Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-22М-2
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Безушко О.С.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Доренський О.П.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Безушку Олександру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET*

2. Керівник роботи *Доренський Олександр Павлович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 33-13 від 04.08.2023 року

3. Строк подання студентом роботи до захисту *10.12.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Економічна ефективність розробленої програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2023	14.11.2023
Охорона праці	Оришака О.В.	06.10.2023	16.11.2023

7. Дата видачі завдання « 6 » вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
« 6 » вересня 2023 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2023 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Безушко О.С. Дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Метою розробки є дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Об'єктом дослідження є процес реінжинірингу та рефакторингу програмного коду до платформи .NET.

Предметом дослідження є методи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Методи дослідження базуються на методах інженерії програмного забезпечення, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Delphi 10.

Ключові слова: комп'ютерні науки, реінжиніринг, рефакторинг

ABSTRACT

Bezushko O.S. Research and software implementation of the system of reengineering and refactoring of software code to the .NET platform. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the system of reengineering and refactoring of software code to the .NET platform.

The goal of the development is the research and software implementation of the system of reengineering and refactoring of the software code to the .NET platform.

The object of research is the process of reengineering and refactoring of software code to the .NET platform.

The subject of research is the methods of reengineering and refactoring of software code to the .NET platform.

Research methods are based on software engineering methods, mathematical statistics methods, and software development methods.

The result of the work is the software implementation of the system of reengineering and refactoring of the software code to the .NET platform.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Delphi 10 environment.

Keywords: computer science, reengineering, refactoring

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ОС – операційна система.

ПЗ – програмне забезпечення.

ПК – персональний комп’ютер .

Лексема – послідовність символів і слів, що однозначно визначають яку-небудь складову мови.

Лексичний аналіз – процес сканування потоку вхідних символів і поділу його на рядки.

Токен – послідовність символів (не обов'язково текстових), однозначно характеризують лексему.

ЦО – цивільна оборона.

ПЗ – програмне забезпечення.

КС – комп’ютерна система.

ВСТУП

Актуальність теми. Щоб не відставати від конкуренції та зробити споживачів задоволеними вашим цифровим продуктом у 2021 році, потрібно підтримувати його програмний компонент на найвищому рівні стандартів якості.

Одним із способів досягти цього є частий перегляд і вдосконалення коду. У цьому відношенні існує два основних підходи до підтримки надійності програмного забезпечення – рефакторинг і реінжиніринг.

У роботі далі детальніше розглянемо ці дві практики, дослідимо їх відмінності та дізнаємось про основні переваги оновлення програмного забезпечення для бізнесу.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем реінжинірингу та рефакторингу програмного коду до платформи .NET.

– Дослідження системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

– Програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Об'єктом дослідження є процес реінжинірингу та рефакторингу програмного коду до платформи .NET.

Предметом дослідження є методи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Методи дослідження базуються на методах інженерії програмного забезпечення, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод реінжинірингу та рефакторингу програмного коду до платформи .NET.

– Розроблено вітчизняний продукт реінжинірингу та рефакторингу програмного коду до платформи .NET, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі реінжинірингу та рефакторингу програмного коду до платформи .NET.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти «Наука – виробництву», 2023, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №14.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

По суті, і рефакторинг, і реінжиніринг спрямовані на переписування існуючого комп'ютерного коду для покращення внутрішнього дизайну, структури та/або реалізації програмного забезпечення, зберігаючи при цьому його зовнішню функціональність.

Рефакторинг описує зміну бітів і фрагментів програмного забезпечення, зберігаючи ядро програми майже недоторканим. Водночас реінжиніринг передбачає внесення фундаментальних змін у структуру та дизайн програми – адаптацію програмного забезпечення до нової апаратної платформи, зміну мови програмування чи перехід на новий діалект.

Таким чином, існує три основних типи практик реінжинірингу програмного забезпечення. Ті:

– Портування – коли програма налаштована для роботи на іншому обладнанні.

– Переклад – коли код перекладається зі старої (застарілої) мови на нову (сучасну).

– і міграція – коли код переходить на новий діалект мови без зміни його внутрішньої природи.

По суті, дві розглянуті практики часто збігаються, оскільки будь-яка реінжиніринг складається з серії ініціатив рефакторингу.

Щоб навести приклад із реального життя для довідки, давайте порівняємо це з ремонтом старого автомобіля. Один із способів – відремонтувати лише ті деталі, які зношилися, щоб він являв собою фактично той самий автомобіль усередині та зовні, лише з деякими свіжими деталями. Це був би рефакторинг. Або ви можете розібрати автомобіль і замінити його двигун, підвіску, трансмісію

тощо, або навіть зробити його газовим. Це реінжиніринг.

Ви обираєте найкращий варіант залежно від налаштувань, ресурсів і потреб. При цьому слід зазначити, що, як і з автомобілем, реінжиніринг часто займає набагато більше часу.

Тепер, коли ми зрозуміли природу редизайну програмного забезпечення, давайте перейдемо до його ключових переваг з точки зору бізнесу.

1.2 Область застосування

Мабуть, робота із застарілим кодом є другим найменш улюбленим заняттям розробників (відразу після стрибка з мосту). Це також найпоширеніша причина розливання кави на клавіатуру в гніві.

Отже, навіщо це робити в першу чергу?

Виявляється, є три досить переконливі причини для того, щоб бізнес розглянув можливість переписати код свого продукту. Розглянемо кожен окремо:

Довгостроковий прибуток

Перше, про що варто думати, розглядаючи будь-яку бізнес-ініціативу, це її грошова оцінка. Хоча відновлення програмного забезпечення передбачає негайні витрати, довгострокова окупність інвестицій від оновлення цифрового продукту в більшості випадків покриє їх удесятеро.

Лише в США різні збої програмного забезпечення коштували компаніям колосальних 1,7 трлн доларів у 2023 році, причому ця цифра зросла на понад 60% за рік. Це, безперечно, одна зі статистичних даних, яку слід враховувати, якщо бюджет вашої компанії є одним із головних проблем.

І навіть якщо зараз немає проблем, пов’язаних із кодом, які безпосередньо впливають на ваш бізнес, проактивний підхід точно заощадить непогані гроші на вирішенні цих проблем у майбутньому. Відомо, що хворобу набагато легше попередити, ніж лікувати. Таким чином, редизайн веб-сайту чи програми – це, безумовно, перспективне рішення.

Якість продукції

ІТ-технології не стоять на місці, і застосування нових методів розробки програмного забезпечення або шліфування існуючого коду завжди позитивно позначиться на його продуктивності. Отже, якість – це ще один важливий момент, який слід враховувати при розгляді настроювання чи перебудови цифрового продукту.

Скільки років вашій платформі? Можливо, варто перенести сервери в хмару для кращої гнучкості або для збільшення масштабу. Можливо, настав час змінити частину свого стеку технологій на новіші технології. Або, цілком можливо, було б гарною ідеєю переписати всю програму іншою мовою програмування.

Як би там не було, правильне перепроектування вашого програмного забезпечення покращить його роботу, зовнішній вигляд і відчуття, що кінцевим користувачам точно сподобається та цінуватиме час.

Безпека та захист

Чим старішою стає частина програмного забезпечення, тим більша ймовірність, що вона зазнає проблем у роботі в сучасному середовищі. Від інтеграції та обслуговування до оновлення – точно не завадить бути в курсі останніх технологічних тенденцій і гарантувати, що нічого не піде не так, коли найменше очікується.

Насправді рефакторинг фрагментів вашого коду може допомогти вам уникнути необхідності переробки всього продукту з часом, що може бути набагато дорожчим і трудомістким.

Поступове модифікування програмного забезпечення також зменшує ризики втрати цінних бізнес-даних або зупинки продукту, тож ви можете безтурботно оновлювати веб-сайт або програму безпечним і безпечним способом.

Коли розглядати оновлення продукту

Тепер усе вищесказане звучить багатообіцяюче та чудово. Але як визначити, чи цифровий продукт потребує термінового оновлення чи перегляду?

У цьому відношенні також варто враховувати кілька червоних прапорців.

Функціональність

Перший і найголовніший підказка про те, що вам потрібно розглянути рефакторинг або реінжиніринг, з'являється після відповіді на таке запитання:

Чи продукт, який ви використовуєте, виконує всі функції, для яких він був розроблений, і чи користувачі повністю задоволені його функціональністю?

Якщо все працює і користувачі задоволені, то, ймовірно, немає потреби турбуватися про модернізацію програмного забезпечення, принаймні, немає гострої потреби. Ви все одно можете розглянути довгострокові переваги оновлення програмного забезпечення та попросити свою команду IT просканувати продукт на наявність можливих вузьких місць і проблем. Але пам'ятайте про філософію «не лагодьте те, що не зламано» і намагайтеся зосередитися на терміновому та важливому.

Продуктивність

Чи ваш додаток або веб-сайт працює достатньо швидко та безперебійно?

Якщо так, хороші новини! Але якщо це не так, ви не захочете витрачати час своїх користувачів на збереження застарілих продуктів і «биття старого коня». Вкладіть частину свого часу та ресурсів у реінжиніринг або реорганізацію вузьких місць програмного забезпечення, і ви побачите, що ви відчуєте набагато більше, коли користувачі помітять покращення.

Помилки

Якщо баги та помилки з'являються швидше, ніж ваша команда в змозі їх виправити, це, безумовно, знак, щоб переглянути структуру коду вашого цифрового продукту та змінити її.

Насправді ніщо так не псує користувацький досвід і сприйняття бізнесу, як помилки продукту, тому краще не ігнорувати проблему, якщо вона є, навіть незначна.

Зміна платформи

І останнє, але не менш важливе: ви можете розглянути можливість відновлення програмного забезпечення під час переходу на нове апаратне чи програмне середовище.

Як уже згадувалося, можливо, настав час перейти до хмари або взагалі перевести свій продукт на нову мову кодування. Поговоріть зі своєю командою IT та обговоріть необхідність змінити основи та потенційні переваги.

Загалом, зміна дизайну вашого додатка чи веб-сайту – це важлива тема, на яку потрібно часто звертатися, і в ній є великий потенціал.

Незалежно від того, виконується рефакторинг частини коду чи реінжинірингу всього продукту, варто все продумати, порівняти плюси та мінуси та оцінити потенційні проблеми. Якщо це зробити належним чином і вчасно, оновлення вашого програмного забезпечення безумовно зміниться на краще як для вас, так і для споживачів ваших послуг.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Рефакторинг – це такий процес зміни коду програми, при якому поліпшується її внутрішня структура, а зовнішнє поводження не змінюється. Інакше кажучи, це спосіб приведення коду в порядок, при якому шанси появи нових помилок у коді мінімальні.

Зміни при рефакторингу коду вносяться покроково, дрібними операціями. Перейменувати функцію, розбити її на декілька підфункцій, змінити сигнатуру методу – всі ці операції легко виконати так, щоб не занести в код помилок. Однак сукупний ефект ряду таких елементарних операцій може бути досить значним – від спрощення структури коду, до повної зміни архітектури розроблювальної програми.

Особливу популярність рефакторинг придбав в останні десять років з розвитком екстремального програмування й інших agile-методик програмування, у яких рефакторинг є частиною циклу розробки програмного забезпечення. Тим часом, проведення рефакторингу вручну – досить стомлююче заняття. Наприклад, щоб змінити порядок проходження параметрів у методі класу, мало змінити сигнатуру методу – необхідно пройти по всьому коді програми й модифікувати всі виклики даного методу. Зміна тривіальне, але вимагає активного "копіпастингу".

Попит народжує пропозиція, і в останні роки виробники засобів розробки стали впроваджувати у свої програмні продукти засобу, що автоматизують і спрощують проведення рефакторингу. Середовище Visual Studio уперше

обзавелася такими засобами у версії VS2005. Розроблювачі, що використовують C# і J#, одержали у своє розпорядження наступний набір інструментів:

– Rename – перейменування ім'я змінної, методу, класу й т.п. з автоматичним відновленням всіх посилань на це ім'я в коді.

– Extract method – оформлення виділеної частини коду в новий, окремий метод.

– Encapsulate field – створення властивості, що приховує обрану змінну-член класу.

– Extract interface – створення інтерфейсу на основі списку методів класу.

– Promote local variable to parameter – винесення локальної змінної в параметр методу.

– Remove/Reorder parameters – видалення параметрів методу й зміна порядку їхнього проходження з автоматичним відновленням всіх посилань у коді на даний метод.

Реалізований набір інструментів, звичайно ж, спростив життя розроблювачам, однак він далеко не ідеальний. З недоліків можна відзначити:

– відсутність підтримки C++ і VB (в VS2008 її всі так само немає);

– досить повільна робота – виклик будь-якої операції рефакторингу приводить до появи діалогу відновлення IntelliSense і розроблювач змушений чекати, причому часом досить довго, поки цей діалог завершить роботу;

– набір засобів рефакторингу мінімальний, ряд корисних інструментів у ньому відсутній;

– інтерфейс далеко не самий ергономічний і вимагає активного використання мишки.

На щастя, в Microsoft Visual Studio здавна реалізована прекрасна підтримка плагінів, за допомогою яких можна розширювати її можливості. Зокрема, існує безліч плагінів, як платних, так і безкоштовних, що доповнюють і поліпшують стандартний набір інструментів рефакторингу в студії. Спробуємо розібратися, хто з них що вміє.

Visual Assist

Visual Assist X – плагін, розроблений фірмою Whole Tomato Inc. Плагін підтримує три мови програмування – C++, C# і VB.NET і працює практично у всіх версіях студії, починаючи с Visual C++ 6.0 (не підтримуються тільки Express-Версії, для якої реалізація плагінів заборонена ліцензійною угодою).

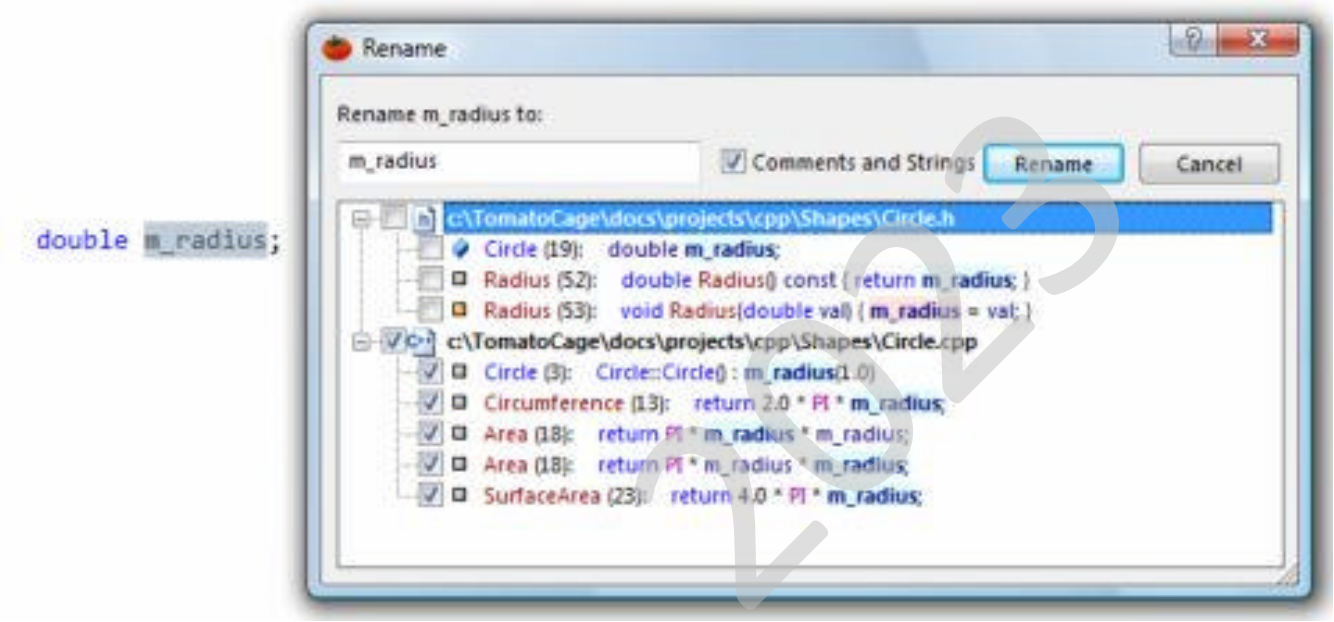

Рисунок 2.1 – Діалог функції Rename в Visual Assist X

Набір засобів рефакторингу, пропонованих Visual Assist X, є загальним для всіх мов і включає аналоги стандартних інструментів "Rename", "Extract Method", "Encapsulate Field", нові інструменти "Change Signature", "AddMethod", "Add Similar Member" і "Document Method". Для розроблювачів C++ так само доступні функції "Create Declaration", "Create Implementation" і "Move Implementation to Source File".

Інструменти рефакторингу Visual Assist X працюють істотно швидше стандартних. Їхній інтерфейс всі так само заснований на використанні модальних діалогів, однак він багато в чому більше ергономічний. Наприклад, при виклику функції "Rename" показується один діалог, а не два – передперегляд сполучений з

уведенням нового імені. Функція "Change Signature" дозволяє змінювати сигнатуру методу довільним образом, що набагато зручніше, ніж тикати мишкою за списком параметрів у стандартних діалогах "Remove Parameter" і "Reorder Parameters".

Visual Assist реалізує зручний спосіб швидкого виклику інструментів рефакторингу – досить навести мишкою на виділений текст або будь-який ідентифікатор, як поруч із курсором миші з'являється іконка, при натисканні на яку відкривається меню зі списком доступних для обраної ділянки коду методів рефакторингу.

Рефакторинг в Visual Assist є крос-проектним, причому навіть у тих випадках, коли проекти написані на різних мовах. Шаблони генеруємого при рефакторингу коду настроюються в настроюваннях. Команда UNDO дозволяє скасувати зміни, внесені в код при рефакторингу, причому всі зміни відміняються скопом, "в один клік".

Refactor! і Refactor! Pro

Компанія Developer Express, розроблювач відомого плагіна CodeRush, розробила цілий набір плагінів для рефакторингу – три безкоштовних плагіна Refactor! для C++, Visual Basic і ASP.NET і один платний -Refactor! Pro, що підтримує вісім мов C#, Visual Basic, C++, ASP.NET, XAML, XML, HTML, JavaScript. Плагіни здатні працювати в VS 2002-VS2008. Платна версія реалізує в цілому близько 150 інструментів рефакторингу, функціональність безкоштовних плагінів трохи більше скромна – 15 інструментів для C++, 36 для VB, 28 для ASP.NET.

Так само як Visual Assist, Refactor! показує іконку, що відкриває меню зі списком доступних, для обраної ділянки коду, методів рефакторингу. Однак, інтерфейс інструментів рефакторингу в Refactor! відрізняється принципово – він є бездіалоговим, інтерактивним і анімованим. Досить вибрати в списку методів рефакторингу необхідний, як Refactor! відобразить спливаючу підказку з інструкціями які клавіші натискати й покаже яким образом при цьому буде

змінений код. Коли потрібно вводити текст, наприклад, нова назва змінної, користувач робить це прямо в редакторі студії – Refactor! автоматично змінює всі посилання на цю змінну в коді програми.

Рисунок 2.2 – Бездіалоговий інтерфейс Refactor!

Порівняльне тестування безкоштовної й платної версії Refactor! для C++ показало, що безкоштовна версія не тільки має меншу кількість функцій, але й працює якось менш упевнено. У процесі роботи дуже часто для обраних ділянок коду не було доступно ніяких інструментів рефакторингу, навіть тривіального Rename. У платній версії такого не спостерігалося.

ReSharper

ReSharper – плагін, розроблений фірмою JetBrains, призначений для розроблювачів, що використовують C# і VB.NET і величезний набір, що надає, різноманітних корисних функцій, у число яких входять і засобу рефакторингу.

Так само як Visual Assist і Refactor!, ReSharper уміє показувати контекстно-залежне меню зі списком інструментів рефакторингу, доступних для обраної ділянки коду. Для виклику меню використовується комбінація клавіш

Ctrl+Shift+R. Це зручно – виклик будь-якого інструмента рефакторингу завжди виконується за допомогою однієї й тої ж комбінації клавіш.

Інструменти рефакторингу в ReSharper мають діалоговий інтерфейс, однак діалоги немодальні. Небагато напружує відсутність можливості попереднього перегляду в деяких діалогах. Втім, внесені зміни при необхідності легко скасувати – одна команда UNDO скасовує всю проведену операцію рефакторингу цілком.

Усього доступно 34 методу рефакторингу, включаючи такі як "Move Type to Another File", "Inline variable", "Copy Type" і "Extract Superclass". Реалізовано вони на гарному рівні, користуватися ними зручно.

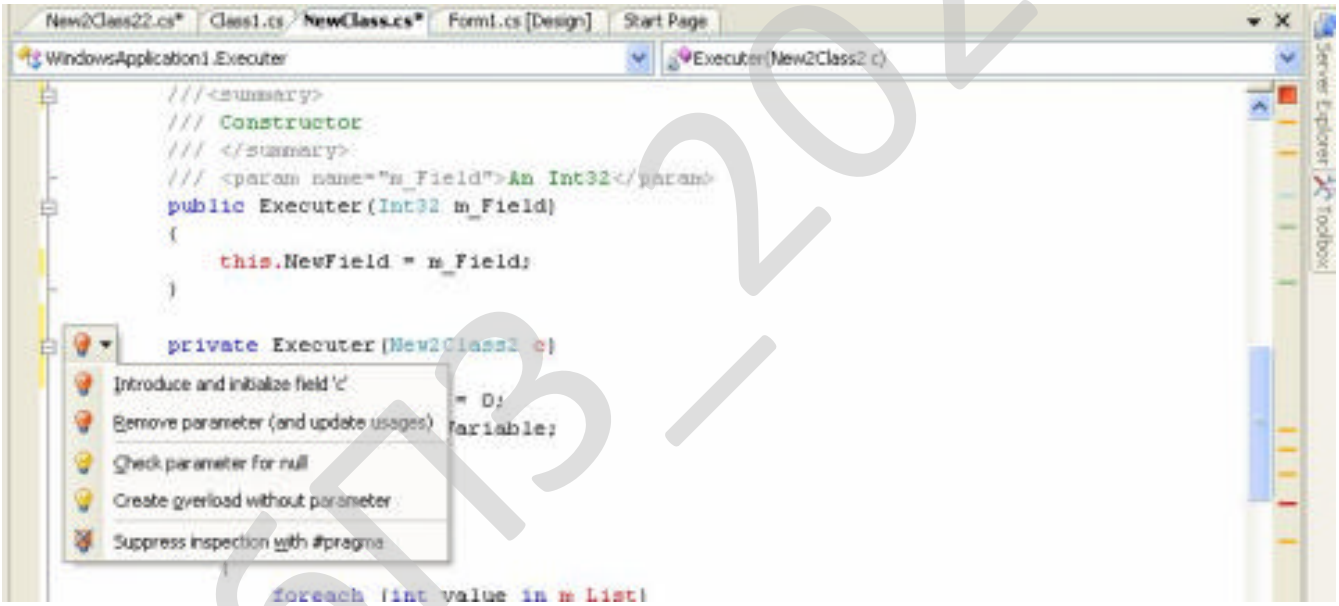

Рисунок 2.3 – Контекстні дії в ReSharper

Крім властиво засобів рефакторингу, ReSharper надає величезний набір так званих контекстних дій (context actions). Фактично – це дрібні операції по перетворенню коду, що перебуває в позиції курсору. Наприклад змінити область видимості поточного методу або конвертувати цикл foreach у цикл for. Для виклику списку доступних у поточному контексті дій досить нажати Alt+Enter або клацнути по значку лампочки, що з'являється на початку рядка.

JustCode!

Плагін JustCode!, розроблений фірмою Omnicore, представляє із себе альтернативу ReSharper і Refactor! Плагін підтримує VS2003 і VS2005, мови C#, J#, VB.NET, ASP.Net, JavaScript і HTML.

