефективніше, ніж використання GitHub Desktop для аналогічних дій, на 40 секунд краще за Snowtrack та на 8 секунд швидше, ніж ручний менеджмент версій у файловому редакторі.

5.5 Визначення економічної ефективності результатів НДР

Для визначення економічної ефективності результатів НДР необхідно порівняти витрати на розробку НДР з отриманими результатами.

Основним показником економічної ефективності науково-дослідної роботи є коефіцієнт «ефект-витрати», який розраховується за формулою:

$$K_{ев} = \frac{\Delta P_j}{B_p}, \tag{5.5}$$

де B_p – витрати (кошторисна вартість) на виконання НДР, грн;

$K_{ев}$ – коефіцієнт «ефект-витрати», який відбиває, наскільки кожна гривня витрат НДР змінює j-ту характеристику досліджуваного процесу.

Підставивши раніше визначені значення до (5.5), розрахуємо чисельне значення коефіцієнту «ефект-витрати» розробленого рішення порівняно з загальними рішеннями:

$$K_{ев(Manual)} = \frac{7,85}{27054,02} \times 100\ \% = 0,029\ \%,$$

$$K_{ев(GitHub)} = \frac{123,5}{27054,02} \times 100\ \% = 0,456\ \%,$$

$$K_{ев(Snowtrack)} = \frac{40,97}{27054,02} \times 100\ \% = 0,151\ \%.$$

У результаті проведених досліджень, можна зробити висновок про те, що дана НДР має позитивний показник економічної ефективності. Використовуючи розроблену методику реалізовану у додатку, можна стверджувати, що кожна гривня витрат на розробку забезпечує зменшення часу на виконання операцій з СКВ. Для ручного менеджменту це значення дорівнює 0,029 %, для GitHub Desktop 0,456 %, для Snowtrack 0,151 %. Роботу можна вважати ефективною та такою, що має науковий і технічний рівень.

ВИСНОВКИ

Під час роботи було проаналізовано методи контролю версій та визначено труднощі, які виникають при використанні цих рішень під час роботи над мультимедійними проектами. Було запропоновано підхід, що дозволяє обійти недостатність функціоналу СКВ, реалізуючи систему контролю версій всередині програми редактора файлу. За допомогою додатку безпосередньо всередині файлу можливо зберігати версії окремих об’єктів та за допомогою вже наявних інструментів програми виконувати операції з версіями об’єктів.

В процесі виконання роботи у рамках сформульованих завдань було виконане наступне:

– проведено аналіз інформаційних джерел, що охоплюють предметну область застосування систем контролю версій;

– проведено аналіз особливостей застосування систем контролю версій;

– визначено проблему наявних систем контролю версій при використанні у мультимедійних проектах;

– реалізовано альтернативний підхід зберігання версій та проведено оцінку його ефективності; на рівні прототипу порівняно з загальними рішеннями;

– проведено аналіз отриманих результатів;

– виконано оцінку економічної ефективності дослідження.

Таким чином, можна вважати, що результат дослідження відповідає вимоги поставленого завдання, мета дослідження досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Про систему контролю версій. URL: https://git-scm.com/book/uk/v2/Вступ-Про-систему-контролю-версій/ (дата звернення: 12.09.2023).

2. Зуєвський Д.Р. Розробка внутрішньопрограмої системи контролю версій // 27-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6, Ч. II. – Харків: ХНУРЕ. 2023. – 182-183 с.

3. Git Basics Recording Changes to the Repository URL: https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository/ (дата звернення: 12.09.2023).

4. An Overview of Software Configuration Management. URL: https://www.researchgate.net/profile/Agha-Haider/ publication/ 258338056_ An_Overview_of_Software_Configuration_Management/links/00463527dd61e90b 68000000/An-Overview-of-Software-Configuration-Management.pdf/ (дата звернення: 12.09.2023).

5. Zolkifli N. N., Ngah A.,Deraman A. Version Control System: A Review // 3rd International Conference on Computer Science and Computational Intelligence 2018. URL: https://www.sciencedirect.com/science/article/pii/ S1877050918314819 /(дата звернення: 12.09.2023).

6. A Modern Approach to Versioning Large Files for Machine learning. URL: https://blog.infuseai.io/a-modern-approach-to-versioning-large-datasets-for-machine-learning-fca2f541dd85/ (дата звернення: 12.09.2023).

7. Version control isn't just for programmers. URL: https://opensource.com/ life/16/2/version-control-isnt-just-programmers/ (дата звернення: 12.09.2023).

8. Blender 4.0 Python API Documentation API Overview. URL: https://docs.blender.org/api/current/info_ overview.html/ (дата звернення: 12.09.2023).

9. Blender 4.0 Reference Manual Addon Tutorial. URL: https://docs.blender.org/manual/en/latest/advanced/scripting/addon_tutorial.html/ (дата звернення: 12.09.2023).

10. Developer Documentation Script Meta Info. URL: https://wiki.blender.org/wiki/Process/Addons/Guidelines/metainfo/ (дата звернення: 12.09.2023).

11. Blender 4.0 Python API Documentation Context Access (bpy.context). URL: https://docs.blender.org/api/current/bpy.context.html/ (дата звернення: 12.09.2023).

12. Creating panels for placing Blender add-ons user interface (UI). URL: https://b3d.interplanety.org/en/creating-panels-for-placing-blender-add-ons-user-interface-ui/ (дата звернення: 12.09.2023).

13. GOMS. URL: https://www.usabilitybok.org/goms/ (дата звернення: 19.10.2023).

14. GOMS KLM. URL: https://www.usabilitybok.org/klm-goms/ (дата звернення: 19.10.2023).

15. UI/UX мультимедійних продуктів «Методи кількісної оцінки юзабіліті». URL: https://dl.nure.ua/ pluginfile.php /655958/mod_resource/ content/2/UI UX 5 курс ТЕМВ 07 GOMS Fitts.pdf/ (дата звернення: 19.10.2023).