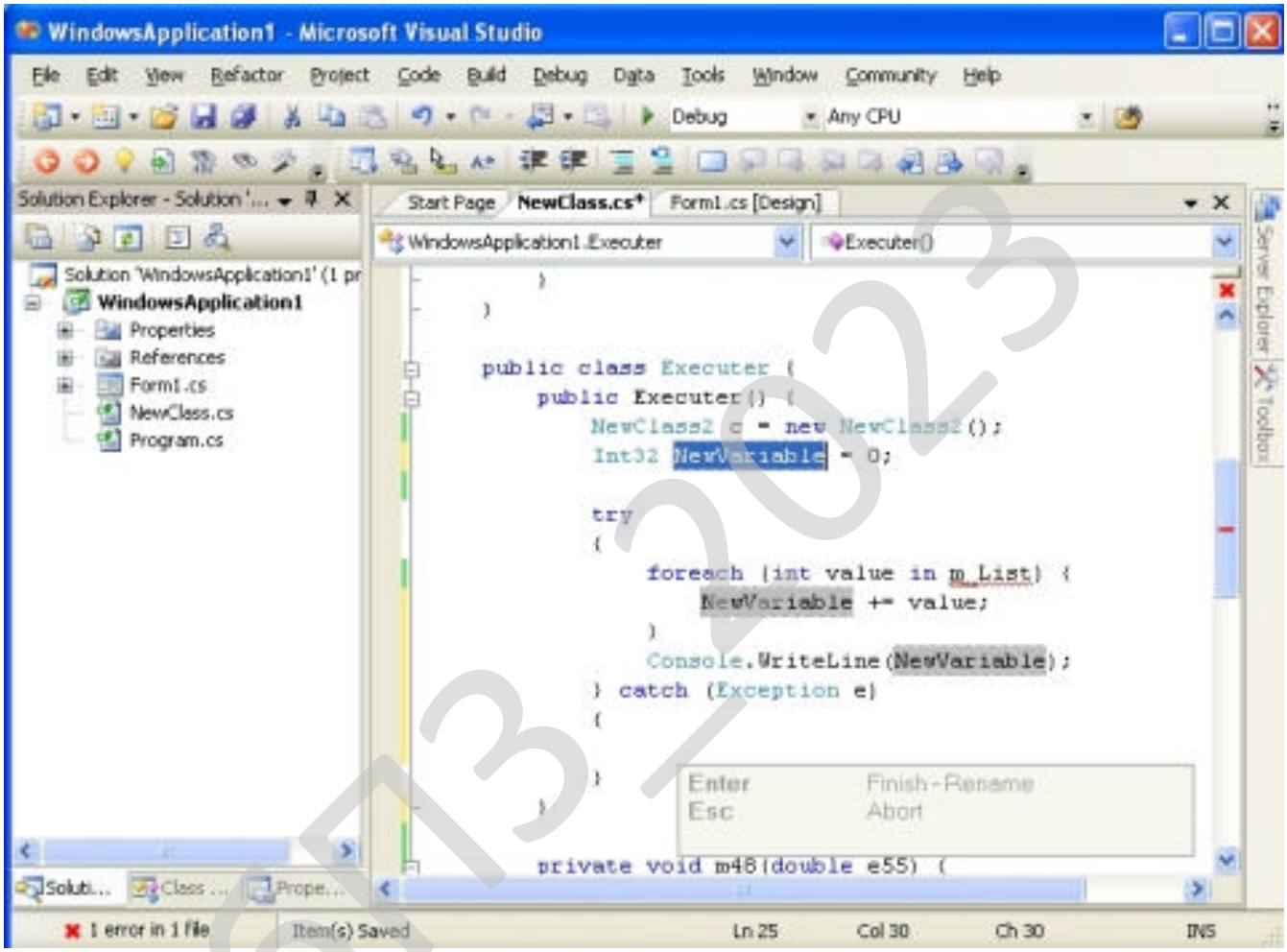

Рисунок 2.4 – Бездіалоговий інтерфейс JustCode!. Процес перейменування змінної

Набір операцій рефакторингу, доступних у плагіні, не занадто великий – "Rename", "Inline variable", "Extract Method", автоматичне створення конструктора й перетворення змінної-члена класу у властивість.

Інтерфейс JustCode! схожий з інтерфейсом Refactor!, правда в ньому поменше анімації. Точно так само набір доступних інструментів рефакторингу

залежить від поточного контексту. Діалогових вікон ні, всі операції рефакторингу можна проводити використовуючи винятково клавіатуру.

До плюсів плагіна ставиться крос-проектність і крос-мовність рефакторингу. До мінусів – неповноцінна реалізація команди скасування операції рефакторингу – кожна зміна в коді потрібно скасовувати окремим викликом UNDO.

Швидкість роботи плагіна цілком на рівні, ніяких особливих "гальм" у роботу студії він не вносить. Однак бездіалоговий інтерфейс в JustCode! поки налагоджений не так чітко, як в Refactor!, до нього доводиться пристосовуватися.

CodeIt.Once

Плагін CodeIt.Once розроблений фірмою Submain і призначений винятково для проведення рефакторингу. Підтримує мови C# і VB.NET, здатний працювати з VS.

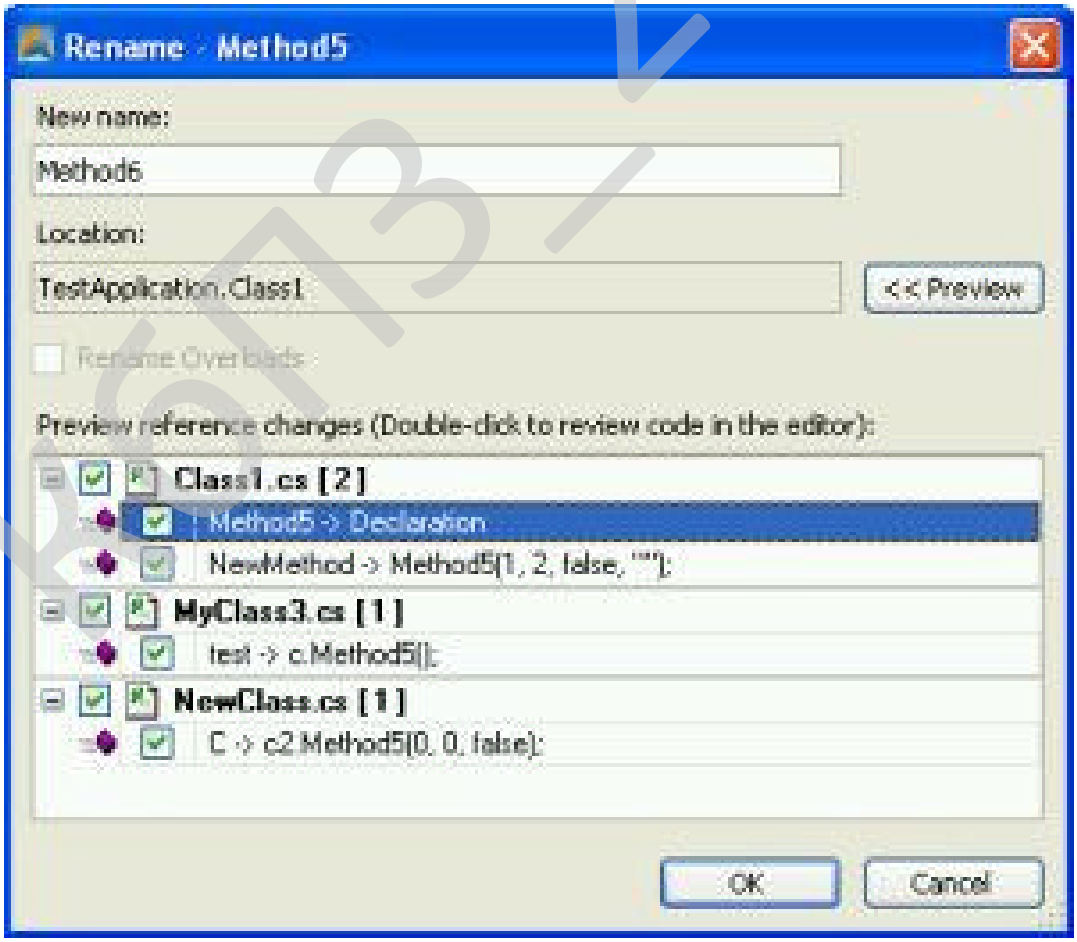

Рисунок 2.5 – Rename в Code.Once

Codeit.Once реалізує 14 інструментів рефакторингу. Трохи несподіваним є їхній інтерфейс, установлений за замовчуванням – багатокрокові візарди (wizards), прямо скажемо, не найшвидший варіант інтерфейсу. На щастя, режим використання візардов можна відключити в настроюваннях, після чого інтерфейс стає діалоговим, що нагадує Visual Assist. Діалоги досить компактні, зручні в роботі й мають функцію передперегляду.

Швидкість і якість роботи методів рефакторингу в Codeit.Once не викликають дорікань. Команда Undo реалізована грамотно – скасовує в коді відразу всі зміни, внесені при виконанні операції рефакторингу.

Resource Refactoring Tool

Resource Refactoring Tool безкоштовний плагін, що автоматизує одну єдину функцію: перенос рядків, що жорстко зашитих у код, у файл ресурсів. Він підтримує мови C# і VB.NET, працює в VS2005/2008 у всіх типах проектів, включаючи web-додатка.

TytanNET v0.14

Плагін TytanNET так само безкоштовний. Підтримує мови C#, VB.NET, Managed C++, здатний працювати в VS2005/2008. Поточна версія надає всього чотири методи рефакторингу, однак у наступній версії їх буде більше. Варто відзначити, що швидкість роботи плагіна поки залишає бажати кращого.

Виводи

На закінчення хотілося б відзначити два моменти. По-перше, всі розглянуті вище плагіни займаються фоновим аналізом коду, багато які з них – його фоновою компіляцією. Так що встановлювати занадто велику кількість плагінів одночасно не рекомендується – можна одержати серйозну з роботи Visual Studio.

По-друге, рефакторинг – це ціпок на два кінці. З одного боку – потрібний набір інструментів для його проведення, з іншого боку – не перешкодить набір засобів, що дозволяють автоматично виявляти місця коду, що бідують у проведення рефакторингу.

Такі інструменти в наш час те ж є. Тим хто зацікавиться можна порекомендувати звернути увагу на NDepend, FXCop і Reflector і, звичайно ж, заглянути на сайт www.visualstudiogallery.com, на якому представлена маса корисних плагінів для Visual Studio.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за

допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL,

включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

– Велика кількість поліпшень STL від Dinkumware.

– Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.

– Поліпшена підтримка Cmake.

– Велика кількість виправлень для підвищення стабільності і якості.

– Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.

– Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній

формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Маючи справу із застарілим програмним забезпеченням, важливо розуміти, що можна зробити з програмним забезпеченням. Застаріле програмне забезпечення часто складається з програмного забезпечення, яке залишалося працювати протягом тривалого часу без надто багатьох внутрішніх змін, стратегія «не виправляти те, що не зламано». Оскільки компілятори таких мов, як Fortran, є зворотно сумісними, часто можна скомпілювати та запустити ці старі програми. Але в якийсь момент виникає необхідність мати справу зі старим кодом. Отже, як цього досягти? *Чи підлягає переробці* чи *рефакторингу* код ?

Реінжиніринг означає внесення фундаментальних змін до коду. Ось три основні методи реінжинірингу:

1. Портування – програми модифікуються для роботи на новій апаратній платформі.

2. Переклад – програми перекладаються із застарілої мови на сучасну.

3. Міграція – програми конвертуються зі старої мови на новіший діалект.

По суті, це нічим не відрізняється від роботи, яку проводили б у старій будівлі. Її можна повністю перенести на нове місце, повністю перебудувати або зробити новою, включивши лише фасад оригінальної будівлі.

Рефакторинг, з іншого боку, залишає речі недоторканими. Рефакторинг передбачає зміну частини програмного забезпечення таким чином, що зовнішня поведінка коду залишається незмінною, але його внутрішня структура та архітектура покращуються. Це схоже на модернізацію сантехніки та електрики старої будівлі. Він все ще функціонує і виглядає так само, але інфраструктура була покращена. Рефакторинг бере під контроль код, що занепадає, покращуючи читабельність і зручність обслуговування існуючого коду. Рефакторинг

виконується, щоб виправити короткі шляхи, усунути дублювання та мертвий код, а також зробити дизайн і логіку зрозумілими. Щоб краще і зрозуміліше використовувати мову програмування. Це не обов'язково означає, що код перенесено на новий діалект мови. Рефакторинг часто є частиною життєвого циклу програмного забезпечення і може не бути націленим конкретно на застарілий код.

Реінжиніринг і рефакторинг виглядають дуже схожими, і є ймовірні сфери, такі як міграція, де вони збігаються. Насправді процес роботи зі застарілим кодом часто починається з рефакторингу та прогресує до реінжинірингу. У ситуаціях, коли база коду надто складна, можливо, варто спробувати підвищити ефективність спочатку шляхом вдосконалення алгоритмів. Однак якщо це не спрацює, можливо, планується реінжиніринг.

Давайте подивимося правді в очі, реінжиніринг програмного забезпечення – це не те, що люблять робити розробники або на що радо погоджуються менеджери. Рефакторинг – це процес зміни програмної системи для покращення її внутрішньої структури без зміни зовнішньої поведінки коду. По суті, це вдосконалення дизайну коду без будь-якої очевидної бізнес-цінності. Звучить контрпродуктивно. Але насправді рефакторинг і реінжиніринг є важливими для еволюції та підтримки програмного забезпечення. Їх головна мета полягає в тому, щоб програмне забезпечення було придатним для обслуговування та корисним протягом усього терміну служби. І хоча рефакторинг коду здається чимось, чим можуть скористатися лише розробники, зрештою це бізнес-рішення. Ось п'ять бізнес-переваг рефакторингу та зворотного проектування.

Вартість реінжинірингу програмного забезпечення

Можливо, це не має сенсу, коли ви чуєте, що переписування вже написаного коду є економічно ефективним, але вислухайте нас. Удосконалення та вдосконалення існуючого коду зараз заощадить ваші гроші. Ось деякі основні математики , зроблені Джоном Вірголіно, щоб довести це. Причина, чому в більшості випадків це правда, проста: кожна система програмного забезпечення

(за винятком, можливо, справді маленьких і базових) будується ітераційно. Ви починаєте з основних і базових функцій, таких як керування користувачами та авторизація, перевірка будь-якого введення користувача, система виставлення рахунків тощо. Після цього ви додаєте функції для бізнесу, такі як багаторівневий каталог, система пропозицій, розширена аналітична система та будь-які інші корисні функції. Таким чином, покращуючи код у ядрі, ви покращуєте наявну та майбутню функціональність. Крім того, ви можете змінити ядро, щоб воно було готове до деяких майбутніх функцій, про які ви не думали півроку тому. Коротше кажучи, легше помістити ще одну коробку в сховище, коли вона акуратна й охайна. Крім того, залежно від системи, витрати на рефакторинг можуть навіть окупитися відразу після його впровадження. Ось чому.

Покращена продуктивність

Досить часто перероблений код працює швидше. Це тому, що розробники вже знають, як поточна система використовує ресурси. Вони вже знають, які запити SQL завжди викликаються разом і можуть комбінуватися. Вони бачили, як система працює у виробництві або в якомусь постановочному середовищі, де навантаження наближене до реальних умов. А хороша продуктивність завжди на користь. Це означає швидшу реакцію системи – те, що ви, ваші клієнти та навіть розробники оціните під час створення нових чудових функцій.

Є багато прикладів, коли рефакторинг програмного забезпечення та реінжиніринг покращили продуктивність системи, і ось три, щоб проілюструвати це .

Знижений ризик

У розробці нового програмного забезпечення завжди є елемент ризику. Це означає, що можуть виникнути проблеми з розробкою, кадровим забезпеченням або специфікацією. Рефакторинг передбачає поступовий підхід до вдосконалення системи, а не радикальну заміну системи. Під час рефакторингу зменшується ймовірність втрати критичних бізнес-знань або створення системи, яка не відповідає потребам користувачів. Оскільки це можна здійснювати поетапно, ваш

бізнес завжди має робочу систему, а кінцевим користувачам надається можливість поступово та легко адаптуватися до оновленої частини. Крім того, поетапна модернізація системи допомагає новим співробітникам дізнатися про потреби вашого бізнесу, недокументовані функції та особливості.

Хоча переробка всього з нуля може вивести ваш бізнес на новий рівень, ризики та витрати, ймовірно, різко зростуть. Отже, рефакторинг може бути не найшвидшим способом розвитку вашого бізнесу, але, мабуть, одним із найбезпечніших. Поступова реінжиніринг програмного забезпечення контролює рівень ризику, каже Майкл Р. Олсем у своїй роботі «Поступовий підхід до реінжинірингу програмних систем». Ключове слово тут «поступове». Без цього реінжиніринг міг би закінчитися двома різними, несумісними програмними системами. Тут вам допоможуть принципи безперервної інтеграції та безперервного розгортання.

Завжди готовий до вдосконалень

Як ми вже зазначали, більшість розробок програмного забезпечення передбачає вдосконалення існуючої системи, а не створення нової. Код має бути структурований таким чином, щоб вітати нові функції. Користувачі звикли до оновлення програмного забезпечення кожні шість місяців або навіть рідше, ринки постійно змінюються, і додавання нових функцій у ваше програмне забезпечення означає бути в курсі.

Макс Канат-Александер, автор «Простоти коду», припускає у своїй статті , що знадобиться менше часу, щоб навести порядок у системі, перш ніж почати додавати нові функції. Щоб довести це, він згадує, як його команда завершувала проекти швидше, ніж команди, які працювали над новими кодовими базами з кращими інструментами, коли це робили.

«Добре, це звучить переконливо», – скажете ви, – але що, якщо я не маю чітких термінів для нових функцій? Що, якщо я пропущу рефакторинг і зосереджуся лише на розробці нового?» Це слизький шлях у розробці програмного забезпечення. Дозвольте познайомити вас з містером Боргом.

Технічний борг.

Дуже рідко архітектура програмної системи є ідеальною з самого початку. Без будь-якого рефакторингу рано чи пізно додати нові функції в систему буде практично неможливо.

Зворотне проектування

Зворотне проектування варто розглянути, якщо існуюча система, яка завоювала довіру та впевненість користувачів, потребує модернізації. У цьому випадку це дозволить створити серйозний програмний продукт в найкоротші терміни. Зворотне проектування також є хорошим тестом безпеки програмного забезпечення. Він широко використовується, щоб переконатися, що система не має будь-яких серйозних недоліків безпеки або вразливостей. Це допомагає зробити систему надійною та захищає її від хакерів і шпигунського програмного забезпечення.

У цьому світі, що постійно змінюється, програмне забезпечення має бути придатним для обслуговування, багаторазового використання та зручним для розширення, щоб задовольнити клієнтів і перемогти конкурентів. Програмне забезпечення стає дедалі складнішим, і ігнорування технічної заборгованості, пропускаючи рефакторинг, з часом повернеться й переслідуватиме ваш бізнес. І якщо ви тільки запускаєте наступну чудову програму, реінжиніринг – це практика, яку варто розглянути. Можливо, цей старий, подряпаний компакт-диск із купою коду на C і ботаном-розробником – це все, що зараз потрібно вашій компанії. Ви можете знайти приклади успішних тривалих програмних проектів, які не мають етапів «рефакторингу». Але це лише тому, що розробники постійно переробляють існуючий код. Вам не потрібно зупиняти процес розробки та планувати рефакторинг. Хороші технічні керівники та архітектори програмного забезпечення все одно роблять це. Отже, якщо у вас є чудова команда розробників, ваш програмний продукт, ймовірно, уже неодноразово перероблявся.

Концепція зворотного проектування та рефакторингу в інженерії програмного забезпечення

Процес зворотного проектування починається з вилучення детальної інформації про проект, а з цього витягується абстракція проекту високого рівня. Детальна (низькорівнева) інформація про проект витягується з вихідного коду та існуючих проектних документів. Ця інформація включає структурні діаграми, описи даних і PDL для опису деталей обробки. Подібний підхід, але автоматизований, описано в інших місцях для відновлення документів Джексона та Уорньєра/Орра з коду. Представлення проекту високого рівня витягується з відновленого детального проекту та виражається за допомогою діаграм потоку даних і потоку керування. У цьому документі термін «відновлений дизайн» буде використовуватися для позначення вилученого дизайну.

Етапи процедури описані нижче:

1. Збирайте інформацію. Зберіть всю можливу інформацію про програму. Джерела інформації включають вихідний код, проектні документи та документацію для системних викликів і зовнішніх маршрутів. Необхідно також визначити персонал, який має досвід роботи з програмним забезпеченням.

2. Вивчіть інформацію. Перегляньте зібрану інформацію. Цей крок дозволяє людині, яка виконує відновлення, ознайомитися із системою та її компонентами. На цьому етапі можна сформулювати план диссемблера програми та запису відновленої інформації.

3. Витягніть структуру. Визначте структуру програми та використовуйте її для створення набору структурних діаграм. Кожен вузол на структурній діаграмі відповідає рядку, який викликається в програмі. Таким чином, діаграма записує ієрархію викликів програми. Для кожного ребра діаграми необхідно записати дані, передані до вузла та повернуті цим вузлом.

4. Функціональність запису. Для кожного вузла на структурній діаграмі запишіть обробку, виконану в програмному рядку, що відповідає цьому вузлу. PDL можна використовувати для вираження функціональності програмних

рядків. Для системних і бібліотечних рядків функціональність може бути описана англійською або більш офіційною нотацією.

5. Запис потоку даних. Відновлену структуру програми та PDL можна проаналізувати для виявлення перетворень даних у програмному забезпеченні. Ці кроки перетворення показують обробку даних, виконану в програмі. Ця інформація використовується для розробки набору ієрархічних діаграм потоків даних, які моделюють програмне забезпечення.

6. Запис контрольного потоку. Визначте керуючу структуру високого рівня програми та запишіть її за допомогою діаграм потоку керування. Це стосується високорівневого контролю, який впливає на загальну роботу програмного забезпечення, а не низькорівневого контролю обробки.

7. Перегляньте відновлений дизайн. Перегляньте відновлений дизайн на узгодженість із доступною інформацією та правильність. Визначте будь-які відсутні елементи інформації та спробуйте знайти їх. Перегляньте дизайн, щоб переконатися, що він правильно представляє програму.

8. Створіть документацію. Завершальним етапом є створення проектної документації. Потрібно буде записати інформацію, що пояснює мету програми, огляд програми, історію тощо. Ці відомості, швидше за все, не міститимуться у вихідному коді та повинні бути відновлені з інших джерел.

«Рефакторинг – це дисциплінована техніка реструктуризації існуючого корпусу коду, зміна його внутрішньої структури без зміни його зовнішньої поведінки. Його серцем є серія невеликих перетворень, що зберігають поведінку. Кожне перетворення (так зване «рефакторинг») робить мало, але послідовність перетворень може призвести до значної реструктуризації. Оскільки кожен рефакторинг невеликий, менша ймовірність того, що він піде не так. Система також підтримує повну працездатність після кожного невеликого рефакторингу, зменшуючи ймовірність того, що система може отримати серйозний злам під час реструктуризації." [Marw Fowler] Рефакторинг використовується для покращення якості коду, надійності та зручності обслуговування протягом життєвого циклу

програмного забезпечення. Дизайн коду та якість коду покращуються завдяки рефакторингу. Рефакторинг також підвищує продуктивність розробника та збільшує кількість повторного використання коду. Наприклад, якщо два методи використовують подібний фрагмент коду, загальний код можна рефакторингувати в інший метод, який потім можуть викликати два батьківські методи.

Рефакторинг програмного забезпечення = «Реструктуризація існуючого коду шляхом зміни його внутрішньої структури без зміни зовнішньої поведінки»

Етапи рефакторингу

Рефакторинг включає:

– Розробку = (додавання функцій, рефакторинг).

– Рефакторинг = (тестування, невеликі кроки).

– Маленькі кроки = один із рефакторингу типи «Зберіть речі разом, коли зміни є разом».

– Витягти методи.

– Перемістити методи.

– Перейменувати методи.

– Замінити Temp на Query.

– Замінити умови на поліморфізм.

– Замінити код типу на State/Strategy.

– Самоінкапсуляція польових додатків рефакторингу.

Ми використовуємо три приклади, щоб поясніть деякі базові рефакторингу

1. Метод вилучення:

– сигналізується коментарями.

– Один вхід, один вихід.

– збільшити рівень опосередкованості.

– зменшити довжину методу.

– збільшити ймовірність повторного використання.

2. Метод Move:

– Метод Place разом з об’єктом; об’єднайте речі разом, коли зміни є разом.

3. Замініть умови поліморфізмом:

– Перемикачі – це «жорсткий код», поліморфізм кращий для програм.

Ми використовуємо три приклади, щоб пояснити деякий базовий рефакторинг

1. Метод вилучення:

– сигналізований коментарями.

– Один вхід, один вихід.

– збільшити рівень опосередкованості.

– зменшити довжину методу.

– збільшити ймовірність повторного використання.

2. Перемістити метод:

– Розмістити метод разом з об'єктом; об’єднайте речі разом, коли зміни є разом.

3. Замініть умови на поліморфізм:

– Перемикачі є «жорстким кодом», поліморфізм кращий для розширюваності в об’єктно-орієнтованих програмах.

3.2 Розробка структурної схеми

Всім відомо що зараз є велика кількість вихідного коду на C/C++. Мови програмування застаріли та їх перестали використовувати та залишилась велика кількість вихідного коду на цих мовах. Цей код не має розвинутого інтерфейсу чи багатофункціональність але в цих кодах є реалізація математичних алгоритмів, високопродуктивних та новаторських підходів у реалізації математичних розв’язків рівнянь.

Незважаючи на те, що ці програми складалися досить давно, ідеї реалізації математичних теорій ніколи не старіють, особливо багато алгоритмічно-математичного вихідного коду використалося у вищих навчальних закладах.

Всі ці наробітки й рішення були заблоковані при старінні мови програмування.

Завдяки цьому розробка напрямку перекладу коду із застарілої мови програмування в більш новий завжди актуальна, код котрий надалі буде більш детально розглядатися та перероблятися під новий лад. Можливо й таке, що код після перетворення не буде у повній мірі робити, але сама структура алгоритму залишиться.

Причини застосування рефакторингу

Рефакторинг потрібно застосовувати постійно при розробці коду. Основними стимулами його проведення є наступні завдання:

– необхідно додати нову функцію, що недостатньо укладається в прийняте архітектурне рішення;

– необхідно виправити помилку, причини виникнення якої відразу не ясні;

– подолання труднощів у командній розробці, які обумовлені складною логікою програми.

Ознаки поганого коду

Багато в чому при рефакторингу краще покладатися на інтуїцію, засновану на досвіді. Проте є деякі видимі проблеми в коді (англ. code smells), що вимагають рефакторингу:

– дублювання коду;

– довгий метод;

– великий клас;

– довгий список параметрів;

– «зайві» функції – це метод, що надмірно звертається до даних іншого об'єкта;

– надлишкові тимчасові змінні;

– класи даних;

– незгруповані дані.

Рефакторинг коду

У програмуванні термін рефакторинг означає зміна вихідного коду програми без зміни його зовнішнього поводження. В екстремальному програмуванні й інших гнучких методологіях рефакторинг є невід'ємною частиною циклу розробки ПЗ: розроблювачі поперемінно те створюють нові тести й функціональність, то виконують рефакторинг коду для поліпшення його логічності й прозорості. Автоматичне юніт-тестування дозволяє переконатися, що рефакторинг не зруйнував існуючу функціональність.

Іноді під рефакторингом неправильно мають на увазі корекцію коду із заздалегідь застереженими правилами відступу, перекладу рядків, внесення коментарів і іншими візуально значимими змінами, які ніяк не відбиваються на процесі компіляції, з метою забезпечення кращої читаності коду (див. code formatting).

Рефакторинг споконвічно не призначений для виправлення помилок і додавання нової функціональності, він взагалі не міняє поводження програмного забезпечення і це допомагає уникнути помилок і полегшити додавання функціональності. Він виконується для поліпшення зрозумілості коду або зміни його структури, для видалення «мертвого коду» – все це для того, щоб у майбутньому код було легше підтримувати й розвивати. Зокрема, додавання в програму нового поводження може виявитися складним з існуючою структурою – у цьому випадку розроблювач може виконати необхідний рефакторинг, а вже потім додати нову функціональність.

Це може бути переміщення поля з одного класу в інший, винесення фрагмента коду з методу й перетворення його в самостійний метод або навіть переміщення коду по ієрархії класів. Кожний окремий крок може здатися елементарним, але сукупний ефект таких малих змін у стані радикально поліпшити проект або навіть запобігти розпаду погано спроектованої програми.

Методи рефакторингу

Найбільше застосовувані методи рефакторингу:

– Зміна сигнатури методу (Change Method Signature).

– Інкапсуляція поля (Encapsulate Field).

– Виділення класу (Extract Class).

– Виділення інтерфейсу (Extract Interface).

– Виділення локальної змінної (Extract Local Variable).

– Виділення методу (Extract Method).

– Генералізація типу (Generalize Type).

– Вбудовування (Inline).

– Введення фабрики (Introduce Factory).

– Введення параметра (Introduce Parameter).

– Підйом методу (Pull Up Method).

– Спуск методу (Push Down Method).

– Перейменування методу (Rename Method).

– Переміщення методу (Move Method).

– Заміна умовного оператора поліморфізмом (Replace Conditional with Polymorphism).

– Заміна спадкування делегуванням (Replace Inheritance with Delegation).

– Заміна коду типу підкласами (Replace Type Code with Subclasses).

Зміна сигнатури методу (Change Method Signature)

Суть зміни сигнатури методу полягає в додаванні, зміні або видаленні параметра методу. Змінивши сигнатуру методу, необхідно скорегувати звертання до нього в коді всіх клієнтів. Ця зміна може торкнути зовнішній інтерфейс програми, крім того, не завжди розроблювачеві, що змінює інтерфейс, доступні всі клієнти цього інтерфейсу, тому може знадобитися та або інша форма реєстрації змін інтерфейсу для наступної передачі їх разом з новою версією програми.

Інкапсуляція поля (Encapsulate field)

У випадку, якщо в класу є відкрите поле, необхідно зробити його закритим і забезпечити методи доступу. Після **«Інкапсуляції поля»** часто застосовується **«Переміщення методу»**.

Виділення методу (Extract Method)

Виділення методу полягає у виділенні з довгого й/або потребуючого коментарів коду окремих фрагментів і перетворенні їх в окремі методи, з підстановкою підходящих викликів у місцях використання. У цьому випадку діє правило: якщо фрагмент коду вимагає коментарю про те, що він робить, то він повинен бути виділений в окремий метод. Також правило: один метод не повинен займати більш ніж один екран (25-50 рядків, залежно від умов редагування), у противному випадку деякі його фрагменти мають самостійну цінність і підлягають виділенню. З аналізу зв'язків виділюваного фрагмента з навколишнім контекстом робиться вивід про перелік параметрів нового методу і його локальних змінних.

Переміщення методу (Move Method)

Переміщення методу застосовується стосовно методу, що частіше звертається до іншого класу, чим до тому, у якому сам розташовується.

Заміна умовного оператора поліморфізмом (Replace Conditional with Polymorphism)

Умовний оператор з декількома галузями заміняється викликом поліморфного методу деякого базового класу, що має підкласи для кожної галузі вихідного оператора. Вибір галузі здійснюється неявно, залежно від того, екземпляру якого з підкласів виявився адресований виклик.

Основні принципи:

– спочатку варто створити базовий клас і потрібне число підкласів;

– у деяких випадках варто провести оптимізацію умовного оператора шляхом «Виділення методу»;

– можливе використання «Переміщення методу», щоб помістити умовний оператор у вершину ієрархії спадкування;

– вибравши один з підкласів, потрібно конкретизувати в ньому поліморфний метод базового класу й перемістити в нього тіло відповідної галузі умовного оператора;

– повторити попередня дія для кожної галузі умовного оператора;

– замінити весь умовний оператор викликом поліморфного методу базового класу.

Проблеми, що виникають при проведенні рефакторингу

– проблеми, пов'язані з базами даних;

– проблеми зміни інтерфейсів;

– труднощі при зміні дизайну.

Засоби автоматизації рефакторингу

Технічні критерії для інструментів рефакторингу:

– бази дані програми;

– дерева синтаксичного розбору;

– точність.

Практичні критерії для інструментів рефакторингу:

– швидкість;

– скасування модифікацій;

– інтеграція з іншими інструментами.

Проблеми при реінжинірингу

Як правило затверджується, що "легше розробити новий програмний продукт". Це зв'язано з наступними проблемами:

– Звичайному програмістові складно розібратися в чужому вихідному коді

– Реінжиніринг найчастіше дорожче розробки нового програмного забезпечення, тому що потрібно забрати обмеження попередніх версій, але при цьому залишити сумісність із попередніми версіями

– Реінжиніринг не може зробити програміст низкою й середньої кваліфікації. Навіть професіонали часто не можуть якісно реалізувати його. Тому потрібна робота програмістів з більшим досвідом переробки програм і знанням різних технологій.

У той же час, якщо споконвічно програма мала строгу і ясну архітектуру, то провести реінжиніринг буде на порядок простіше. Тому при проектуванні як правило аналізується, що вигідніше провести реінжиніринг або розробити програмний продукт "з нуля".

Керування вимогами під час реінжинірингу та рефакторингу

Вимоги до програмних систем

Вимога – це характеристика або умова, якій повинна задовольняти програмні системи.

Функціональні вимоги визначають дії, які повинна виконувати програмні системи, без обліку фізичних обмежень. Тим самим вони визначають поводження системи при уведенні й виводі. Процес виявлення функціональних вимог досить складний і трудомісткий. Це пояснюється наступними причинами:

– таких вимог до системи звичайно багато;

– замовник не завжди здатний чітко сформулювати, чого він хоче від системи;

– вимоги в підсумковому документі повинні бути викладені так, щоб вони однаково розумілися замовником і виконавцем і не допускали неоднозначності;

– між функціональними вимогами можуть бути різні залежності, що ускладнюють керування ними якщо буде потреба внесення змін.

Для подолання цих труднощів застосовується моделювання вимог. Модель вимог дозволяє, по-перше, установити ієрархію вимог, що сприяє кращому розумінню людиною, по-друге, дає наочне графічне подання вимог і залежностей між ними, у третіх, дозволяє зв'язати графічну форму подання з текстової, забезпечуючи людини повною інформацією.

Нефункціональні вимоги не описують поводження програмної системи, але описують її атрибути або атрибути оточення. Нефункціональні вимоги не потрібно включати в модель вимог, але вони повинні бути точно сформульовані. Звичайно нефункціональних вимог не буває багато, однак вони кардинальним образом впливають на вибір архітектури системи.

Керування вимогами – це досить складний і розтягнутий у часі процес. Він триває протягом більшої частини життєвого циклу, оскільки зміни можуть вноситися як під час розробки, так і після здачі системи на етапі досвідченої експлуатації й при супроводі. Причини цього полягають у тім, що вимоги:

– неочевидні;

– виходять із багатьох джерел;

– важко формулюються (мова неоднозначна);

– складаються з безлічі різних деталей;

– нерівнозначні;

– зв'язано один з одним;

– лежать не тільки у функціональній області;

– можуть змінюватися протягом розробки й при супроводі.

Цілі аналізу й моделювання вимог

Цілями аналізу й моделювання вимог є:

– досягнення угоди між розроблювачами, замовниками й користувачами про те, що повинна робити програмні системи;

– досягнення кращого розуміння розроблювачами того, що повинна робити система;

– обмеження системної функціональності;

– створення базису для планування розробки проекту;

– визначення користувальницького інтерфейсу;

– складання специфікації вимог.

Ролі

– Системний аналітик – фахівець організації-розроблювача, що очолює й координує роботи з виявлення й моделювання вимог.

– Розроблювач варіантів використання – фахівець організації-розроблювача, що деталізує й уточнює моделі вимог.

– Зацікавлені особи – люди, що надають інформацію.

– Експерт – представник замовника, що бере участь у розробці моделі вимог.

– Розроблювач користувальницького інтерфейсу – фахівець організації-розроблювача, що створює прототип користувальницького інтерфейсу програмні системи.

Артефакти

Для досягнення поставлених цілей передбачається створення наступних документів:

– Попередня згода – текстовий документ, що описує, що буде включено в програмні системи й що вирішено виключити, тобто, він обмежує системну функціональність. У ньому відбиваються побажання зацікавлених осіб, а також вказуються основні нефункціональні вимоги (наприклад, описується платформа реалізації, точність обчислень, час відгуку).

– Модель вимог служить для досягнення угоди між замовником і розроблювачами, даючи можливість замовникові переконатися в тому, що система буде робити те, що вони очікують, а розроблювачам створити те, що потрібно. Модель вимог дозволяє, по-перше, установити ієрархію вимог, що сприяє кращому розумінню людиною, по-друге, дає наочне графічне подання вимог і залежностей між ними, у треті дозволяє зв'язати графічну форму подання з текстової. Модель включає акторів і варіантів використання, показує, як система взаємодіє з акторами й що вона робить у кожному варіантів використання.

– Специфікація вимог (Software Requirements Specification) – основний документ, використовуваний при проектуванні й розробці програмні системи. Вона включає модель вимог і додаткові специфікації, які являють собою текстовий опис вимог до кінцевого продукту, але не до процесу його розробки й не містять деталей реалізації вимог.

– Прототип користувальницького інтерфейсу забезпечує візуальне подання інтерфейсу користувача із програмні системи.

– Глосарій – текстовий документ, що містить визначення основних понять і термінів, які повинні однаково розумітися замовником і розроблювачем. Джерелами даних для створення перерахованих артефактів можуть, зокрема, служити артефакти, створені при бізнес-аналізі.

Процес

У процесі аналізу й моделювання вимог можна виділити кілька основних етапів.

Початок аналізу під час реінжинірингу та рефакторингу

Спочатку варто визначити коло зацікавлених осіб. Якщо проводився бізнес-аналіз, то вони вже відомі. Збираються побажання зацікавлених осіб до майбутнього програмні системи. Ці побажання аналізуються, визначаються основні властивості й границі програмні системи, досягаються угоди про те, які проблеми повинні бути вирішені.

Результати. Повинен бути складений документ «Попередня згода», що буде відправною крапкою для виконання всіх наступних робіт. На цьому етапі починається створення глосарія. Якщо глосарій був створений при бізнес-аналізі, то він використовується як відправний документ. Оскільки тут мова йде про виявлення вимог до програмних систем, у глосарій можуть включатися терміни, що відносяться до реалізації (наприклад, браузер, сервер і ін.). Визначення цих термінів повинні сприяти кращому розумінню системи зацікавленими особами.

Побудова моделі вимог

Ця робота припускає виявлення акторів, варіантів використання й взаємодій між ними. Якщо було проведене обстеження організації, то як джерело для визначення акторів і варіантів використання може служити бізнес-модель.

Якщо число варіантів використання занадто велике, то для спрощення читаності й підтримки моделі доцільно розділити їх по пакетах. Це також спрощує розуміння моделі й розподіл відповідальності шляхом призначення розроблювачів, відповідальних за пакети варіантів використання. Пакети дозволяють організувати ієрархію вимог. Верхній рівень ієрархії звичайно визначається, виходячи з основних видів діяльності організації. Якщо був виконаний бізнес-аналіз, то основні види діяльності вже представлені в бізнес-моделі у вигляді пакетів, і вони можуть бути просто скопійовані в модель вимог. Пакетів верхнього рівня не повинне бути багато. Доцільно виділити пакет адміністрування й пакет допоміжних дій, і 2 – 4 пакети, основних видів діяльності. У свою чергу, пакети верхнього рівня можуть включати пакети наступного рівня й т.д.

Коли ви окреслите вихідну модель вимог, ви повинні уважно перевірити, що вона покриває всі заявки зацікавлених осіб, представлені в документі «Попередня згода».

Деталізація моделі вимог

Цілі даної діяльності:

– добування з варіантів використання характерних фрагментів, які можуть розглядатися як окремі абстрактні варіантів використання. Прикладами таких частин можуть бути загальні фрагменти, необов'язкові фрагменти й виключення;

– виявлення нових абстрактних акторів, які грають ролі, поділювані декількома акторами;

– реструктуризація моделі вимог;

– детальний опис потоків подій для варіантів використання;

– завдання пріоритетів варіантів використання.

Коли ви просунетеся досить далеко, ви можете захотіти переглянути вихідну модель, оскільки існує тенденція вводити спочатку занадто багато акторів. Варто пам'ятати, що будь-яка модифікація акторів може викликати істотні зміни в системному інтерфейсі й поводженні.

Деталізація варіантів використання припускає розробку діаграм послідовностей, кооперацій або діяльностей, що описують виконання основної й допоміжної робіт у конкретному варіантів використання.

Визначення пріоритетів означає, що все варіантів використання ранжируються на критичні, важливі й допоміжні. Критичні варіантів використання, представляють основну системну функціональність або мають істотне архітектурне значення. Важливі варіантів використання визначають такі системні функції, як збір статистики, складання звітів, контроль і функціональне тестування. Якщо вони не реалізовані, то система може виконувати своє призначення, але зі значно гіршою якістю сервісу. Допоміжні варіантів використання визначають зручність використання програмні системи.

Складання додаткових специфікацій

Додаткові специфікації являють собою текстові описи вимог. Вони доповнюють модель вимог і поряд з нею включаються в підсумковий документ – специфікацію вимог до програмних систем. Варто чітко розуміти, що кожний варіантів використання є деяку ієрархічну вимогу до програмних систем. Дуже важливо, щоб між моделлю вимог і описом вимог у специфікації була точна відповідність. Якщо варіантів використання простій, то в специфікації він описується як окрема функціональна вимога. У більше складних випадках варіантів використання може бути розбитий на трохи більше простих варіантів (на діаграмах це можна зробити за допомогою пакетів, увівши ще один рівень ієрархії, у специфікації – увести наступний рівень нумерації).

Проектування користувальницького інтерфейсу

Цей процес виконується для того щоб замовник міг більш точно уявити собі роботу й можливості майбутньої програмної системи й видати свої

зауваження й уточнення вимог. Залежно від складності проекту й рівня підготовленості замовника результати цих робіт можуть бути представлені в різних формах:

– програмна реалізація, що відтворює точний вид екранних вікон;

– альбом екранних форм;

– модель навігації екранів у вигляді діаграм класів із вказівкою атрибутів – полів і операцій – кнопок.

Створення специфікації вимог

Специфікація вимог створюється на основі моделі вимог і додаткових специфікацій. Вона затверджується керівництвом замовника й розроблювача й служить основним відправним документом для проектування й розробки. Зокрема, модель вимог, що входить у неї, надалі буде розвинена в модель аналізу й дизайну.

Аналіз і проектування під час реінжинірингу та рефакторингу

Мета й завдання аналізу й проектування

Ціль процесу аналізу й проектування складається в розробці технічних інструкцій, що пропонують, як реалізувати програмні системи, що задовольняє сформульованим вимогам. Для цього варто добре зрозуміти вимоги до програмних систем і перетворити їх у проект системи, вибравши правильну стратегію реалізації. На ранніх стадіях процесу повинна бути створена стійка архітектура, на основі якої можна спроектувати програмні системи, легку для розуміння, побудови й розгортання. Архітектура повинна бути погоджена із середовищем реалізації з метою задоволення вимог до продуктивності, стійкості, безпеці, розширюваності й тестуємості.

До числа розв'язуваних завдань при цьому відносяться:

– розробка точної архітектури розподіленої програмної системи;

– перетворення моделі вимог у модель проектну розроблювальної системи;

– адаптація проекту системи до середовища реалізації з метою підвищення продуктивності розробки;

– вибір механізмів реалізації й визначення обмежень на реалізацію;

– розробка компонентної структури;

– розподіл компонентів по вузлах.

Головним завданням аналізу є перетворення вимог у форму, зрозумілу розроблювачеві, тобто, визначення підсистем, компонентів і класів, за допомогою яких реалізується необхідне поводження програмні системи. В основі такого перетворення лежать варіантів використання, створені при визначенні вимог до програмних систем. При цьому розглядаються тільки функціональні вимоги й ігноруються нефункціональні.

Проектування – це уточнення результатів аналізу, спрямоване на оптимізацію з урахуванням обмежень, що накладаються нефункціональними вимогами, середовищем реалізації й т.д.

Ролі

Системний архітектор – керує роботами з аналізу й проектування програмні системи. Він визначає загальну структуру кожного архітектурного подання, декомпозицію подань, угруповання елементів і інтерфейси між групами.

Розроблювач – проектує класи й відносини між ними Він визначає, як погоджувати класи із середовищем реалізації.

Розроблювач БД – відповідає за проектування бази даних програмні системи.

Артефакти

У процесі аналізу й проектування створюються наступні документи:

Модель проектування – це основна модель програмні системи. Вона описує підсистеми, пакети, компоненти, інтерфейси й класи, а також їхньої взаємодії, що забезпечують необхідне поводження програмні системи.

Документ «Архітектура програмні системи», у якому зібрані різні архітектурні подання програмні системи.

Модель даних – це опис структури даних, збережених у БД (наприклад, реляційна модель даних).

Технологічний процес

Виконання деяких діяльностей залежить від фази розробки, що показано у вигляді коментарів на діаграмі діяльностей.

Визначення потенційної архітектури. Дана діяльність включає архітектурний аналіз і аналіз варіантів використання. Визначається первісний набір архітектурно значимих елементів і механізмів реалізації, виконується початкова розбивка на рівні, визначається структура системи, вибираються варіантів використання, які будуть реалізовуватися в першій ітерації фази розвитку проекту. У результаті створюється ескіз архітектури програмні системи. На основі аналізу архітектурно значимих варіантів використання визначаються основні класи, які включаються в модель аналізу. У модель аналізу включаються діаграми, що описують взаємодію основних класів.

Уточнення архітектури. Діяльність включає визначення механізмів проектування, елементів проекту, об'єднання існуючих елементів проекту, опис архітектури реального часу (якщо проектована програмної системи відноситься до цього класу). У результаті виконання цих робіт досягаються наступні цілі:

– Забезпечується перехід від аналізу до проектування шляхом визначення з елементів і механізмів аналізу елементів і механізмів проектування.

– Підтримується цілісність і несуперечність архітектури шляхом інтеграції нових елементів проекту, обумовлених у поточній ітерації, із уже існуючого й повторного використання доступних елементів проекту.

– Здійснюється плавний перехід від проектування до реалізації.

Аналіз поводження. Ця діяльність включає аналіз варіантів використання, визначення елементів проекту й огляд проекту. Ця діяльність має на меті перетворення описів поводження у вигляді варіантів використання в набір елементів проекту (класи, відносини, операції й ін.).

Проектування компонентів. Цілі даної діяльності складаються в:

– Визначенні й уточненні елементів проекту шляхом докладного опису того, як ці елементи реалізують необхідне поводження.

– Визначенні й уточненні реалізації варіантів використання на основі нових елементів проекту.

– Контролі й рецензуванні проекту в міру його розвитку.

Проектуються варіантів використання, підсистеми, класи й компоненти програмні системи. Точно описуються інтерфейси компонентів і їхня реалізація.

Проектування БД. Дана діяльність виконується для проектів, що використовують бази даних. Вона включає:

– Визначення персистентних (постійно збережених) класів.

– Проектування структури БД для зберігання таких класів.

– Визначення механізмів і стратегій зберігання й доступу до збережених даних, що задовольняють вимогам до продуктивності й надійності програмні системи.

Аналіз і проектування зв'язує керування вимогами й реалізацію. У цьому технологічному процесі створюється модель проектування. Одне з її подань – логічна модель – відбиває декомпозицію програмні системи в набір логічних елементів (класи, підсистеми, взаємодії). Процедурне подання відображає ці елементи в процеси й підпроцеси системи. Подання розгортання відображає ці процеси в набір вузлів обчислювального комплексу, на яких вони виконуються.

Реалізація під час реінжинірингу та рефакторингу

Цілі процесу реалізації

Основні цілі процесу можна сформулювати в такий спосіб:

– Визначити структуру коду у вигляді рівнів.

– Реалізувати компоненти, класи й об'єкти.

– Провести блокове тестування компонентів.

– Інтегрувати розробки, виконані окремими розроблювачами, у єдину систему, що виконується.

У процес реалізації не включене тестування всієї програмні системи, для якого в RUP передбачений окремий процес.

Особливості процесу реалізації

RUP припускає заелементну інтеграцію протягом усього життєвого циклу. Це означає, що коди пишуться невеликими блоками, після чого вони поєднуються в єдине ціле шляхом поступового додавання блоків. Це спрощує процес локалізації помилок. Передбачено два рівні інтеграції – інтеграція результатів роботи групи розроблювачів у підсистему й інтеграція підсистем у програмні системи. Інтеграція відбувається в кожній ітерації відповідно до плану ітерації, де визначені варіантів використання, які проектуються й реалізуються в цій ітерації. Таким чином, план ітерації визначає класи, які будуть реалізовані в цій ітерації.

У фазі конструювання створюється еволюційний прототип системи, що згодом розвивається в кінцеву програмні системи. Це прототип використовується для демонстрації фрагментів програмні системи замовникові й керівництву. За результатами подання прототипу можна одержати зауваження, які дозволяють уточнити, змінити або доповнити вимоги до програмних систем. RUP декларує можливість створення, крім еволюційних, поведінкових одноразових прототипів для проведення певних досліджень, що стосуються функціональних можливостей системи.

В RUP декларується необхідність відповідності моделі й програмного коду. При цьому допускається можливість зміни коду з наступною переробкою моделі, що забезпечувала б необхідну відповідність. Для цієї цілі використовують інструментальні засоби, що включають можливості автоматичного реінжинірингу

Ролі

Конструктор (кодувальник) розробляє компоненти й класи, виконує блокове тестування.

Системний інтегратор виконує інтеграцію елементів у програмні конструкції (систему й підсистеми).

Архітектор визначає структуру реалізації (організацію рівнів і підсистем).

Рецензент коду перевіряє якість програмного коду і його відповідність стандартам проекту.

Артефакти

Підсистема реалізації – це набір компонент і інші підсистеми реалізації, використовуваних для утворення моделі реалізації. Це поняття дозволяє ієрархічно представити модель реалізації.

Компонент частина програмного коду (див. статтю «Компонентно-компонентно-базована розробка»).

План інтеграції документ, що визначає порядок реалізації компонентів і підсистем.

Процес

Створення моделі реалізації. Ця діяльність виконується у фазі розвитку. Метою її є побудова моделі реалізації, що дозволить щонайкраще виконати розробку й кодування. При цьому кінцевий продукт буде створюватися за допомогою послідовно, що укрупнюються прототипів.

Планування ітерації. Для кожної ітерації треба визначити, які підсистеми повинні бути реалізовані в цій ітерації й порядок їхнього інтегрування. За яку підсистему відповідає конкретний конструктор, що визначає порядок реалізації класів.

Реалізація компонентів. Виконується написання вихідних кодів програм, їхня компіляція, формування коду, що виконується, і блокове тестування. Блокове тестування виконує сам кодувальник. Це, по суті, автономне налагодження розроблених програм. При виявленні помилок у вихідний код вносяться зміни, після чого зазначені дії повторюються. Після завершення блокового тестування перевіряється якість вихідного коду і його відповідність прийнятим стандартам проекту.

Інтеграція підсистем. Якщо підсистема розробляється групою виконавців, виконується інтеграція компонентів, розроблених всіма членами групи.

Інтеграція підсистеми завершується створенням набору програмних конструкцій підсистеми, кожна з яких тестується окремо.

Інтеграція системи. Виконується інтеграція системи шляхом послідовного додавання в неї нових підсистем, створених у рамках поточної ітерації. Інтеграція підсистеми завершується створенням набору програмних конструкцій підсистеми, кожна з яких тестується окремо. Для тестування системи в RUP передбачений окремий процес.

Тестування під час реінжинірингу та рефакторингу

Тестування – це процес, що дозволяє оцінити якість виробленого продукту. Якісний програмний продукт повинен відповідати пропонованим до нього вимогам як функціональним, так і нефункціональним. програмні системи повинна реалізовувати всі необхідні варіантів використання й не мати дефектів – відмінностей реально існуючих властивостей або поводження від необхідних. Крім того, програмні системи повинна мати властивості надійності (повинні бути відсутніми зависання, аварійні відмови та ін.), безпеки, забезпечувати потрібну продуктивність, бути зручної в експлуатації, розширюваної й т.д. Таким чином, тестування являє собою процес аналізу програмних систем, спрямований на виявлення дефектів і на оцінку властивостей програмних систем.

Цілі процесу тестування

Метою тестування є оцінка якості програмного продукту шляхом:

– Перевірки взаємодії компонентів.

– Перевірки правильності інтеграції компонентів.

– Перевірки точності реалізації всіх вимог і виявлення дефектів.

Особливості процесу тестування в RUP

Тестування – це ітеративний процес, виконуваний у всіх фазах життєвого циклу. Робота над тестами починається із самого початкового етапу виявлення вимог до майбутнього продукту й тісно інтегрується з поточними завданнями. На кожну ітерацію визначається мета тестування й методи її досягнення. Наприкінці

кожної ітерації визначається, наскільки ця мета досягнута, чи потрібні додаткові випробування, чи не варто змінити принципи й інструменти тестування.

Кожний знайдений дефект реєструється в базі даних проекту з описом ситуації, у якій він знайдений. Аналітик визначає, чи дійсно це дефект і чи не є він повтором виявленого раніше дефекту. Знайденому дефекту привласнюється пріоритет, що визначає важливість виправлення. Конструктор, відповідальний за розробку підсистеми, компонента або класу, або інший виконавець, призначений керівником, приступає до усунення дефекту. Порядок виправлення дефектів регулюється їхніми пріоритетами. Тестувальник повторює виконання тестів і переконується (або не переконується) в усуненні дефекту.

Ролі

Розроблювач тестів відповідає за планування, розробку й реалізацію тестів. Він створює план і модель тестування, методики випробувань і виконує оцінку результатів тестування.

Тестувальник (випробувач) відповідає за виконання системного тестування. У його обов'язку входить настроювання й виконання тестів, оцінки виконання тесту, відновлення після помилок, реєстрація виявлених дефектів.

Артефакти

У процесі тестування створюються наступні документи:

План тестування – документ, що визначає стратегію тестування в кожній ітерації. Він містить опис цілей і завдань тестування в поточній ітерації, а також стратегій, які будуть використовуватися. У плані вказується, які будуть потрібні ресурси, і приводиться перелік тестів.

Модель тестування – це подання того, що і як буде тестуватися. Модель включає набір контрольних завдань, методик випробування, сценаріїв випробувань і очікуваних результатів (test cases), тестових скриптів і описів взаємодій тестів.

– Контрольне завдання – набір тестових даних, умов виконання тестів і очікуваних результатів.

– Методика випробувань – документ, що містить вказівки по настроюванню й виконанню контрольних завдань, а також по оцінці одержуваних результатів.

– Сценарій тестування – це спрощений опис тесту, включаючи вихідні дані, умови й послідовності виконання дій і очікувані результати.

– Тестовий скрипт є програмою, виконуваної при автоматичному тестуванні за допомогою інструментальних засобів тестування.

– Опис взаємодії тестів являє собою діаграму послідовностей або кооперацій, що відбиває впорядкований у часі потік повідомлень між компонентами тестів і об'єктом тестування.

Результати тестування й дані, отримані в процесі виконання тестів.

Модель робочого навантаження використовується для моделювання зовнішніх функцій, виконуваних кінцевими користувачами, обсягів цих функцій і навантаження, створюваної цими функціями. Модель призначається для проведення навантажувального й/або стресового тестування, що імітує роботу системи в реальних умовах.

Дефекти – це опису виявлених при проведенні тестування фактів невідповідності системи пропонованим вимогам. Вони являють собою вид запитів на внесення змін.

Процес

Роботи з тестування виконуються в кожній ітерації у всіх фазах, але цілі й завдання в різних фазах проекту істотно різні.

Фаза входження в проект. У цій фазі виконується підготовка до тестування. Вона включає:

– Створення плану тестування, що містить вимоги до тестів і стратегії тестування. Може створюватися єдиний план для всіх видів тестування (функціональне, навантажувальне й т.д.) або окремі плани для кожного виду.

– Аналіз обсягу тестування.

– Формулювання критеріїв якості й завершення тестування.

– Установку й підготовки до роботи інструментальних засобів тестування.

– Формулювання вимог до проекту розробки програмні системи, обумовлених потребами тестування.

Фаза розвитку. В ітераціях цієї фази починається побудова моделі тестування й пов'язаних з нею артефактів. Оскільки в цій фазі вже присутня модель варіантів використання можна починати проектувати сценарії тестування. У той же час недоцільне виконання тестів, оскільки звичайно в цій фазі ще не існує завершених фрагментів програмні системи. Виконуються наступні діяльності:

– Створення плану тестування для кожної ітерації.

– Розробка сценаріїв тестування.

– Створення заготівель тестових скриптів.

– Розробка контрольних завдань.

– Розробка методики випробувань.

– Розробка моделі робочого навантаження.

Фаза конструювання. У цій фазі з'являються завершені фрагменти систем і прототипи, які повинні тестуватися. При цьому практично в кожній ітерації перевіряються всі модулі (як раніше розроблені й протестовані, так і нові, додані в поточній ітерації). Тести, застосовані в попередніх ітераціях, використовуються й на наступних для регресійного тестування, тобто для перевірки того, що раніше реалізована функціональність системи збереглася в новій ітерації. Виконуються наступні діяльності:

– Створення плану тестування для кожної ітерації.

– Уточнення й доповнення моделі тестування.

– Виконання тестів.

– Опис виявлених дефектів.

– Опис результатів тестування.

– Оцінка результатів тестування.

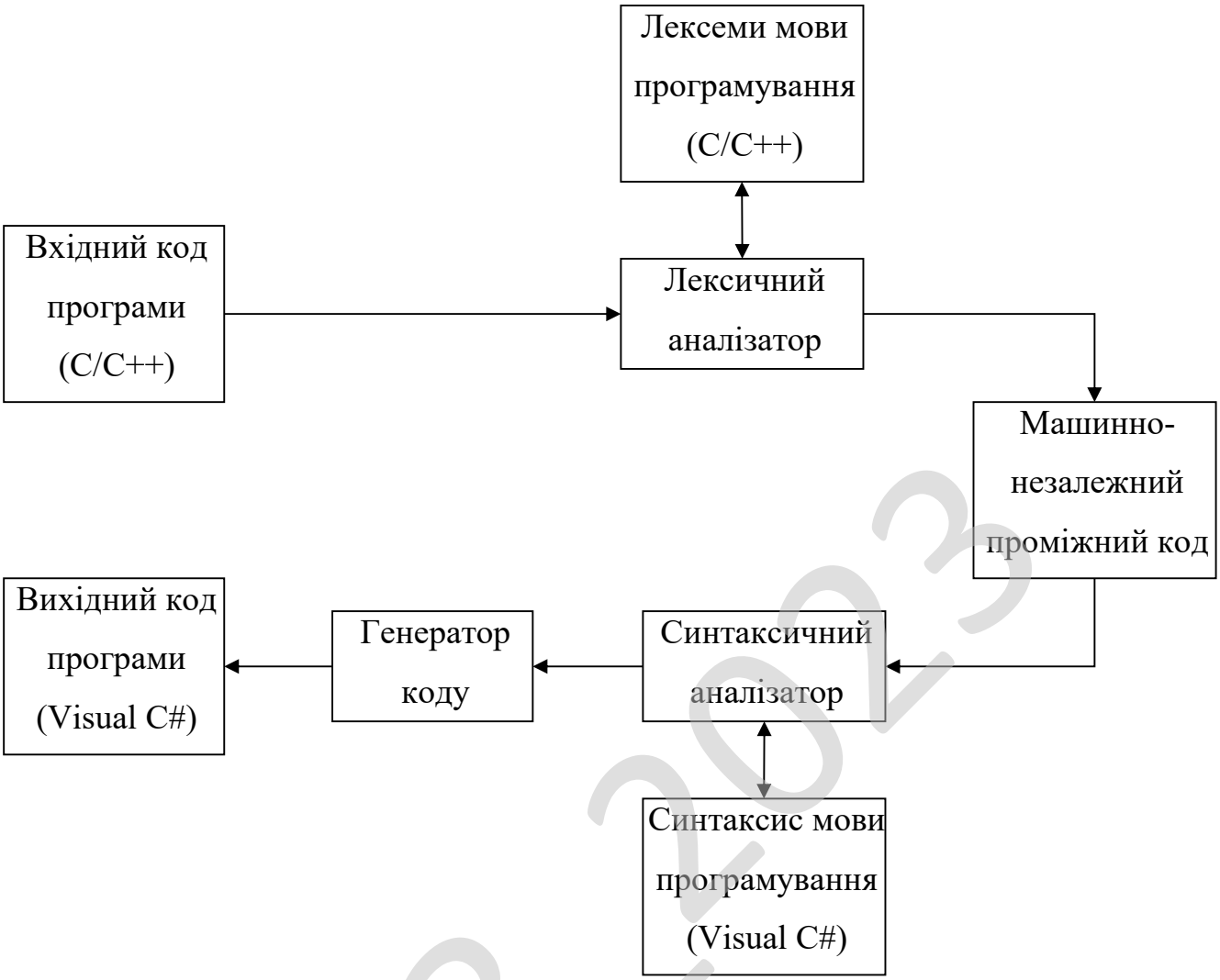

Рисунок 3.1 – Структурна схема системи

За результатами тестування вносяться зміни в програмний код з метою усунення виявлених дефектів, після чого тестування виконується повторно.

Фаза розгортання. В ітераціях цієї фази виконується тестування всієї програмні системи як програмного продукту. Виконувані діяльності аналогічні діяльностям попередньої фази. Виявлення дефектів визначає необхідність внесення змін і повторного тестування. Ітераційний процес повторюється доти, поки не будуть виконані критерії завершення тестування.

Оцінка результатів тестування виробляється на основі метрик тестування, що дозволяють визначити якість тестируемої програмні системи й самий процес тестування.

У розробленій, в результаті виконання магістерського проектування, програмі використається переклад частин вихідного коду з мови програмування C/C++ у Visual C#.

На рисунку 3.1 зображена структурна схема системи де розглянута будова транслятора. При розробці схеми транслятора був проведений аналіз з теорії трансляторів [1-10].

З рисунку добре видно що транслятор розбито на декілька блоків а саме – блок лексичного аналізатора який взаємодіє з набором лексем мови програмування C/C++, проміжного машино-незалежного коду, синтаксичного аналізатора з синтаксисом мови Visual C# та генератора коду. Детальний розгляд роботи цих компонентів розглянуто на функціональній схемі. На вхід транслятора поступає код C/C++, перед тим як запустити його у лексичний аналізатор (почати його оброблювати) проходить перевірка коду, що це дійсно є код C/C++. Ця дія виконується простою підстановкою шаблонного коду початку програми C/C++.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Інструментальна підтримка під час реінжинірингу та рефакторингу

Оскільки ітераційний процес тестування передбачає багаторазове повторення тестів, ручне тестування стає неефективним і не дозволяє ретельно оцінити якість програмного продукту. Особливо це стосується навантажувального й стресового тестування, де потрібно моделювати робоче навантаження й накопичується значний обсяг даних. Вихід складається в застосуванні інструментальних засобів, що підтримують автоматизацію складання й виконання тестів.

Процеси підтримки

RUP передбачає три процеси підтримки:

– Керування проектом.

– Керування конфігурацією.

– Керування середовищем.

Керування проектом. Цілі

Основною метою процесу керування проектом є забезпечення керівництва проектом, спрямоване на успішну здачу програмного продукту. RUP акцентує увагу на плануванні життєвого циклу й окремих ітерацій, керуванні ризиками, наблюдаємості ходу проекту й метриках проекту.

Планування припускає створень двох видів планів:

– План фаз – великомасштабний план проекту, що показує основні віхи життєвого циклу (дати завершення великих етапів – фаз, випуску еволюційних прототипів і т.д.) і необхідні ресурси. Він створюється на початку фази дослідження й обновляється в міру необхідності.

– План ітерацій створюється для кожної ітерації й призначається для визначення й розподілу завдань між учасниками проекту.

Ризиком будемо називати все те, що може стояти на шляху до успіху проекту й на даний момент є невідомим або невизначеним. Головна ідея керування ризиками полягає в тому, що не потрібно пасивно чекати, поки ризик стане проблемою, але потрібно завчасно визначати лінію поводження. Керування ризиками означає визначення й оцінку ризиків, прийняття лінії поводження, спрямованої на усунення, зниження ймовірності ризику, а також вибір дій на випадок реалізації ризику.

Для контролю й керування проектом використовуються виміри. Вони проводяться для того щоб установити, наскільки віддалений поточний стан проекту від поставленої цілі, спланувати роботи й визначити, як можна підвищити ефективність процесу.

Керування проектом. Діяльності

Відкриття нового проекту. Ця діяльність виконується тільки в першій ітерації. Здійснюється ініціалізація проекту, визначаються й оцінюються ризики, розробляється бізнес-план. Ціль – переклад проекту в стадію, коли можливе ухвалення рішення про продовження або про відмову від проекту.

Оцінка області дії проекту й ризиків. Метою є перегляд і уточнення можливостей і характеристик проекту, а також пов'язаних з ним ризиків.

Створення плану розробки програмні системи. Створюється план розробки програмні системи, що включає перелік ризиків, плани вимірів, керування ризиками, дозволу проблем, прийняття продукту. Визначається структура й ресурси проекту. Розробляється план фаз проекту.

Планування ітерації. Розробляється план чергової ітерації, уточнюється й коректується бізнес-план і план розробки програмні системи. План ітерації докладно описує, що повинне відбутися за час ітерації. Він визначає виконавців і виконувані ними роботи. При створенні плану ітерації необхідно:

– Зформулювати об'єктивні критерії успіху ітерації. Вони будуть використовуватися при її оцінці.

– Визначити конкретні, вимірні артефакти, які потрібно розробити або змінити, а також виконувані для цього роботи.

– Використовувати типову ітераційну декомпозицію робіт для реальних дій, які повинні бути зроблені.

– Використовувати кошторис при визначенні тривалості й обсягу робіт для кожного виду діяльності, утримуючи всі значення в межах бюджету.

Спостереження й контроль. Ця діяльність включає розподіл робіт і створення графіка робіт, спостереження за станом проекту, обробку виняткових ситуацій, і створення звіту про стан. Виконується обробка запропонованих змін і включення їх у графік робіт поточної або наступної ітерацій, виробляється спостереження за активними ризиками, дається оцінка прогресу і якості проекту. В RUP постійно виконувані оцінки стану проекту є основою для рішення різних

проблем і керування ризиками проекту. Якщо команда розроблювачів виявила проблему, призначається співробітник, відповідальний за її дозвіл, і визначається дата, коли проблема повинна бути дозволена. Хід процесу повинен регулярно контролюватися, а відновлення повинні виконуватися в міру необхідності.

Керування ітерацією. Метою цієї діяльності є одержання достатніх для виконання ітерації ресурсів, поділ необхідних робіт, оцінка результатів ітерації.

Завершення фази. Виконуються роботи, що завершують виконання фази. Даються відповіді на наступні основні питання:

– Чи вирішені всі основні проблеми попередньої ітерації?

– Чи відомо стан всіх основних артефактів?

– Чи розглянуті всі проблеми розгортання?

При задовільному стані проекту видається дозвіл на перехід до наступної фази.

Завершення проекту. Керівник проекту підготовляє проект до завершення. Готується заключна оцінка стану. При успішній здачі проекту замовник одержує програмний продукт у користування. Ресурси, що вивільняють, можуть бути перерозподілені (використані в інших проектах).

Керування конфігурацією й змінами. Цілі

Метою керування конфігурацією й змінами є підтримка цілісності артефактів з урахуванням можливості внесення в них змін відповідно до запитів на зміни. Цей допоміжний процес поширюється на весь життєвий цикл програмного продукту й складається із трьох окремих процесів.

Керування конфігурацією (Configuration management)

Процес керування конфігурацією (КК) пов'язаний з визначенням артефактів, залежностей між ними й конфігурацій, що є несуперечливими безлічами взаємозалежних артефактів. Сюди ж відносяться питання розподілу робітничих середовищ між учасниками проекту з метою запобігання непотрібного дублювання документів. КК декларує, що всі артефакти підкоряються версійному керуванню. У міру розвитку проекту в кожного

артефакту з'являються безліч версій. Всі вони зберігаються в архіві проекту, причому ведеться історія змін, у якій вказується причини й цілі створення кожної версії кожного артефакту. Записану в архів проекту версію не можна змінити, можна лише створити нову версію. Крім того, знання про залежності між артефактами дозволяють точно повторювати дії при створенні наборів артефактів (наприклад, при складанні програми з окремих файлів). Створення версій і зв'язків підтримується за допомогою інструментальних засобів КК.

КК дозволяє:

– виключити можливість втрати артефактів,

– забезпечити можливість «повернення назад» у випадках, коли ухвалені рішення й отримані при їхній реалізації версії артефактів не влаштовують замовника або розроблювачів проекту,

– гарантувати точне повторення дій над групою зв'язаних артефактів в ітераціях,

– уникати дублювання артефактів і пов'язаної із цим можливості привнесення додаткових дефектів,

– підтримувати наблюдаємість ходу виконання робіт із проекту шляхом надання потрібної інформації.

Керування внесенням змін під час реінжинірингу та рефакторингу

Діяльності цього процесу спрямовані на збір і збереження запитів на внесення змін, що поставляються як учасниками розробки програмні системи, так і зовнішніми сторонами. Запити на внесення змін можуть з'являтися по різних причинах (виправлення дефекту, поліпшення параметра якості продукту, наприклад, продуктивності, завдання додаткових вимог і т.д.). Кожний запит на внесення змін зберігається в базі даних проекту й може перебувати в різних станах залежно від ходу виконання робіт із цього запиту (новий, зареєстрований, прийнятий, завершений і ін.). Стани запитів змінюються учасниками проекту відповідно до виділених прав. У міру роботи із запитом у нього додається інформація (наприклад, спочатку це може бути тільки опис дефекту, потім

додається результати аналізу, вказуються артефакти, що зачіпаються, призначається виконавець). Із запитом може бути зв'язаний пріоритет, що дозволяє визначити порядок виконання робіт із запитів на зміни.

Керування станом і вимірами

Цей процес пов'язаний з керуванням проектом і спрямований на надання інформації, корисної для оцінки:

– Стану, прогресу, загальних тенденцій і якості продукту.

– Витрат і витрат.

– Проблемних областей, що вимагають уваги.

– Того, що зроблено й що необхідно зробити.

Вся необхідна інформація перебуває в базі даних і архіві проекту. Керівник робіт оцінює стан справ шляхом безпосереднього перегляду запитів, історії версій артефактів або на основі аналізу різних звітів, формованих інструментальним засобом КК. Аналіз дозволяє керівникові оцінити наявні ресурси й прийняти необхідні управлінські рішення.

Діяльності

Планування конфігурації проекту й керування змінами. Описуються всі види діяльності, пов'язані із КК, які треба виконати. Описується процес контролю над змінами.

Створення середовища КК. Метою цієї діяльності є створення робітничого середовища, у якій будуть доступні всі артефакти, буде забезпечена розробка, інтеграція й архівування продукту.

Створення робітничого середовища виконавців. У межах свого робітничого середовища виконавець одержує доступ до артефактів проекту, має можливість вносити в них зміни й пропонувати внесення змін у проект у цілому. Зміни, зроблені окремими виконавцями, направляються в робітниче середовище інтеграції.

Керування версіями. При записі артефакту в архів проекту формується його нова версія. Керування версіями передбачає обов'язкову вказівку причин і

цілей створення версії. При виборі артефакту з архіву можна вказати конкретну версію.

Керування запитами на внесення змін. Метою цієї діяльності є забезпечення послідовного внесення змін у проект і інформування про стан продукту, його змінах і про вплив змін на вартість і строки виконання проекту.

Спостереження за станом конфігурації. Наблюдаємість процесу розробки забезпечується шляхом доступу керівництва проекту до збережених артефактів і запитів на зміни, а також шляхом надання звітів про стан конфігурації. При цьому інструментальні засоби, що підтримують керування конфігурацією й змінами, можуть виконувати необхідні виміри. Наприклад, можна показати відсоток завершених запитів, число відкритих запитів високого пріоритету й т.д.

Керування середовищем. Цілі

Багато видів діяльностей, передбачені RUP, можуть бути автоматизовані за допомогою інструментальних засобів, що дозволить уникнути найбільш трудомістким, напруженим і підданих помилкам аспектів розробки програмні системи.

Ціль процесу керування середовищем – процедурна й інструментальна підтримка організації, що розробляє програмні системи. Вона включає:

– Вибір і придбання інструментальних засобів.

– Настроювання інструментальних засобів під вимоги організації-розроблювача.

– Конфігурування процесу.

– Удосконалення процесу.

– Створення технічних служб підтримки.

Ролі й артефакти

Основним виконавцем процесу є технолог. У його обов'язку входить конфігурування процесу перед запуском проекту й удосконалення процесу в ході проектних робіт. Підтримка апаратного й програмного середовища розробки лягає на плечі системного адміністратора.

Головним створюваним артефактом є план розробки, що описує процес, використовуваний у даному проекті. У ньому вказується, які артефакти, що передбачаються в RUP, будуть використовуватися в проекті і якому образі.

Діяльності

Підготовка середовища до реалізації проекту. Виконується оцінка поточного стану організації-розроблювача й наявної інструментальної підтримки. Створюється попередній план розробки. Складається перелік інструментальних засобів, які передбачається використовувати при розробці. Складається перелік шаблонів для виробництва основних артефактів.

Підготовка середовища до ітерації. Виробляється доповнення й уточнення плану розробки, виконується підготовка й настроювання інструментальних засобів, які будуть використовуватися в ітерації. Складається набір шаблонів для артефактів, створюваних в ітерації. Здійснюється підготовка співробітників до використання інструментальних засобів.

Підготовка керівних директив. Для кожного основного технологічного процесу підготовляється набір директив на основі аналізу проблем і дефектів попередніх ітерацій.

Після початку обробки коду C/C++ проходить перевірка коду на відповідність (докладніше розглянуто у блок-схемах). Далі у лексичний аналізатор подається код та проводиться запуск лексичного аналізатора, обробка та одержання лексем, створення послідовності із всіх лексем коду та одержання машинно-незалежного проміжного коду. Далі відбувається запуск синтаксичного аналізатора котрий обробляє одержані фрагменти об'єктної моделі та створює послідовності із всіх фрагментів об'єктної моделі. Після проведення цих дій генератор коду проводить створення та виведення коду на екран.

Лексичний аналізатор виконує розпізнавання лексем мови і заміну їхніми відповідними кодами.Під лексемами розуміються елементарні одиниці, що входять у структуру пропозиції мови, такі як ключові слова, константи, імена і т.п. Правильність завдання структури речення мови на фазі лексичного аналізу

не виконується. Синтаксичний аналізатор виконує перевірку правильності завдання пропозицій мови відповідно до граматики мови. Тільки при відсутності помилок можлива робота наступних модулів компілятора.

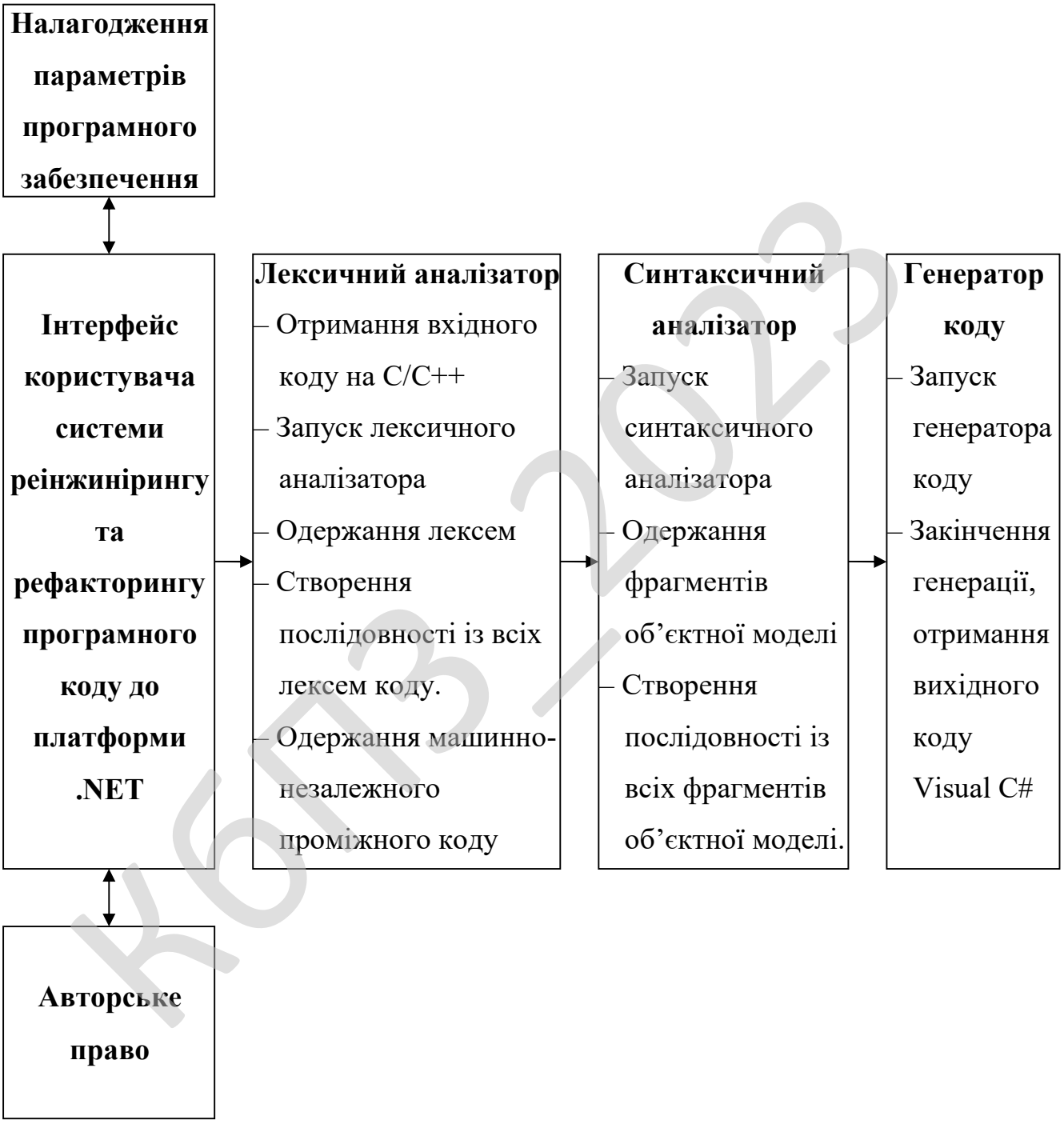

Рисунок 3.2 – Функціональна схема системи

На фазі трансляції в проміжну форму виконується перетворення програми в матрицю проміжного кодові, де кожен рядок матриці містить у загальному випадку трійку – код операції і два операнда. Проміжні коди не мають прямих аналогів у системі машинних команд, тому дану форму представлення програми називають машинно-незалежною.

Фаза оптимізації призначена для зменшення надмірності програми по витратах годин. У залежності від критеріїв проектування транслятора дана фаза обробки програми може виключатися з циклові обробки програми.

На фазі розподілу пам'яті визначаються обсяги пам'яті, використовуваних у даній програмі, і вносяться коди резервування пам'яті в матрицю проміжного кодові. Кожна програма використовує принаймні два види пам'яті:

– статичну – для розщеплення даних і кодів програми;

– стекову – для звертання до підпрограм.

На фазі генерації кодові виконується підстановка кодових зразків вихідною мовою, що відповідають проміжним кодам програми. Крім того, на цій фазі обробки програми виконується також машинно-залежна оптимізація програми, що полягає в обліку особливостей архітектури даного комп'ютера. Використання загальних регістрів процесора для збереження результатів проміжних обчислень дозволяє майже в два рази зменшити годино виконання програми.

Інтерфейс WINDOWS взаємодіє з інтерфейсом розробленого магістерського програмного забезпечення котрий в свою чергу розгалуджується на налагодження параметрів ПЗ, авторське право, лексичний аналізатор котрий взаємодіє з синтаксичним аналізатором та генератор коду.

Розроблена програма містить в собі дві основні таблиці – таблиця термінальних символів C/C++ та синтаксис Visual C#.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання магістерського проектування, наведена на рисунку 3.3.

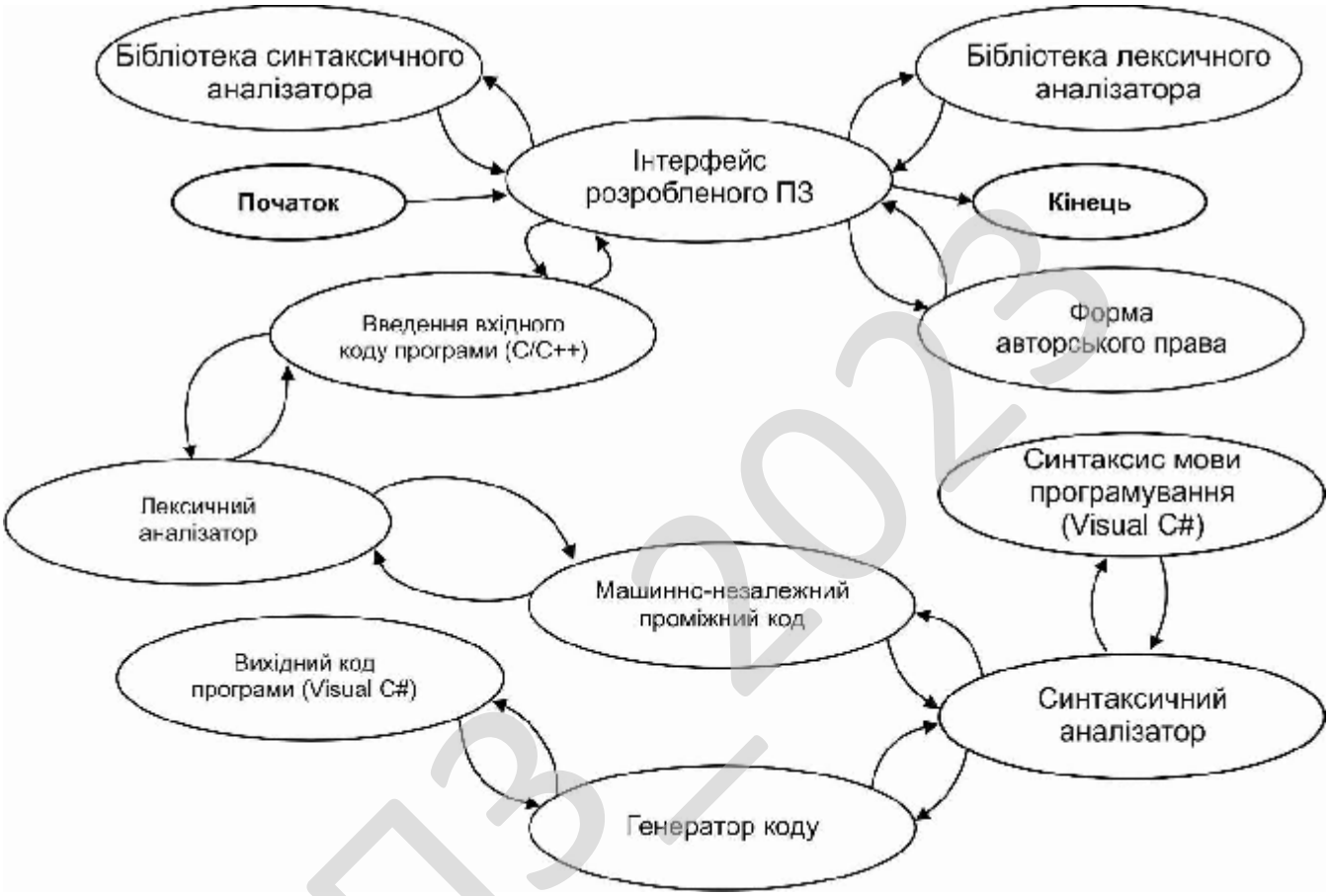

Рисунок 3.3 – Діаграма взаємодії процесів

Спочатку ми потрапляємо до інтерфейсу розробленого ПЗ, звідки проходе взаємодія з блоками бібліотеки лексичного аналізатора, бібліотеки синтаксичного аналізатора та формою авторського права. Далі через введення вхідного коду програми (C/C++), лексичний аналізатор ми отримаємо машинно-незалежний проміжний код. Та через синтаксичний аналізатор з синтаксисом мови програмування (Visual C#) за допомогою генератора коду отримуємо вихідний код програми (Visual C#).

Тобто проводиться трансляція вихідних текстів – трансформація програми з однієї мови програмування (C\C++) або з одного діалекту в іншій (C#).

Відзначимо, що на початку процесу реінжинірингу старої системи проводиться глобальний аналіз і оцінка програми: інвентаризація, планування й оцінка ресурсів, аналіз частин (перевірка на коректність компонентів, наявність недозволених зовнішніх посилань і ін.). У процесі проведення таких дій ставиться діагноз системи, визначаються частини, що вимагають заміни, і виробляється загальна стратегія трансформації. При проведенні реінжинірингу складові етапи проводяться в певному порядку: зворотне проектування, потім пряме (якщо використовуються) і ін., при цьому можливо одночасне виконання деяких процесів, наприклад, реструктуризації й переорієнтації.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок–схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Розглянемо блок-схему крок за кроком, спершу відбувається ініціалізація змінних, та підключення бібліотек, пошук та підключення бібліотеки лексичного аналізатора, пошук та підключення бібліотеки синтаксичного аналізатора, підключення лексем мови програмування C#, підключення синтаксису мови програмування C++ та виведення головного вікна ПЗ на екран. Після чого проходе очікування дій користувача, якщо цих дій немає та є код Exit якій дорівнює одиниці проходе завершення роботи ПЗ.

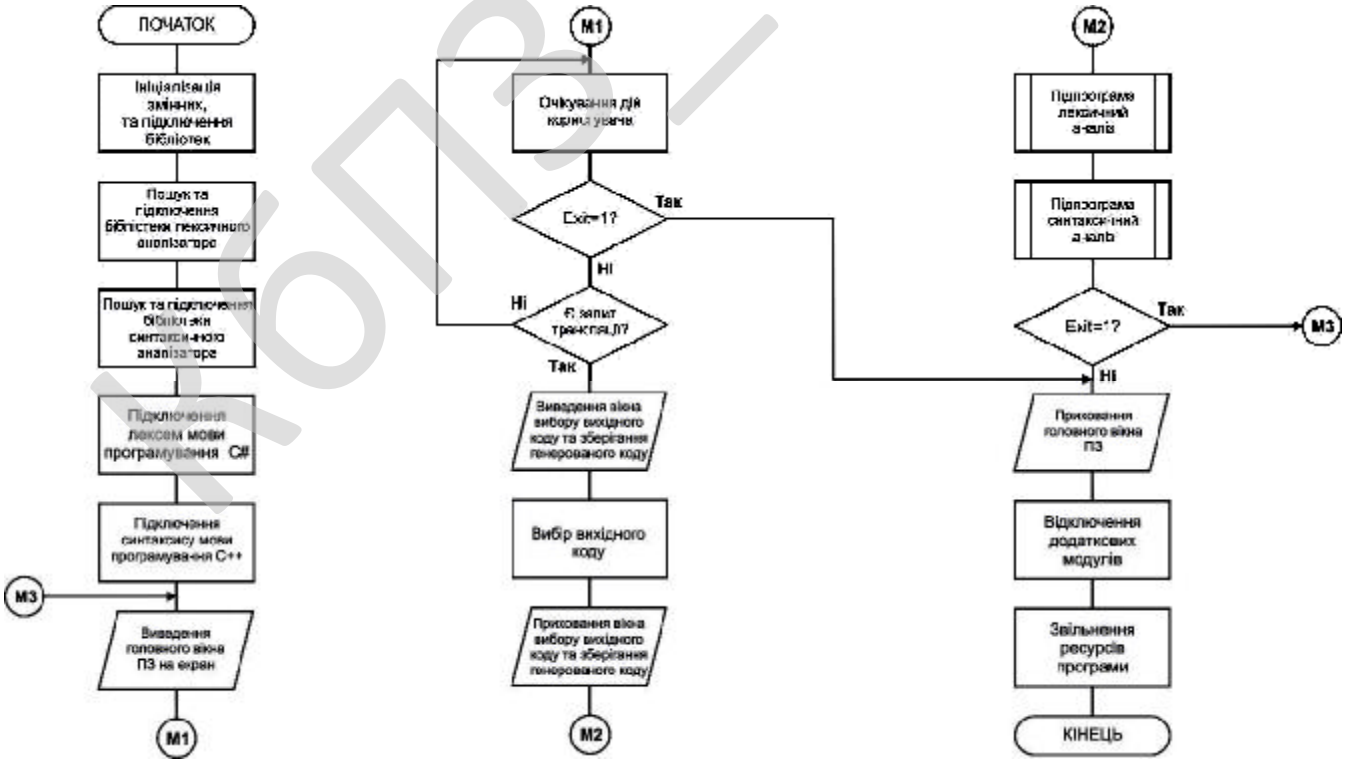

Рисунок 4.1 – Блок-схема основної програми

Якщо дії є, проходе перевірка запит це трансляції. Та якщо так проводиться виведення вікна вибору вихідного коду та зберігання генерованого коду, вибір вихідного коду, приховання вікна вибору вихідного коду та зберігання генерованого коду.

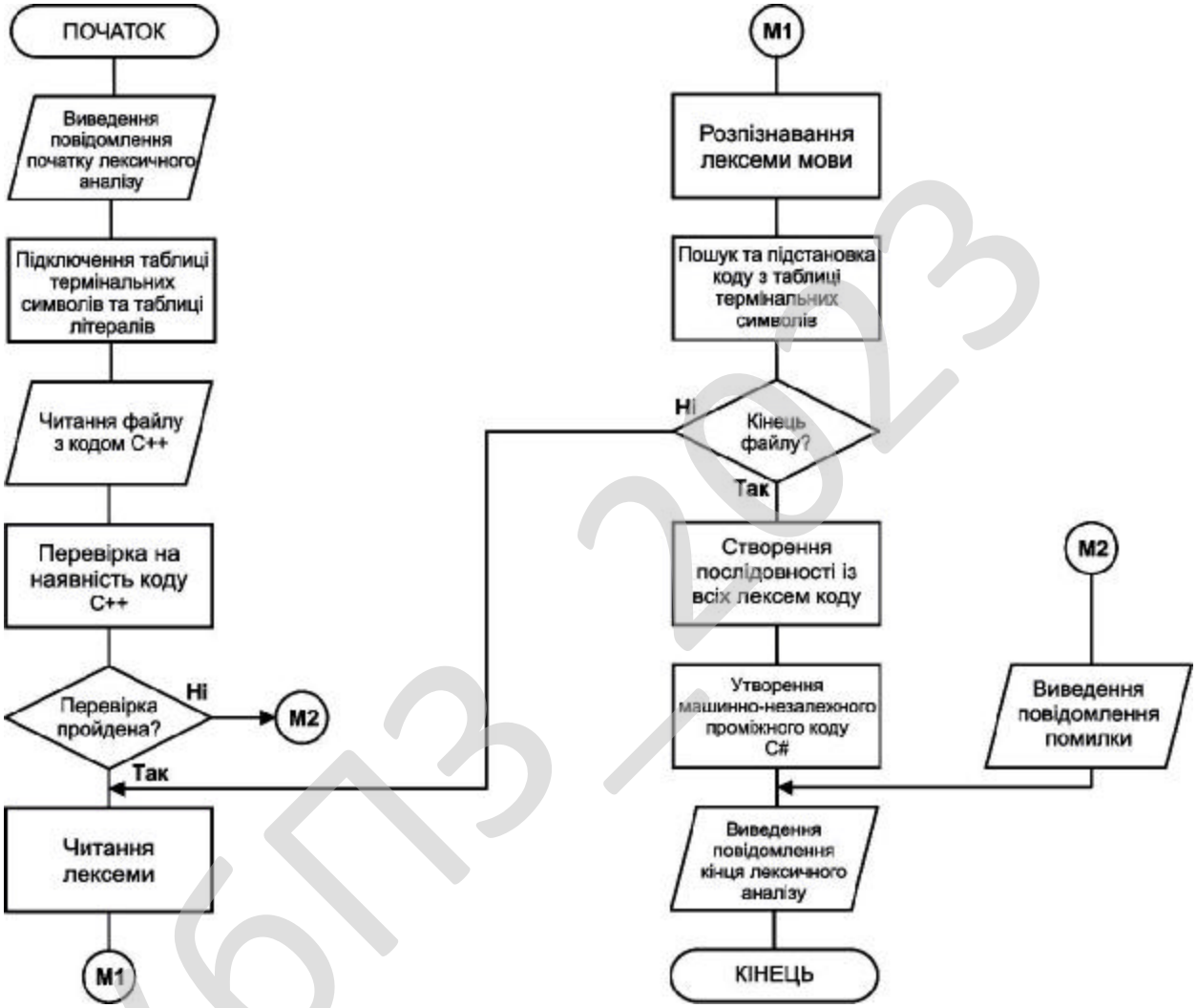

Рисунок 4.2 – Блок-схема роботи підпрограми лексичний аналіз

Проводиться виклик підпрограми лексичний аналіз який зображено на рисунку 4.2. Там проводиться виведення повідомлення початку лексичного аналізу, підключення таблиці термінальних символів та таблиці літералів, читання файлу з кодом C++, перевірка на наявність коду C++. Якщо перевірка пройдена проводиться читання лексеми, розпізнавання лексеми мови, пошук та

підстановка коду з таблиці термінальних символів. Якщо всі лексеми розпізнано (Кінець файлу), проводиться створення послідовності із всіх лексем коду, утворення машинно-незалежного проміжного коду C#, виведення повідомлення кінця лексичного аналізу. Та повернення до головної блок-схеми з викликом підпрограми синтаксичний аналіз – рисунок 4.3.

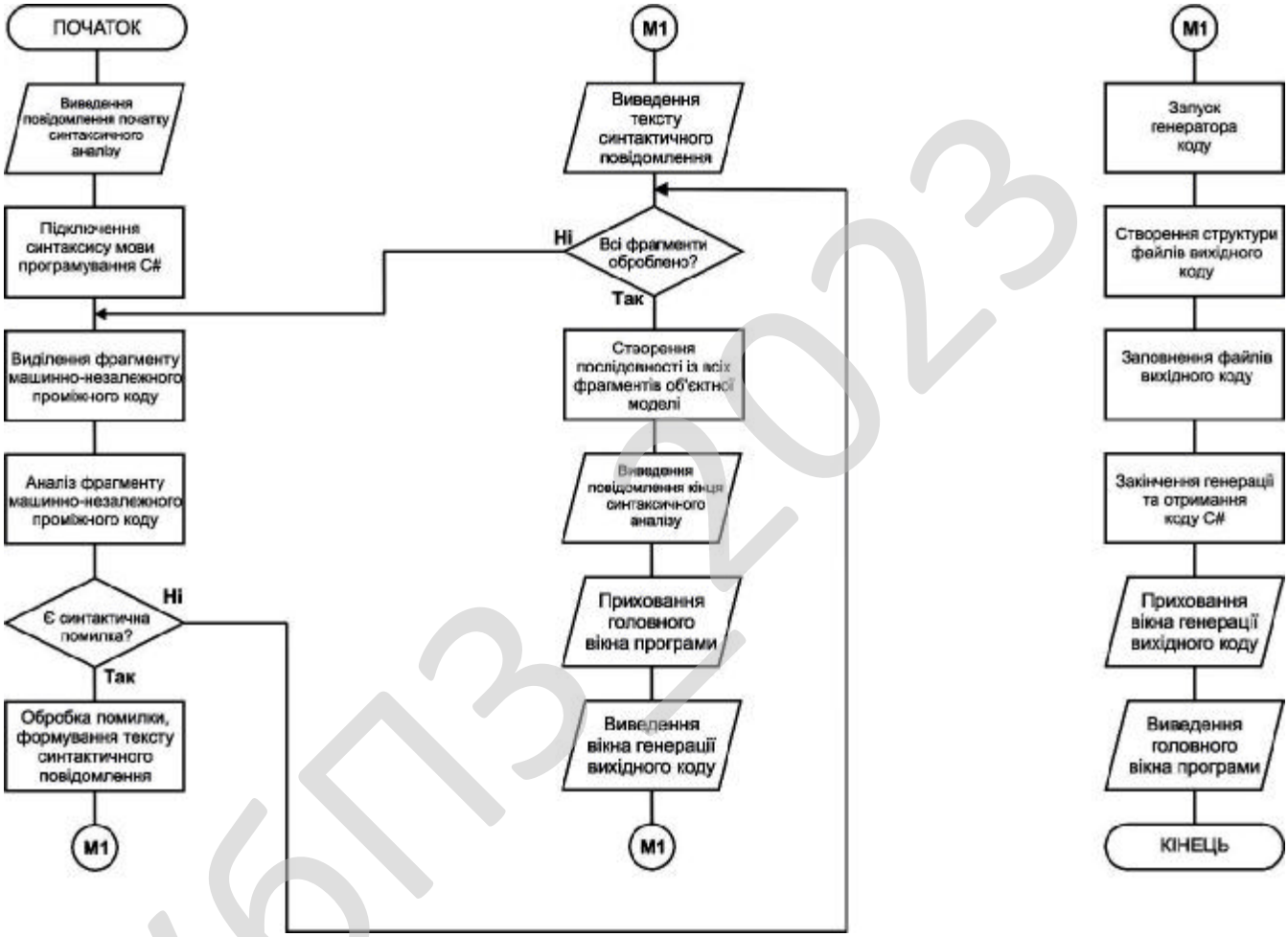

Рисунок 4.3 – Блок-схема підпрограми синтаксичного аналізатора

У синтаксичному аналізі відбувається виведення повідомлення початку синтаксичного аналізу, підключення синтаксису мови програмування C#, виділення фрагменту машинно-незалежного проміжного коду, аналіз фрагменту машинно-незалежного проміжного коду та перевірка синтактичної помилки. Якщо помилка є проводиться обробка помилки, формування тексту

синтактичного повідомлення та виведення тексту синтактичного повідомлення. Так доки всі фрагменти не буде оброблено. Далі проводиться створення послідовності із всіх фрагментів об'єктної моделі, виведення повідомлення кінця синтаксичного аналізу, приховання головного вікна програми, виведення вікна генерації вихідного коду, запуск генератора коду, створення структури файлів вихідного коду, заповнення файлів вихідного коду, закінчення генерації та отримання коду C#, приховання вікна генерації вихідного коду та виведення головного вікна програми яке зображено на рисунку 4.1.

Якщо поступає сигнал Exit=1 проводиться приховання головного вікна ПЗ, відключення додаткових модулів, звільнення ресурсів програми.

Розглянемо реалізацію лексичного аналізатора.

Лексичний аналіз – це процес сканування потоку вхідних символів і поділу його на рядки, які називаються лексеми. Більшість книг по компіляторам починаються з цього і присвячують кілька глав обговоренню різних методів побудови сканерів. Такий підхід має своє місце, але існують безліч речей, які можна зробити навіть ніколи не зверталися до цього питання.

Теорія компіляторів і, отже, програми, наступні з неї, повинні працювати з більшістю загальних правил синтаксичного аналізу.

Як правило, лексичний аналізатор створюється як окрема частина компілятора, так що синтаксичний аналізатор по суті бачить тільки потік вхідних лексем. Теоретично немає необхідності відокремлювати цю функцію від іншої частини синтаксичного аналізатора. Є тільки один набір синтаксичних рівнянь, який визначає весь мову, тому теоретично можна б написати весь аналізатор в одному модулі.

Лексичний аналізатор це самостійна програма (або її фрагмент), реалізує функцію лексичного аналізу.

Лексичний аналіз це розкладання і перетворення вхідного тексту на певні послідовності символів (не обов'язково текстового подання), іменовані токенами. Кожен токен – це послідовність символів (не обов'язково текстових), однозначно

характеризують лексему.

Лексема це послідовність символів і слів, що однозначно визначають яку-небудь складову мови.

Реалізація.

Стандартна методика роботи наступна: береться суцільний потік символів, розбивається на токени, аналізується, і (в залежності від типу транслятора) виконуються наступні дії – компіляція або виконання.

Але, як це зазвичай буває, сучасні технології пішли далеко вперед, а методи підносяться ті ж самі. Розглянемо лексичний аналізатор, отриманий трохи іншим способом.

Спочатку виробимо якесь безліч абстрактних мов, з якими буде працювати наш лексичний аналізатор. Це важливо, тому що, в кінцевому рахунку, всі лексичні аналізатори мають обмеження і здатні працювати лише під контролем жорстких заборон, накладених численними правилами (граматикою) мови. Отже, наша мова програмування C #, з метою спрощення викладу, передбачає побудову по строковому принципом, іншими словами, один рядок може містити лише одну команду.

Дійсно, працювати з потоком символів незручно (але швидше), в той же час є велика кількість класів, орієнтованих на роботу з наборами рядків. Крім того, наявність декількох команд в одному рядку було викликано історичними причинами: ранні редактори були екраноорієнтіроваммими, і велика кількість рядків було недозволеною розкішшю. У той же час, ми не обмежені зобов'язаннями сумісності з попередніми версіями нашої мови. Далі введемо такий термін, як конструкція.

Конструкція це символьне представлення команди мови програмування C#. Здавалося б, навіщо таке ускладнення концепції, але ми ж не збираємося відставати від сучасних технологій і будемо рівнятися на. Net. Тут для нас найважливішою рисою. Net є різноманіття мов програмування, що виконують по суті одну і ту ж роботу (можливо, з різних точок зору, – зараз це не принципово).

Хоч C#, хоч Delphi з приставкою. Net – всі вони різні зовні, але побудовані на одній платформі, і це відчувається, незважаючи на різний синтаксис. Саме з метою легкої зміни синтаксису й вводиться термін «конструкція». Отже, загальний вид конструкції мови програмування у нас наступний:

```
using System;
namespace Example
{
      class Program
      {
            static void Main ()
            {
                  Console.WriteLine ("Hello World!"); // Виведення
                  заданого тексту в консоль
                  Console.ReadKey (); // Очікування натискання клавіші
                  користувачем
            }
      }
}
```

Таким чином, в конструкції може бути не більше восьми лексем. Тут слід зробити застереження: роздільники лексемами не є (в інших мовах це може бути не так). Практика показує, що восьми лексем вистачає для кодування переважної більшості команд мови.

Більша кількість лексем надлишкова і веде до утруднення сприйняття (але, природно, після засвоєння принципів побудови лексичних аналізаторів кожен може підібрати оптимальну кількість лексем самостійно) і зменшує швидкість роботи аналізатора. Мої лексеми представлені або частиною команди, або командою, або виразами декількох видів (останнє не обмовляється: розбір виразів – це інша велика тема). Так що всі згадки змінних і чисел відносяться до виразів. Сама команда може розглядатися як комбінація службових слів (в деяких джерелах літератури їх називають ключовими) і параметрів.

І до того ж містить п'ять лексем. Тобто треба враховувати той факт, що команда не обов'язково займає всі вісім лексем, а може і менше (але не більше, інакше наш лексичний аналізатор буде непридатний).

Він відповідає не за конкретну лексему, а за всю команду. Це цілком природно: якщо в рядку може бути тільки одна команда, то її можна умовно позначити певним числом.

Немає сенсу на кожен шматок конструкції заводити своє число. Достатньо пам'ятати саму команду (у вигляді числа) і її параметри. Тепер поле Remark – це просто опис команди, яке можна, наприклад, виводити в спливаючій підказці при наведенні на команду (що корисно починаючим), в скриптах або в мовах програмування, орієнтованих на предметні області (математика, фізика, бухгалтерія і т.д.). Це опис однієї конструкції мови. Відповідно, щоб описати весь мову, потрібен масив таких ось записів.

Це нескладно, але є нюанси: в опис команди потрібно запхнути параметри, а також врахувати рядкові константи (якщо вони будуть в мові). Перше вирішується просто і універсально, місце, де повинен знаходитися параметр, будемо позначати цифрою від 1 до 8 (навіщо – поясню трохи нижче). А ось зі рядковими константами треба порозумітися. Справа в тому, що в них може міститися фрагмент тексту, зовнішнім виглядом сильно нагадує конструкцію.

Тому, щоб лексичний аналізатор був максимально незалежний від решти реалізації програми, нам потрібно знати як мінімум ознака строковою константи (для C # це текст між апострофами).

Тобто якщо, наприклад, зустрілася рядок "for ...", лексичний аналізатор повинен відразу визначити, що перед ним не операція, що позначає цикл, а щось зовсім інше (а саме for параметр 1, ...).

Тепер розглянемо кодування параметрів. Для початку покажемо, як ми будемо представляти параметри всередині лексичного аналізатора.

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
      public partial class Form1: Form
      {
            public Form1 ()
            {
```

```
                InitializeComponent ();
            }
            private void button1_Click (object sender, EventArgs e)
            // Елементи label та button додані попередньо
            {
                label 1. Text = "Hello, World!";
            }
        }
    }
```

Тут все просто, тому що, по суті, лексичному аналізатору параметри конструкції абсолютно не потрібні – вони потрібні далі, при трансляції коду. Ми не будемо обробляти дані параметри (в переважній більшості випадків це математичні вирази), а просто надамо можливість їх отримання з лексичного аналізатора. Тепер повернемося до цифр. Вони позначають відповідний індекс від 1 до 8. Це потрібно для додання універсальності лексичному аналізатору: як уже було сказано вище, синтаксис для нас – річ умовна, і ми будемо будувати код таким чином, щоб мати можливість змінити синтаксис будь-якої конструкції (тобто ключовим полем для лексичного аналізатора є ідентифікатор конструкції).

В результаті ми отримуємо наступне:

– можна створювати різні конструкції, що виконують одні й ті ж функції;

– можна міняти порядок проходження параметрів в деяких межах без перебудови транслятора;

– оскільки опис всіх конструкцій є записи, що складаються з простих полів, ми можемо їх зберігати і зчитувати з зовнішніх джерел – файлів, отримувати по мережі, генерувати під час роботи і т.д., і все це прямо час роботи транслятора;

– маючи чітке і просте внутрішнє уявлення, ми можемо готувати різні звіти по своїй мові, наприклад, автоматично генерувати форму Бекуса-Наура (форма опису синтаксису мови програмування);

– при певній вправності ми можемо описувати синтаксис, вираз якого звичайними засобами важко, або неможливо.

Однак на цьому корисності нашого лексичного аналізатора не закінчуються. Багато мов програмування вводять так званий синтаксичний цукор- різні подання одного і того ж механізму. Ефекту від них небагато, але це просто зручно для сприйняття людиною. Зокрема, наші подвійні конструкції – на латиниці і на кирилиці – вже сам по собі синтаксичний цукор, оскільки нових можливостей, що дозволяють ефективніше вирішувати завдання, програміст не отримав.

Кожна конструкція є відповідна строчка в масиві конструкцій, які перебираються до тих пір, поки не буде знайдено збіг рядки зі зразком.

Процес розбору рядка заняття трудомістке, і проводиться він кожен раз заново, поки не буде знайдений потрібний елемент або поки весь масив не буде переглянутий до кінця (в такому випадку повертається спеціальний ідентифікатор – невідома конструкція). Змінити ж такий порядок (в сенсі розкладання рядки на складові) не представляється можливим: потрібно кардинальна перебудова алгоритму, і не факт, що новий варіант буде швидше (а для нас чималу значимість представляє і простота внутрішнього устрою).

Особливості реалізації.

Лексичний аналізатор представлений у вигляді класу, що дає магістерській програмі, велику гнучкість:

```
// Реалізація лексичного аналізатора
type
TLA = class
  protected
Nabor: Array of TKonstr; // Набір конструкцій
Constr: TConstrParam; // Тимчасове місце для зберігання параметрів
// Отримання фрагмента рядка до першого символу табуляції
       function GetTab (Stroka, Symb: String): String;
// Отримання рядки без першого входження символу табуляції
        function EraseTab (Stroka, symb: String): String;
        // Перевірка збігу рядка з шаблоном конструкції
        function CompareStrId (Stroka: String; Index: Integer):
                                Boolean;
// Перевірка: входить слово в указану безліч слів
// розділених пробілами
```

```
function CheckInSet (Template, Value: String): Boolean;
// Перевірка: чи є рядок числом
function CheckNumberStroka (Stroka: String): Boolean;
// Отримання числа слів в рядку
function GetIndex (Stroka1, razdelitel: String): Integer;
// Відкидає з рядка елемент, поки не знайдеться шуканий елемент
        function GetEnd (Stroka, Element: String): String;
        // Відкидає елементи, укладені в зазначений символ
        function DelAp (Stroka, Symb: String): String;
        // Видалення елемента строкового масиву за індексом
        Function DelItemMas (Stroka1, razdelitel: String;
        index:Integer):String;
// Повертає частину рядка, яка знаходиться після зазначеного підрядка
        Function After (Src: String; S: String): String;
        // Отримання елемента строкового масиву за індексом
        function GetItemMas (Stroka1, razdelitel: String;
         Index:Integer): String;
private
public
constructor Create;
procedure Init (); // Ініціалізація класу
        destructor Destroy; override;
// Додавання нової конструкції (у вигляді рядка)
procedure AddConstr (Constr: String);
// Отримання ідентифікатора і параметрів конструкції
      procedure GetID (Stroka: String; var Id: Integer; var Param:
      TConstrParam);
// Читання набору конструкцій з списку
procedure ReadNabor (var Spisok: TStringList);
        // Запис набору конструкцій в список
procedure GetNabor (var Spisok: TStringList);
// Читання конструкції у вигляді рядка
        function GetConstr (Index: Integer): String;
        // Визначення числа конструкцій в наборі
function GetCount ():Integer;
end;
```

Користування готовим лексичним аналізатором ніяких турбот не доставляє, тим більше що він добре документований у вигляді коментарів до коду. Додавання конструкцій в їх набір (по суті, опис мови) здійснюється у вигляді рядка, де кожне поле відділене символом табуляції. Зроблено це також

для простоти і для подальшої можливості скидання конструкцій в текстовий файл CSV формату (поля розділяються певним символом).

Сам клас аналізатора зберігати і читати в файл не вміє (для додання універсальності), але вміє це робити в список рядків TStringList, де список конструкцій можна обробити або просто зберегти безпосередньо. Читати з набору конструкції можна також у вигляді рядків.

Найголовніша і відповідальна функція тут GetID, яка в циклі порівнює рядок з конструкціями допомогою самої хитрою і цікавою функції CompareStrId.

Остання зовсім не проста, вона розкладає рядок з урахуванням багатьох параметрів: кількість лексем у конструкції, порядок проходження параметрів, наявність в рядку рядкових констант і т.д. Також в секції protected міститься ряд досить-потужних допоміжних функцій по обробці рядків, які обов'язково знадобляться для обробки рядків.

У лексемах заборонені символи табуляції (оскільки не можна правильно зберегти і прочитати дану конструкцію в подальшому), символи пробілу (вони є роздільниками службових слів усередині лексем). Будь-які самостійні цілі числа (так як вони позначають місце розташування параметрів, але дозволені, наприклад, шістнадцяткові) і символ апострофа, як ознака текстової константи (в даний момент це жорстко прописано в лексичному аналізаторі, але можна виправити в функції GetEnd).

Зате все інше, включаючи недруковані символи, цілком припустимо. Це дозволяє використовувати такі конструкції, використання яких при традиційних лексичних аналізаторах неможливо: багаторазове використання точкової нотації (як для визначення структур даних, так і для визначення конструкцій), скобочної нотації і т.д. Можливо це тому, що лексичний аналізатор не виділяє токени лексем, зокрема, роздільників (дужки, крапки, коми та інші спецсмволи).

При правильній розстановці конструкцій в наборі допускається використання і арифметичних знаків у складі лексем конструкцій. Все це веде до більш широкого використання контекстно-залежних граматик (для довідки: на

даний момент більшість мов програмування підпорядковується безлічі контекстно-залежних граматик – контекстно вільні граматики), що дозволяє писати програми на мовах, більш наближених до природних (це призводить до зменшення числа логічних помилок в програмах). З метою спрощення роботи з лексичним аналізатором лексеми (але не параметри) порівнюються з рядком без урахування регістра.

Розглянемо структури даних котрі використовувалися у трансляторі опишемо їх в вигляді роз'яснення.

Рядок – послідовність символів , що належати кінцевому безлічі, або алфавітові. Наприклад , якщо A – безліч{0,1} , те послідовність 01011101101110 є рядок над A. Для відділення рядків друг від друга застосовуються або спец символи, або змінюється алфавіт.

Властивості:

– довжина рядків перемінна;

– звертання до елементів рядка відбувається з якого-небудь кінця(тобто

– важлива упорядкованість послідовності , а не її індексація);

– якщо рядок представляє кінцеву пропозицію деякої мови, то існують обмеження, які символи можуть зустрічатися разом , що задається синтаксисом мови.

Орієнтований граф – формально визначається як G=(X,U) ,де X – безліч елементів, називаних вершинами; U – безліч дуг або ребер графа: U X*X. Якщо a і b вершини , a (a,b) – ребро.

Неорієнтований граф – у якому відсутні орієнтовані ребра: (a,b)U, ті і (b,a) U. Якщо на додаток до графові задана функція w:U R , ті це зважений граф . Функція w визначає довжину або ваги шкірного ребра в графі.

Дерево – кореневе дерево(орієнтоване дерево, дерево) – орієнтований граф, у якому – усі вершини, крім однієї (кореня), знаходяться в голові тільки одний дуги; корінь дерева не знаходиться в голові якої-небудь дуги; корінь зв'язаний з кожною вершиною дерева.

Обхід дерева – це упорядкована послідовність вершин дерева, у якому кожна вершина зустрічається тільки один раз. У кожну вершину попадаємо принаймні один раз, а взагалі говорячи 3 рази. При обході дерева можуть виконуватися різні дії.

Стеки і черги – динамічні структури, приспособлені для того, щоб додавати елементи в безлічі і видаляти їх. Стекова пам'ять, організована за принципом LIFO ("Last In First Out"). Стек – обмежен з однієї сторони послідовністю перемінної довжини.

Список – послідовність елементів, де кожен елемент містить інформацію двох типів: внутрішню-властиву даному елементові, і зовнішню-про зв'язок даного вузла з іншими елементами спискові. Зовнішня інформація про зв'язок елемента має вигляд покажчиків, тобто адреса вузлів, зв'язаних з даними.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Lucifer. Алгоритм Lucifer являє собою мережу перестановок і підстановок, його основні блоки нагадують блоки алгоритму DES. В DES результат функції f складається операцією XOR із входом попереднього раунду, утворюючи вхід наступного раунду. В S-блоках алгоритму Lucifer 4-бітові входи й виходи, вхід S-блоків являє собою перетасований вихід S-блоків попереднього раунду, входом S-блоків першого раунду служить відкритий текст. Для вибору використовуваного S-блоку із двох можливих використовується біт ключа. (Lucifer реалізує все це в єдиному T-блоці з 9 бітами на вході й 8 бітами на виході). На відміну від алгоритму DES, половини блоку між раундами не переставляються, та й саме поняття половини блоку в алгоритмі Lucifer не використовується. У цього алгоритму 16 раундів, 128-бітові блоки й більше проста, чим в DES, схема розгорнення ключа.

Блок тексту розглядається як ненегативне ціле число, або як кілька незалежних ненегативних цілих чисел. Довжина блоку завжди вибирається рівною ступеню двійки. У алгоритмі Lucifer використовуються наступні типи операцій:

– Таблична підстановка, при якій група біт відображається в іншу групу біт. Це так звані S-box.

– Переміщення, за допомогою якого біти повідомлення переупорядковуються.

– Операція додавання по модулю 2, позначувана XOR або ⊕.

– Операція додавання по модулю 2^{32} або по модулю 2^{16}.

– Циклічне зрушення на деяке число біт.

Ці операції циклічно повторюються в алгоритмі, створюючи так звані раунди. Входом кожного раунду є вихід попереднього раунду й ключ, що отриманий по певному алгоритму із ключа шифрування K. Ключ раунду називається підключем. Алгоритм шифрування може бути представлений у такий спосіб:

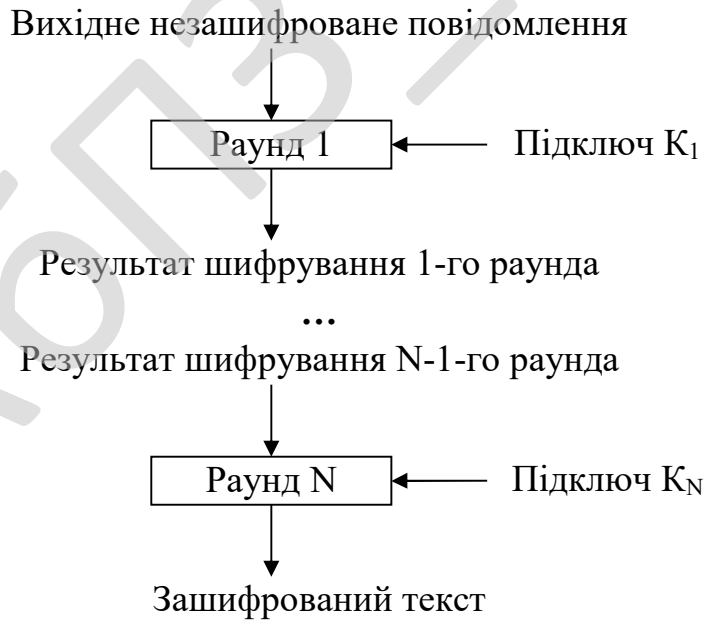

Рисунок 4.4 – Структура алгоритму алгоритмі Lucifer

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ, яке зображено на рисунку 5.1. На рисунку зображено інтерфейс головного вікна, якій розподіляється на наступні частини:

1. Меню. Складається з розділів:

– Файл – операції над файлами вихідного та вхідного коду;

– Налаштування – налаштування інтерфейсу ПЗ та модулів лексичного та синтаксичного аналізатора;

– Виключення – список лексем при виявленні яких трансляція призупиняється.;

– Дані – налаштування типів даних що використовуються при трансляції коду.;

– Семантичний аналізатор – поточні налаштування та статус аналізатора.;

– Синтаксический аналізатор – поточні налаштування та статус аналізатора.;

– Авторське право – виклик вікна розробника.

2. Розділ введення шляхів файлів. Забезпечує встановлення директорій де зберігається вхідний код та вихідний код. Дає можливість переглянути таблицю перетворення, редагувати синтаксичний та лексичний аналізатор (повторення дії меню), також викликати меню налаштування.

3. Вхідний код C\C++. Поле виведення коду що буде транслюватися. При виведенні проводиться перевірка коду на відповідність (вихідний код це чи ні).

4. Журнал роботи. Виводить всі проведені дії користувачем та програмою з виказанням часу виконання дії. Якщо виникають помилки при трансляції, вони також виводяться через це поле.

5. Кнопки дій. Виконують основний функціонал програми – початок трансляції, зупинка трансляції, пауза та вихід з програми.

6. Вихідний код. В цьому полі відображається результат роботи програми – трансляція на мову C#.

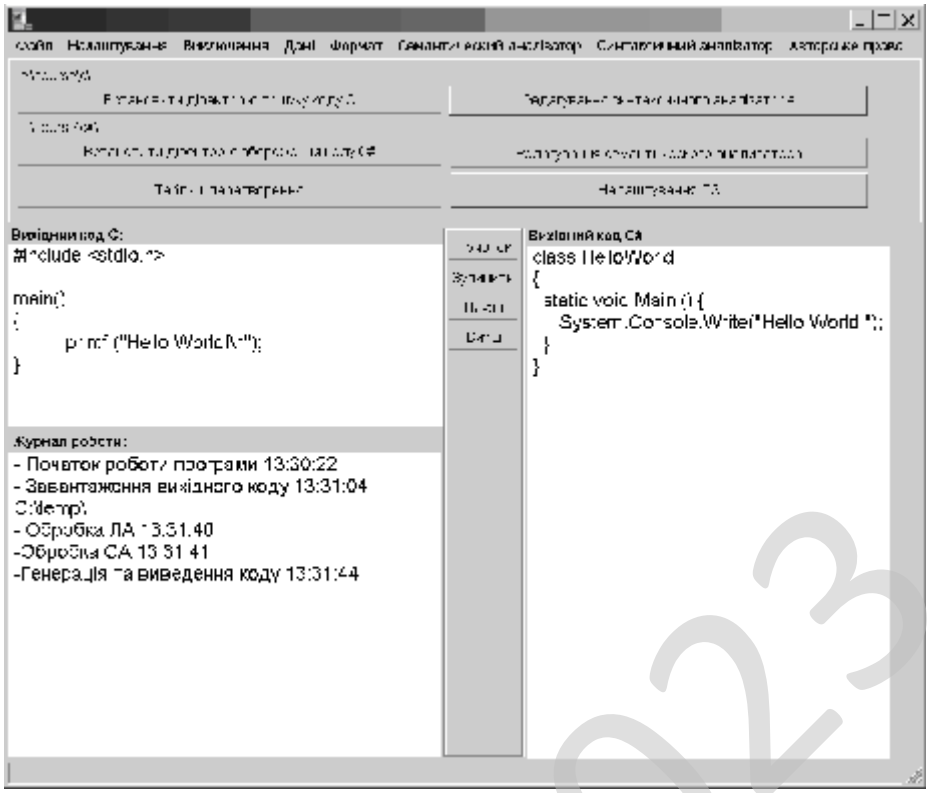

Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено вікно авторського права розробника програми.

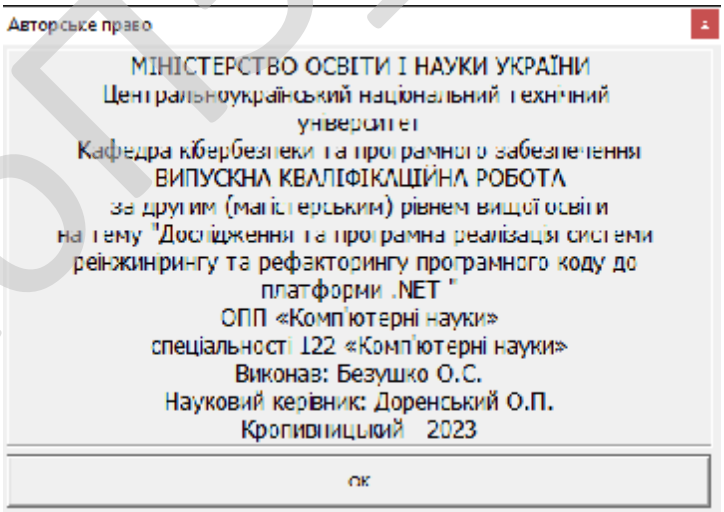

Рисунок 5.2 – Форма авторського права

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Метою розробки є дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Об'єктом дослідження є процес реінжинірингу та рефакторингу програмного коду до платформи .NET.

Предметом дослідження є методи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Методи дослідження базуються на методах інженерії програмного забезпечення, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод реінжинірингу та рефакторингу програмного коду до платформи .NET.

– Розроблено вітчизняний продукт реінжинірингу та рефакторингу програмного коду до платформи .NET, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці). В магістерській роботі проведене дослідження та виконана програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET. Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність

Таблиця 7.1 – Початкові дані

Показники	Позна-чення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	57
3. Запланований термін розробки, днів	Fpq	48 (2 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3

Продовження таблиці 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	_	2
13. Рівень спрацьованості колективу (1-6)	_	2
14. Ступінь вимірності процесів (1-6)	_	3
15. Необхідна надійність програмного забезпечення (1-6)	_	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	_	2
17. Складність кінцевого програмного продукту (1-6)	_	2
18. Необхідний рівень забезпечення повторного використання (1-6)	_	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	_	2
20. Вимоги до швидкодії ПП (1-6)	_	2
21. Обмеження на розміри основного сховища даних (1-6)	_	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	_	2
23. Професійний рівень аналітиків (1-6)	_	2
24. Професійний рівень програмістів (1-6)	_	2
25. Постійність складу команди розробників (1-6)	_	2
26. Досвід розробки додатків (1-6)	_	2
27. Досвід роботи з обчислювальною платформою (1-6)	_	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПО для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн	–	60000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A\ \text{Size}^B \qquad (7.1)$$

де А – коефіцієнт Боема, А=2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням

$$B = 1{,}01 + 0{,}001\sum W_i \tag{7.2}$$

де Wi – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн}=T_{ном}\, ПV_j, \tag{7.3}$$

де $ПV$j – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 6{,}78\cdot(0{,}88\cdot0{,}93\cdot0{,}88\cdot0{,}91\cdot0{,}95\cdot1\cdot1\cdot0{,}87\cdot1{,}22\cdot1{,}16\cdot1{,}1\cdot1{,}1\cdot$$

$$\cdot1{,}12\cdot1{,}1\cdot1\cdot1{,}1\cdot1{,}1)=9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3CT_{уточн}^{0{,}33+0{,}2(B-1{,}01)}S, \tag{7.4}$$

де C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot2{,}66\cdot9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot65=109 \text{ люд/день}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	109	Ф 7.1-7.4
Впровадження	13	Д13
Всього	150	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів,

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні,

$$Ч = \frac{150 \cdot 1}{48\text{-}5} = 3,5 \text{ ставки}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	6	540	9
Монітор	60	6	360	6
Клавіатура	30	6	180	3
Маніпулятор «мишка»	30	6	180	3
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор-маршрутизатор	30	1	30	0,5
Кабельні господарства ЛОМ на 1 м.п.	2,5	140	350	5,83
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	32,99

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2} \quad (7.6)$$

$$Ф_{др}^{с} = \frac{33 \cdot 3}{1,2} = 82,5 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}} \quad (7.7)$$

$Ч_{ел}$=82,5/(48·8)=0,2 ставки

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів–електронщиків. Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп’ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2019, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	0,8	0,2
	Налаштування і конфігурування базової станції безпровідного зв’язку (CMTS)	0,2	
	Розробка та впровадження проектів з організації зв’язку між віддаленими об’єктами, ЛОМ	0,2	
	Забезпечення цілодобової роботи зв’язку клієнтів до мережі Інтернет	0,4	
Всього		1,6	

Продовження таблиці 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	2	0,5
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	0,5	
	Контроль взаєморозрахунків з постачальниками	0,5	
Всього		4	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	0,5	0,2
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,3	
	Розміщення графіки і контенту на Інтернет сторінках	0,3	
Всього		1,6	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,2
	Верстка друкованих видань	0,2	
	Додрукова підготовка макетів	0,2	
	Розміщення графіки і контенту на Інтернет сторінках	0,2	
Всього		1,6	

Складемо штатний розклад виконавців у таблицю 7.5.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (ІТ-менеджер)	1	19103	38206
Продакт-менеджер	0,5	17000	17000
Інженер-програміст	3,5	20000	140000
Інженер-електронщик	0,2	15000	6000
Інженер-системотехнік	0,2	15000	6000
Адміністратор мережі	0,2	11000	4400
Системний програміст	0,2	9000	3600
Дизайнер WEB	0,2	9000	3600
Інженер-верстальник	0,2	9000	3600
Бухгалтер-економіст	0,2	10000	4000
Всього за період розробки	$R_{сп}$=6,4	-	$Ф_{роб}$=226406

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рq}}, \quad (7.8)$$

де $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{226406}{6,4 \cdot 48} = 737 \ грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць.

$S_у$ – питома площа на одне робоче місце, $м^2$,

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7. Дані по оптовій ціні на обладнання та комплектуючі вибирались за комерційною пропозицією фірми Компбест за 15.10.23 – джерело https://compbest.com.ua/

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11771
Системний блок		7771
Процесор	Intel Core i7-3770K 3.9 GHz. 4 ядра/8 потока	-
Системна плата	Huanan B75-PRO (Intel B75, PCI-Ex16) DVI/VGA/HDMI	-
Жорсткий диск	240 HDD + 1 Tb SAMSUNG	-
Оперативна пам’ять	DIMM 8Gb DDR3 PC3-12800 Kingston, 1600MHz, 512M x 64, CL9-9-9-27, 1.65V, w/heatsink, HyperX	-
DVD-привод	DVD±RW ASUS DRW-24B5ST Black Bulk	-
Корпус	Logicpower 8702 – 550w 12cm	-
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	-
інше	Клавіатура, мишка	-
інше	Плата розширення Dynamode PCI-E-2xSATAIII-Marvell	-
Монітор	Монітор BenQ GL2450HM Black	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійного живлення	Powercom BNT-600AP USB	1400

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	8	11771	9416,8	103584,8
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	125216,3

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400

Продовження таблиці 7.8

1	2	3	4
Група 4			
3. Обчислювальна техніка	125216	-	-
Всього по групі	125216	50	62608
Група 5,6			
4. Вимірювальні пристрої	5190	-	-
5. Транспортні засоби	143000	-	
6. Господарський інвентар	28000	-	-
Всього по групі	176190	20	35238
7. Нематеріальні активи	60000	10	6000
Разом	$К_р$= 1769406		$А_р$= 174246

Примітка: вартість автомобіля Chevrolet Aveo 2010 взята за даними сайту «Авто-Ріа», джерело https://auto.ria.com/uk/auto_chevrolet_aveo_32795647.html, складає 143000 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де $N_е$ – Кількість екземплярів програм, шт.

$$З_о = 737 \cdot 150/57 = 1940 \text{ грн}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 1940 \cdot 10 \cdot 0{,}01 = 194 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=22% від суми основної та додаткової зарплати

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де $Н_с$ – відрахування на соціальні потреби, %

$$С_{оц} = 0{,}01 \cdot 22(1940+194) = 470 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом *Нг*=15% від основної зарплати

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 1940 \cdot 15 \cdot 0{,}01 = 291 \text{ грн}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картріджей, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,4 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=200 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 200 \cdot 0{,}4 = 80 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 10):

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 23,6 грн./шт., DVD-R box – 46,6 грн./шт.

$$З_{М2} = 46{,}6 \cdot 10 = 466 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \tag{7.18}$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574 + 558 + 570 = 1702 \text{ грн.}$$

$$З_М = (80 + 466 + 1702)/57 = 39 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \tag{7.19}$$

де $Н_п$ – норматив витрат на освоєння нових мов програмування, %

$$О_п = 1940 \cdot 15 \cdot 0{,}01 = 291 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 57 прим.)

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \tag{7.20}$$

де $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 174246 \cdot 2/(57 \cdot 12) = 509 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \tag{7.21}$$

$$С_п = 1940 + 194 + 470 + 291 + 39 + 291 + 509 = 3734 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (Рп) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%

$$П_р = 0{,}01 \cdot P_п \cdot C_п, \quad (7.22)$$

де $P_с$ – рівень рентабельності, %

$$П_р = 0{,}01 \cdot 50 \cdot 3734 = 1867 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	1940
2. Додаткова зарплата виконавців	$З_д$	194
3. Відрахування на соціальні потреби	$С_{оц}$	470
4. Загальногосподарські витрати	$Г_{осп}$	291
5. Витрати на матеріали	$З_М$	39
6. Освоєння нових операційних систем, мов програмування	$О_п$	291
7. Амортизація основних фондів	$А_м$	509
8. Повна собівартість програмного забезпечення	$С_п$	3734
9. Плановий прибуток	$П_р$	1867
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	5601
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1120,2
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	6721,2

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	6721
Всього капітальних витрат	–	6721

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позна-чення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	33550	20130
2. Витрати на електроенергію	$З_{ел}$	4608	4224
3. Витрати на амортизацію	$З_{ам}$	0	3361
Всього витрат за рік	$І$	38158	27715

Витрати на технічне обслуговування:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \quad (7.23)$$

де $Т_р$ – кількість годин обслуговування системи за рік, *год.*,

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 250 годин на рік до 150 годин на рік, тому витрати на технічне обслуговування зменшилися з

$$З_{р\ баз} = 250 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 33550\ грн.$$

до

$$З_{р\ нов} = 150 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 20130\ грн.$$

Витрати на електроенергію визначаються з урахуванням спожитої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$).

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

Зел баз=0,2·7200·3,2=4608 грн

Зел нов=0,2·6600·3,2=4224 грн

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	50	–	6721	–	3360,5
Всього відрахувань	-	–	6721	–	3360,5

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (5601 - 3734) \cdot 57 - (0,05 \cdot 1408000 + 0,5 \cdot 125216 +$$
$$+0,2 \cdot 176190 + 0,1 \cdot 60000) \cdot 2/12 = 77378 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{361406}{(5601 - 3734) \cdot 57 \cdot 12 / 2} = 0,6 \text{ років}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (38158\text{-}27715)\text{-}0,5 \cdot 6721 = 7083 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н} \quad (7.28)$$

$$T_{сп} = \frac{6721}{38158 - 27715} = 0,6 \text{ років}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	57
2. Повна собівартість розробленої програми	Грн.	3774
3. Ціна розробленої програми	Грн.	5601
4. Плановий прибуток від реалізації розробленої програми	Грн.	1827
5. Рентабельність програмної продукції	%	50
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1769406
7. Загальний прибуток від реалізації програмної продукції	Грн.	104139
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	77378
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,6
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	6721
11.Величина економічного ефекту у користувача програмної продукції	Грн.	7083
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,6

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Аналізуючи умови працівників ІТ-сфери, на перший погляд, може здатися, що працівники сфери інформаційних технологій не схильні до ризиків на виробництві, та якщо більш глибоко розглянути умови і специфіку праці фахівців сфері it-індустрії, можна виявити ряд факторів які будуть мати негативний сплив на стан охорони праці, та на самого it-фахівця зокрема. Сюди можна віднести як невідповідність освітлення, так і високий рівень шуму, що негативно позначатимуться як на емоційному так і на фізичному стані фахівця, призводитимуть до зниження ефективності праці та виробничих травм. Також, важливим моментом охорони праці it-фахівця є врахування його психологічних можливостей (швидкість реакції, особливості пам'яті та уваги, емоційний стан, тощо). Для того, щоб забезпечити ефективну роботу it-фахівця, потрібно враховувати та максимально компенсувати такі негативні фактои як: надмірне нервово-емоційне навантаження, довготривалі статичні перевантаження, обмежена рухова активність. Всі ці чинники призводить до різноманітних відхилень у стані здоров’я, зокрема до перевтоми, зниження фізичної та розумової працездатності, неврозів, захворювань серцево-судинної системи тощо. Метою даного розділу є огляд конкретних умов праці спеціаліста у сфері it-індустрії. Завданнями для даного розділу є: аналіз умов праці на робочому місці фахівця it-індустрії, розробка конкретних рекомендацій щодо покрашення умов праці фахівців it-індустрії, огляд пожежної безпеки на it-підприємстві та розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ – фахівці.

8.2 Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця (або програміста) виникають небезпечні та шкідливі для безпечної життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів. Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри. Робота ІТ-фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним

напруженням) категорій. В таблиці 8.1. наведені оптимальні параметри мікроклімату в приміщеннях.

Таблиця 8.1 – Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24°С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С 40...60% 0,1...0,2 м/с

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року. Слід зазначити, що для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у таблиці 8.2.

Таблиця 8.2 – Норми подачі свіжого повітря в приміщення

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$ на одну людину в годину
Об'єм до 20 $м^3$ на людину 20... 40 $м^3$ на людину Більше 40 $м^3$на людину	Не менше 30 Не менше 20 Може біти використана природна вентиляція

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, позитивно впливає на продуктивність праці. Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів – малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань. [2]

8.3 Пропозиції щодо підвищення працездатності ІТ-фахівців

Поява та впровадження нових інформаційно-комунікаційних технологій зумовлює необхідність подальшого вдосконалення охорони праці фахівців it-індустрії. Все це потребує розробки нових нормативно-правових актів з регламентації праці та відпочинку фахівців it-індустрії і стандартів підприємств,

центрів комп'ютерної техніки, центрів інформаційних технологій, сучасних комп'ютерних класів. Для підвищення розумової працездатності то зорової роботи повинна здійснюватися ергономічна оптимізація в рамках системи «оператор-термінал», яка сприятиме результативній фізичній та інтелектуальний працездатності і відновленню психосоматичного здоров'я фахівців it-індустрії. Всі наведені заходи щодо вдосконалення охорони праці фахівців it-індустрії повинні контролюватися службою охорони праці та комісією з охорони праці підприємства. Особливе значення у соціальному захисті цієї категорії працівників належить прийняття комплексного договору, який може забезпечити фахівців додатковими пільгами та компенсаціями.

Для більшого розуміння, пропозиції щодо підвищення працездатності it-фахівців, розіб'ємо на декілька категорій:

1 Середовище і розпорядок праці. Для мінімізації негативних ефектів, що пов'язані з перевтомленням it-фахівців, потрібно чітко прописати і реалізувати графік періодів праці-відпочинку, щоб фахівець міг можливість переключити увагу, дати можливість відпочити очам, мозку, елементарно, встати розім'яти ноги. Також потрібно зробити максимально комфортними умови мікроклімату у офісному приміщенні, де працюють it-фахівці. Мається на увазі встановлення і експлуатація, коли виникає необхідність, кондиціонерів, опалення, та системи вентиляції, задля попередження перегрівання, переохолодження it-фахівців, і подальшої неможливості ними виконувати свої функції. Також, за можливості, нами пропонується введення практики віддаленої праці it-фахівцями, якщо роботодавець не може забезпечити оптимальні і безпечні умови в офісному приміщенні, або якщо фахівця вони не влаштовують із певних причин.

2 Фізичні і психоемоційні чинники. Першим і найважливішим чинником, що впливає на працездатність it-фахівців є робоче місце, і саме тому, роботодавець має забезпечити максимальний його комфорт і безпеку. Гарантією цих факторів може слугувати сертифікація меблів, що використовуються на підприємстві it-галузі. Тому нами пропонується закупівля тільки меблів , які пошли сертифікацію

на відповідність. Під психоемоційними чинниками ми розуміємо гарне самопочуття фахівців, позитивний настій, гарний психологічний клімат у колективі, тощо. Задля того, щоб психоемоційні чинники мали максимально позитивний ефект, керівництву слід поводити заходи, які сприятимуть укріпленню і покращенню міжособистісних стосунків у колективі, таких як психологічні тренінги, тімбілдінг, спортивні змагання і естафети. Також, сюди можна віднести розробку і впровадження системи мотивації працівників, як фінансової, так моральної і адміністративної.

8.4 Розрахункова частина

Система освітлення робочого місця користувача ПК має відповідати наступним вимогам (рис. 8.1).

Рисунок 8.1 – Вимоги до системи освітлення робочого місця користувача ПК

Проводемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 7 м, довжина – 8 м, висота – 3,3 м.

У зазначеному приміщенні працює 9 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F=ESKZ/n,$$

де: *F* – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення.

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін **($\rho_{стін}$.)** і стелі **($\rho_{стелі}$),** значення коефіцієнтів дорівнюють **$\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.**

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де: *S* – площа приміщення, *S* = 56 $м^2$*;*

h – розрахункова висота підвісу, *h* = 3,2 *м. (співпадає з в*исотою стелі, т.я. лампи освітлення закріплюються на стелі)*;*

A – ширина приміщення, *A = 7 м;*

B – довжина приміщення, *B* = 8 *м.*

Підставимо всі значення у формулу та визначемо індекса приміщення: *i=1,1.*

Знаючи індекс приміщення, за знаходимо *n = 0,46 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам) [8].* Підставимо всі значення у формулу, визначемо світловий потік: *F=60260 Лм.*

Для розрахунку дудемо використовувати стельові *світлодіодні панелі Призма-72 6400К*, світловий потік яких $F_л$ = *7200* Лм.

Число ламп визначається по формулі:

$$N=F/F_л$$

де: *F* – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначемо індекса приміщення:

$$N= 60260 / 7200=8,3 \text{ шт.}$$

Приймаємо необхідну кількість *світлодіодних світильників* 9 шт.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів реінжинірингу та рефакторингу програмного коду до платформи .NET.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем реінжинірингу та рефакторингу програмного коду до платформи .NET.

– Досліджена система реінжинірингу та рефакторингу програмного коду до платформи .NET.

– На основі отриманих результатів досліджень створена програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання реінжинірингу та рефакторингу програмного коду до платформи .NET.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Delphi 10. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Lucifer.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 7083 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,6 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Безушко О.С. Дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.

2. Tilley S.R., Smith D.B. Legacy System Reengineering // 8th International Workshop on Software Technology and Engineering Practice (STEP97: July 15, 1997; London, UK) – см. http://www.sei.cmu.edu/reengineering/pubs/step97/tutorial/

3. Quilici A. Reverse Engineering: A Path Towards Success // ACM Communications, ICSE'95 P. 333-336

4. Adam Freeman. Pro Go The Complete Guide to Programming Reliable and Efficient Software Using Golang. Apress Media. 2022. 1078 p.

5. Fernando Doglio. Skills of a Successful Software Engineer. Manning. 2022. 182 c.

6. M. Holmes He. Creating Apps with React Native. Apress Media. 2022. 445 p.

7. Maurício Aniche. Effective Software Testing. Manning Publications. 2021. 372 p

8. Priscila Heller. Automating Workflows with GitHub Actions. Packt Publishing. 2021. 216 p.

9. JJ Geewax. API Design Patterns. Manning Publications Co. 2021. 481 p.

10. Prateek Prasad. App Design Apprentice. Razeware LLC. 2020. 272 p.

11. Dawn Griffiths, David Griffiths. Head First Android Development. O'Reilly Media, Inc. 2021. 1414 p.

12. Nathan Metzler. Kotlin Programming for Beginners. Independently published. 2021. 158 p.

13. Aaron Torres. Go Programming Cookbook Second Edition. Packt Publishing Ltd. 2019. 427 p.

14. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с.

15. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

16. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.

17. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.

18. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

19. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

20. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

21. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

22. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143

23. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools».

Communications in Computer and Information Science, 2021, vol 1486. Springer, Cham. pp 169-184.

24. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

25. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

26. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

27. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». International Journal of Computer Network and Information Security (IJCNIS). Vol. 12, No. 3, 2020. PP.33-43.

28. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

29. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

30. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

31. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

32. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

33. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P. 707-712.

34. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.701-706.

35. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

36. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629.

37. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС,

важливих для безпеки». Системи управління, навігації та зв'язку, 2023, вип. 2(72), С. 170-178.

38. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

39. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А. «Дослідження нормативної документації та стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». VI міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 20-21 квітня 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 35-36.

40. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

41. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

42. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

43. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

44. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019:

Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

45. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

46. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

47. O. Smirnov, O. Kovalenko, A. Kovalenko, S. Smirnov, V. Vialkova. The mathematical model of the testing technology for DOM XSS vulnerabilities. Scientific & practical cyber security journal (SPCSJ) Vol 2 Issue 1, 22-28 pp. [Электронный Журнал]. Georgia. Tbilisi: SCSA – 2018.

48. Oleksii Smirnov, Oleksandr Kovalenko, Jamil Al-Azzeh, Anna Kovalenko, Serhii Smirnov. Qualitative risk analysis of software development. Asian Journal of Information Technology. – Volume 17(3). – Medwell Journals. – 2018. – P. 218-230.

49. Смірнов О.А., Коваленко О.В., Коваленко А.С., Смірнов С.А. Розробка методу передтестової компіляції й розподілу доступу. Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 19-20 квітня 2018р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215

50. Smirnov Oleksii, Kovalenko Oleksandr, Kovalenko Anna, Smirnov Serhii. Method of testing the DOM XSS vulnerability. International Conference «Information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. 2017. P7.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 33-13 від 04.08.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи реінжинірингу та рефакторингу програмного коду до платформи .NET;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Delphi 10.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинні бути розглянуті пропозиції щодо підвищення працездатності ІТ-фахівців.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 123 аркуші.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2023 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 11.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

_____ Доренський О.П.

Дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 36

Літера: РП

Кропивницький – 2023 року

MAINUNIT.PAS – ФАЙЛ ОСНОВНОЇ ФОРМИ

```
unit MainUnit; // назва
interface

uses // бібліотеки що використовуються
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Menus,
ToolWin, ComCtrls, StdCtrls, ClipBrd, TreeUnit;

{
Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Дослідження та програмна реалізація системи реінжинірингу та рефакторингу
програмного коду до платформи .NET
Кафедра кібербезпеки та програмного забезпечення
Розробив: Безушко Олександр Сергійович
2023 рік
}

type
// Тип для зберігання
TNumber = record
  Ident: Word;
  Words: Word;
  Consts: W ord;
  Opers: Word;
  Devs: Word;
  Errors: Word;
  Total: Word;
  Strings: Word;
end;
// Клас - головна форма програми
TMainForm = class (TForm)
   StatusBar: TStatusBar;
   MainMenu1: TMainMenu;
   miFile: TMenuItem;
   miExit: TMen uItem;
   miBreak1: TMenuItem;
   miSaveAs: TMenuItem;
   miSave: TMenuItem;
   miOpen: TMenuItem;
   miNew: TMenuItem;
   SaveTextDialog: TSaveDialog;
   OpenTextDialog: TOpenDialog;
   Edit: TMemo;
   miEdit: TMenuItem;
   miUndo: TMenuItem;
   miBreak2: TMenuItem;
   miCut: TMenuItem;
   miCopy: TMenuItem;
   miPaste: TMenuItem;
   miSelectAll: TMenuItem;
   miAnalize: TMenuItem;
   miRunAnalizier: TMenuItem;
   miSyntaxAnalize: TMenuItem;
   miBreak3: TMenuItem;
   miCodePrg: TMenuItem;
   miLexemsCodeTable: TMenuItem;
   miBreak4: TMenuItem;
   miReservedWords: TMenuItem;
   miIdentefier: TMenuItem;
   miConstant: TMenuItem;
   miDev: TMenuItem;
   miOperations: TMenuItem;
   miErrors: TMenuItem;
   procedure miExitClick (Sender: TObject);
   procedure miNewClick (Sender: TObject);
```

```
   procedure miSaveClick (Sender: TObject);
   procedure miOpenClick (Sender: TObject);
   procedure miSaveAsClick (Sender: TObject);
   procedure EditChange (Sender: TObject);
   procedure FormActivate (Sender: TObject);
   procedure EditKeyUp (Sender: TObject; var Key: Word; Shift: TShiftState);
   procedure FormClose (Sender: TObject; var Action: TCloseAction);
   procedure miUndoClick (Sender: TObject);
   procedure miCutClick (Sender: TObject);
   procedure miCopyClick (Sender: TObject);
   procedure miPasteClick (Sender: TObject);
   procedure miSelectAllClick (Sender: TObject);
   procedure miEditClick (Sender: TObject);
   procedure miRunAnalizierClick (Sender: TObject);
   procedure miCodePrgClick (Sender: TObject);
   procedure miReservedWordsClick (Sender: TObject);
   procedure miIdentefierClick (Sender: TObject);
   procedure miConstantClick (Sender: TObject);
   procedure miDevClick (Sender: TObject);
   procedure miOperationsClick (Sender: TObject);
   procedure miErrorsClick (Sender: TObject);
   procedure miAnalizeClick (Sender: TObject);
   procedure miLexemsCodeTableClick (Sender: TObject);
public
// Ім'я редагованого файлу
FFileName: String;
// Покажчик на вершину дерева кодувань
FCodeTree: TSearchTree;
// Число різних лексемм
FLexems: TNumber;
// Закодована програма
FCodePrg: String;
// Метод - збереження тексту
function SaveFile: Boolean;
// Метод - визначення позиції курсору в редакторі
procedure EditTextPos;
// Метод - лексичний аналізатор редагованого файлу
procedure AnalizeCode;
end;

var MainForm: TMainForm;

implementation
uses ViewUnit, CodeUnit;
{$ R *. DFM}

const
// Безліч - символів-роздільники
Devs: set of char = [',', '(', ')', ':', ';'];
// Безліч - символів-оператори
Oper: set of char = ['+', '-', '*', '/', '<', '>', '='];
// Безліч - символів мови
Chars: set of char = ['A' .. 'Z', '0 '.. '9', '_'];


// Функція переводить рядок у верхній регістр і видаляє з неї
// перші і останні пробіли

function PrepareLine (S: String): String;
begin
// Видаляємо пробіли на початку рядка
while S [1] = '' do Delete (S, 1, 1);
// Видаляємо прогалини в кінці рядка
while S [Length (S)] = '' do Delete (S, Length (S), 1);
S:= AnsiUpperCase (S);
PrepareLine:= S;
end;

// Функція перевіряє чи є рядок S зарезервованим словом
```

```
function IsReservedWord (S: String): Boolean;
begin
Result:= (S = 'PROGRAM') or (S = 'VAR') or (S = 'INTEGER') or
(S = 'REAL')
or (S = 'BEGIN') or (S = 'IF') or (S = 'THEN') or (S = 'ELSE') or (S = 'FOR')
or (S = 'TO') or (S = 'DO') or (S = 'END') or (S = 'WRITELN')
or (S = 'READLN') or (S = 'WRITE') or (S = 'READ');
end;

// Функція перевіряє чи є рядок S ідентифікатором
function IsIdentifier (S: String): Boolean;
var I: Byte;
Res: Boolean;
begin
Res:= S [1] in ['A' .. 'Z'];
for I:= 1 to Length (S) do
if not (S [I] in ['0 '.. '9', 'A' .. 'Z']) then Res:= False;
if Res then Res:= (S <> 'AND') and (S <> 'OR') and (S <> 'NOT');
IsIdentifier:= Res
end;

// Функція перевіряє чи є рядок S константою
function IsConstant (S: String): Boolean;
var I: Byte;
Res, F: Boolean;
begin
Res:= True;
F:= False;
for I:= 1 to Length (S) do
if not (S [I] in ['.', '0 '.. '9']) then begin
if (S [I] = '.') and (not F) then F:= True
else Res:= False;
end;
IsConstant:= Res
end;

// Функція перевіряє чи є рядок S операцією
function IsOpera tion (S: String): Boolean;
begin
Result:= (S = '+') or (S = '-') or (S = '*') or (S = '/')
or(S= '<=') or (S = '> =') or (S = '=') or (S = '<') or (S = '>')
or (S = ':=') or (S = 'AND') or (S = 'OR') or (S = 'NOT');
end;

// Метод - аналізатор редагованого файлу
procedure TMainForm.AnalizeCode;
var I, J: Word; // Лічильники
CurrLine: String; // Поточна рядок тексту
CurrChar: Char; // Поточний символ рядка
ErrorLex: Boolean; // Помилкова лексемма
LocLexem: PI tem; // відшукувати лексемма
CurrLexem: TLexem; // Поточна лексемма
begin
FCodeTree:= TSearchTree.Create;
FCodePrg:='';
// Обнуляємо інформацію про поточну лексему
with CurrLexem do begin
Name:='';
Line:= 0;
Code:= lcUnknown;
CodeName:='';
end;
// Обнуляємо інформацію по лексемах
with FLexems do begin
Ident:= 0;
Words:= 0;
Consts:= 0;
Opers:= 0;
Devs:= 0;
```

```
Errors:= 0;
Total:= 0;
Strings:= 0;
end;
// Аналізуємо редагований текст порядково
for I:= 0 to Pred (Edit.Lines.Count) do begin
CurrLine:= PrepareLine (Edit.Lines [I]);
J:= 1;
// Аналізуємо поточний рядок посимвольно
while J <= Length (CurrLine) do begin
CurrChar:= CurrLine [J];
// Зустрівся рядок
if CurrChar ='''' then begin
Inc (J);
ErrorLex:= False;
repeat
if CurrLine [J] ='''' then begin
// Помилка
if J = Length (CurrLine) then begin
ErrorLex:= True;
       Break;
end;
// Подвійний апостроф
if CurrLine [Succ (J)] ='''' then begin
Inc (J, 2);
Continue;
end;
// Закриваючий апостроф
Inc (J);
Break;
end;
Inc (J);
until False;
// Аналізуємо результат
if ErrorLex then begin
// Помилкова лексемма
CurrLexem.Code:= lcError;
// Збільшуємо лічильник
Inc (FLexems.Errors);
// Формуємо текст закодованої програми
CurrLexem.CodeName:= 'E' + IntToStr (FLexems.Errors);
// Додаємо лексему в дерево
LocLexem:= FCodeTree.Search (CurrLexem);
// Додаємо в закодіроованную програму
  FCodePrg:= FCodePrg + LocLexem.Value.CodeName + '';
end
else begin
CurrLexem.Code:= lcString;
// Збільшуємо лічильник
Inc (FLexems.Strings);
// Формуємо текст закодованої програми
      CurrLexem.CodeName:= 'S' + IntToStr (FLexems.Strings);
// Додаємо лексему в дерево
LocLexem:= FCodeTree.Search (CurrLexem);
// Додаємо в закодіроованную програму
FCodePrg:= FCodePrg + LocLexem.Value.CodeNam e + '';
end;
// Продовжити аналіз подальшого символу
Continue;
end;
// Зустрівся символ-роздільник, операція або кінець рядка
if (CurrChar = '') or (J = Length (CurrLine)) or (CurrChar in Oper) or (CurrC
har in Devs) then
begin
if (J = Length (CurrLine)) and (CurrChar <> '') and (not (CurrChar in Oper))
and (not (CurrChar in Devs)) then CurrLexem.Name:= CurrLexem.Name + CurrChar;
if CurrLexem.Name <>'' then begin
// Неправильна лексема
ErrorLex:= True;
```

```
// Лексема раніше не зустрічалася
LocLexem:= FCodeTree.Locate (CurrLexem);
if LocLexem = nil then begin
CurrLexem.Line:= Succ (I);
// Зарезервоване слово
if IsReservedWord (CurrLexem.Name) and ErrorLex then begin
CurrLexem.Code:= lcReservedWord;
// Збільшуємо лічильник
Inc (FLexems.Words);
// Формуємо текст закодованої програми
CurrLexem.CodeName:= 'W' + IntToStr (FLexems.Words);
// Додаємо лексему в дерево
LocLexem:= FCodeTree.Search (CurrLexem);
// Додаємо в закодіроованную програму
FCodePrg:= FCodePrg + LocLexe m.Value.CodeName + '';
// Продовжуємо аналіз
ErrorLex:= False;
end;
// Ідентифікатор
if IsIdentifier (CurrLexem.Name) and ErrorLex then begin
CurrLexem.Code:= lcIdentefier;
// Збільшуємо лічильник
Inc (FLexems.Ident);
// Формуємо текст закодованої програми
CurrLexem.CodeName:= 'I' + IntToStr (FLexems.Ident);
// Додаємо лексему в дерево
LocLex em:= FCodeTree.Search (CurrLexem);
// Додаємо в закодіроованную програму
FCodePrg:= FCodePrg + LocLexem.Value.CodeName + '';
// Продовжуємо аналіз
ErrorLex:= False;
end;
// Мітка
if IsConstant (CurrLexem.Name) and ErrorLex and (CurrLine [J] = ':') then begin
CurrLexem.Code:= lcLabel;
// Збільшуємо лічильник
Inc (FLexems.Consts);
// Формуємо текст закодованої програми
CurrLexem.CodeName:= 'L' + IntToStr (FLexems.Consts);
// Додаємо лексему в дерево
LocLexem:= FCodeTree.Search (CurrLexem);
// Додаємо в закодіроованную програму
FCodePr g:= FCodePrg + LocLexem.Value.CodeName + '';
// Продовжуємо аналіз
ErrorLex:= False
end;
// Константа
if IsConstant (CurrLexem.Name) and ErrorLex then begin
CurrLexem.Code:= lc Constant;
// Збільшуємо лічильник
Inc (FLexems.Consts);
// Формуємо текст закодованої програми
CurrLexem.CodeName:= 'C' + IntToStr (FLexems.Consts);
// Додаємо лексему в дерево
    LocLexem:= FCodeTree.Search (CurrLexem);
// Додаємо в закодовану програму
FCodePrg:= FCodePrg + LocLexem.Value.CodeName + '';
// Продовжуємо аналіз
ErrorLex:= False;
en d;
// Зустріли операцію
if IsOperation (CurrLexem.Name) and ErrorLex then begin
CurrLexem.Code:= lcOperation;
// Збільшуємо лічильник
Inc (FLexems.Opers);
// Формуємо текст закодованої програми
CurrLexem.CodeName:= 'O' + IntToStr (FLexems.Opers);
// Додаємо лексему в дерево
LocLexem:= FCodeTree.Search (CurrLexem);
```

```
// Додаємо в закодовану програму
FCodePrg:= FCodePrg + LocLexem.Value.CodeName + '';
// Продовжуємо аналіз
ErrorLex:= False;
end;
// Помилкова лексемма
if ErrorLex then begin
CurrLexem.Code:= lcError;
     // Збільшуємо лічильник
Inc (FLexems.Errors);
// Формуємо текст закодованої програми
CurrLexem.CodeName:= 'E' + IntToStr (FLexems.Errors);
// Додаємо лексему в дерево
LocLex em:= FCodeTree.Search (CurrLexem);
// Додаємо в закодовану програму
FCodePrg:= FCodePrg + LocLexem.Value.CodeName + '';
end;
end
// Лексема вже є в дереві
else FCodePrg:= F CodePrg + LocLexem.Value.CodeName + '';
end;
// Визначаємо роздільник (Якщо є)
if (CurrChar in Oper) or (CurrChar in Devs) then begin
if J <Pred (Length (CurrLine)) then begin
if IsOperation (CurrChar + Curr Line [Succ (J)]) then begin
CurrLexem.Name:= CurrChar + CurrLine [Succ (J)];
Inc (J);
end
else CurrLexem.Name:= CurrChar;
end
else CurrLexem.Name:= CurrChar;
LocLexem:= FCodeTree.Locate (CurrLexem);
// Зустріли операцію
if (IsOperation (CurrLexem.Name)) and (LocLexem = nil) then begin
CurrLexem.Code:= lcOperation;
// Збільшуємо лічильник
Inc (FLexems.Opers);
// Формуємо текст програми
CurrLexem.CodeName:= 'O' + IntToStr (FLexems.Opers);
// Додаємо лексему в дерево
LocLexem:= FCodeTree.Search (CurrLexem);
end;
// Зустріли роздільник
if (CurrChar in Devs) and (LocLexem = nil) then begin
CurrLexem.Code:= lcDev;
// Збільшуємо лічильник
Inc (FLexems.Devs);
// Формуємо текст програми
CurrLexem.CodeName:= 'R' + IntToStr (FLexems.Devs);
// Додаємо лексему в дерево
LocLexem:= FCodeTree.Search (CurrLexem);
end;
// Додаємо в програму
if (LocLexem.Value.Name = ':') and (CurrLine [Succ (J)] = '=') then
else FCodePrg:= FCodePrg + LocLexem.Value.CodeName + '';
end;
// Обнуляємо інформацію про поточну лексему
with CurrLexem do begin
Name:='';
Line:= 0;
Code:= lcUnknown;
CodeName:='';
end;
end
else if CurrChar <> '' then CurrLexem.Name:= CurrLexem.Name + CurrChar;
Inc (J);
end;
FCodePrg:= FCodePrg + # 13 # 10
end
```

```
end;

// Метод - визначення позиції курсору в редакторі
procedure TMainForm.EditTextPos;
var LineNum, CharNum: LongInt;
begin
LineNum:= Edit.Perform (EM_LINEFROMCHAR, Edit.SelStart, 0);
CharNum:= Edit.Perform (EM_LINEINDEX, LineNum, 0);
MainForm.StatusBar.Panels [0]. Text:= IntToStr (Succ (LineNum)) + ':' + IntToStr
(Succ ((Edit.SelStart - CharNum)));
end;

// Метод - збереження тексту
function TMainForm.SaveFile: Boolean;
begin
SaveFile:= True;
if FFileName ='' then begin
// Відкриваємо діалог
if SaveTextDialog.Execute then begin
FFileName:= SaveTextDialog.FileName;
// Формуємо ім'я файлу
if ExtractFilePath (FFileName) ='' then
if FFileName [Length (FFileName)] = '.' then FFileName:= FFileName + '.' +
SaveTextDialog.DefaultExt
else FFileName:= FFileName + SaveTextDialog.DefaultExt;
end
else SaveFile:= False;
end;
if FFileName <>'' then Edit.Lines.SaveToFile (FFileName);
end;

procedure TMainForm.miExitClick (Sender: TObject);
begin
Close
end;

procedure TMainForm.miNewClick (Sender: TObject);
var Ans: Integer;
begin
if miSave.Enabled then begin
Ans:= MessageBox (Handle, 'Файл не збережений!
Зберегти його зараз?',
'Попередження', MB_YESNOCANCEL or MB_DEFBUTTON3 or MB_ICONQUESTION);
if Ans = mrCancel then Exit;
if Ans = mrYes then
if not SaveFile then Exit
end;
FFileName:='';
Edit.Lines.Clear;
miSave.Enabled:= False;
miSaveAs.Enabled:= True;
EditTextPos;
Edit.SetFocus
end;

procedure TMainForm.miSaveClick (Sender: TObject);
begin
SaveFile;
end;

procedure TMainForm.miOpenClick (Sender: TObject);
var Ans: Integer;
begin
if miSave.Enabled then begin
Ans:= MessageBox (Handle, 'Файл не збережений! Зберегти його зараз?',
'Попередження', MB_YESNOCANCEL or MB_DEFBUTTON3 or MB_ICONQUESTION);
if Ans = mrCancel then Exit;
if Ans = mrYes then
if not SaveFile then Exit
```

```
end;
if OpenTextDialog.Execute then begin
FFileName:= OpenTextDialog.FileName;
Edit.Lines.LoadFromFile (FFileName);
miSave.Enabled:= False;
miSaveAs.Enabled:= True;
EditTextPos
end;
Edit.SetFocus
end;

procedure TMainForm.miSaveAsClick (Sender: TObject);
var FN: String;
begin
if not SaveFile then FFileName:= FN
end;

procedure TMainForm.EditChange (Sender: TObject);
begin
if not miSave.Enabled then miSave.Enabled:= True;
if not miUndo.Enabled then miUndo.Enabled:= True;
end;

procedure TMainForm.FormActivate (Sender: TObject);
begin
miNewClick (Sender)
end;

procedure TMainForm.EditKeyUp (Sender: TObject; var Key: Word; Shift:
TShiftState);
begin
EditTextPos
end;

procedure TMainForm.FormClose (Sender: TObject; var Action: TCloseAction);
var Ans: Integer;
begin
if miSave.Enabled then begin
Ans:= MessageBox (Handle, 'Файл не збережений! Зберегти його зараз?',
'Попередження', MB_YESNOCANCEL or MB_DEFBUTTON3 or MB_ICONQUESTION);
if Ans = mrCancel then Action:= caNone;
if Ans = mrYes then
if not SaveFile then Action:= caNone
end
end;

procedure TMainForm.miUndoClick (Sender: TObject);
begin
with Edit do if HandleAllocated then
SendMessage (Handle, EM_UNDO, 0, 0)
end;

procedure TMainForm.miCutClick (Sender: TObject);
begin
Edit.CutToClipboard
end;

procedure TMainForm.miCopyClick (Sender: TObject);
begin
Edit.CopyToClipboard
end;

procedure TMainForm.miPasteClick (Sender: TObject);
begin
Edit.PasteFromClipboard
end;

procedure TMainForm.miSelectAllClick (Sender: TObject);
begin
```

```
Edit.SelectAll
end;

procedure TMainForm.miEditClick (Sender: TObject);
var HasSeleted: Boolean;
begin
HasSeleted:= Edit.SelLength <> 0;
miPaste.Enabled:= Clipboard.HasFormat (CF_TEXT);
miCut.Enabled:= HasSeleted;
miCopy.Enabled:= HasSeleted;
end;

procedure TMainForm.miRunAnalizierClick (Sender: TObject);
begin
AnalizeCode
end;

procedure TMainForm.miCodePrgClick (Sender: TObject);
begin
CodeForm.Memo.Lines.SetText (PChar (FCodePrg));
CodeForm.ShowModal;
end;

procedure TMainForm.miReservedWordsClick (Sender: TObject);
var K: Integer;
S: String;
begin
with ViewForm do begin
Caption:= 'Зарезервовані слова';
sgridView.RowCount:= Succ (FLexems.Words);
for K:= 1 to FLexems.Words do begin
S:= 'W' + IntToStr (K);
sgridView.Cells [0, K]:= IntToStr (K);
sgridView.Cells [1, K]:= S;
sgridView.Cells [2, K]:= FCodeTree.Info (S);
end;
end;
ViewForm.ShowModal;
end;

procedure TMainForm.miIdentefierClick (Sender: TObject);
var K: Integer;
S: String;
begin
with ViewForm do begin
Caption:= 'Ідентифікатори';
sgridView.RowCount:= Succ (FLexems.Ident);
for K:= 1 to FLexems.Ident do begin
S:= 'I' + IntToStr (K);
sgridView.Cells [0, K]:= IntToStr (K);
sgridView.Cells [1, K]:= S;
sgridView.Cells [2, K]:= FCodeTree.Info (S)
end
end;
ViewForm.ShowModal
end;

procedure TMainForm.miConstantClick (Sender: TObject);
var K: Integer;
S: String;
begin
with ViewForm do begin
Caption:= 'Константи';
sgridView.RowCount:= Succ (FLexems.Consts);
for K:= 1 to FLexems.Consts do begin
S:= 'C' + IntToStr (K);
sgridView.Cells [0, K]:= IntToStr (K);
sgridView.Cells [1, K]:= S;
sgridView.Cells [2, K]:= FCodeTree.Info (S)
```

```
end
end;
ViewForm.ShowModal
end;

procedure TMainForm.miDevClick (Sender: TObject);
var K: Integer;
S: String;
begin
with ViewForm do begin
Caption:= 'Роздільники';
sgridView.RowCount:= Succ (FLexems.Devs);
for K:= 1 to FLexems.Devs do begin
S:= 'R' + IntToStr (K);
sgridView.Cells [0, K]:= IntToStr (K);
sgridView.Cells [1, K]:= S;
sgridView.Cells [2, K]:= FCodeTree.Info (S)
end
end;
ViewForm.ShowModal
end;

procedure TMainForm.miOperationsClick (Sender: TObject);
var K: Integer;
S: String;
begin
with ViewForm do begin
Caption:= 'Операції';
sgridView.RowCount:= Succ (FLexems.Opers);
for K:= 1 to FLexems.Opers do begin
S:= 'O' + IntToStr (K);
sgridView.Cells [0, K]:= IntToStr (K);
sgridView.Cells [1, K]:= S;
sgridView.Cells [2, K]:= FCodeTree.Info (S)
end
end;
ViewForm.ShowModal
end;

procedure TMainForm.miErrorsClick (Sender: TObject);
var K: Integer;
S: String;
begin
with ViewForm do begin
Caption:= 'Помилкові лексемми';
sgridView.RowCount:= 2;
for K:= 1 to FLexems.Errors do begin
S:= 'E' + IntToStr (K);
sgridView.Cells [0, K]:= IntToStr (K);
sgridView.Cells [1, K]:= S;
sgridView.Cells [2, K]:= FCodeTree.Info (S);
end;
end;
ViewForm.ShowModal
end;

procedure TMainForm.miAnalizeClick (Sender: TObject);
begin
miCodePrg.Enabled:= FCodePrg <>'';
miReservedWords.Enabled:= FCodePrg <>'';
miIdentefier.Enabled:= FCodePrg <>'';
miConstant.Enabled:= FCodePrg <>'';
miDev.Enabled:= FCodePrg <>'';
miOperations.Enabled:= FCodePrg <>'';
miErrors.Enabled:= FCodePrg <>'';
miLexemsCodeTable.Enabled:= FCodePrg <>'';
miSyntaxAnalize.Enabled:= Edit.Lines.Count> 0;
miRunAnalizier.Enabled:= Edit.Lines.Count> 0;
end;
```

```
procedure TMainForm.miLexemsCodeTableClick (Sender: TObject);
var S: String;
I, N: Integer;
begin
with ViewForm do begin
Caption:= 'Таблиця кодів лексемм';
N:= 1;
S:='';
for I:= 1 to Length (FCodePrg) do begin
if FCodePrg [I] = '' then begin
sgridView.RowCount:= Succ (N);
sgridView.Cells [0, N]:= IntToStr (N);
sgridView.Cells [1, N]:= S;
sgridView.Cells [2, N]:= FCodeTree.Info (S);
Inc (N);
S:='';
end
else if not (FCodePrg [I] in [# 13, # 10]) then S:= S + FCodePrg [I];
end;
ShowModal
end
end;

end.
```

VIEWUNIT.PAS – ФОРМА ТАБЛИЦІ НАЛАШТУВАНЬ ЛЕКСИЧНОГО АНАЛІЗАТОРА

```
unit ViewUnit; //назва модулю
interface // інтерфейс на частина

uses //бібліотеки
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Grids,
StdCtrls;

type

TViewForm = class (TForm) // клас
sgridView: TStringGrid;
btnOk: TButton;
procedure btnOkClick (Sender: TObject);
procedure FormCreate (Sender: TObject);
end;

var ViewForm: TViewForm;

implementation  // реалізація
{$ R *. DFM} // ресурси


procedure TViewForm.btnOkClick (Sender: TObject);
begin
Close
end;

procedure TViewForm.FormCreate (Sender: TObject);
begin
with sgridView do begin
Cells [0, 0]:= 'Порядковий номер';
Cells [1, 0]:= 'Кодове ім'я';
Cells [2, 0]:= 'Розшифровка';
end;
end;

end.
```

LA.PAS – ПРОГРАМА ЛЕСКИЧНОГО АНАЛІЗАТОРА

```
program LA;

{ Об'ява констант }
const TAB = ^I;
       CR  = ^M;
       LF  = ^J;

{ Об'ява типів даних }

type Symbol = string[8];
      SymTab = array[1..1000] of Symbol;
      TabPtr = ^SymTab;

{ Об'ява Variable }

var Look  : char;              { символ }
     Token : char;              { росшифрований токен }
     Value : string[16];        { Token     }
     Lcount: integer;           { лічильник }

{ Визначення ключових слів і типів маркерів }
const KWlist: array [1..4] of Symbol =
               ('IF', 'ELSE', 'ENDIF', 'END');

const KWcode: string[5] = 'xilee';

{ Читання нового символу з вхідного потоку }
procedure GetChar;
begin
    Read(Look);
end;

{ Сповішення про помилку }
procedure Error(s: string);
begin
    WriteLn;
    WriteLn(^G, 'Error: ', s, '.');
end;

{ Сповішення про помилку та вихід }
procedure Abort(s: string);
begin
    Error(s);
    Halt;
end;

{ Сповішення виключення}
procedure Expected(s: string);
begin
    Abort(s + ' Expected');
end;

{ Визнати буквений символ }
function IsAlpha(c: char): boolean;
begin
    IsAlpha := UpCase(c) in ['A'..'Z'];
end;

{ Визнати число }
function IsDigit(c: char): boolean;
begin
    IsDigit := c in ['0'..'9'];
end;

{ Визнати алфавітно-цифрові символи }
function IsAlNum(c: char): boolean;
```

```
begin
    IsAlNum := IsAlpha(c) or IsDigit(c);
end;

function IsAddop(c: char): boolean;
begin
    IsAddop := c in ['+', '-'];
end;

function IsMulop(c: char): boolean;
begin
    IsMulop := c in ['*', '/'];
end;

{ Визнати пробіл }
function IsWhite(c: char): boolean;
begin
    IsWhite := c in [' ', TAB];
end;

{ Пропустити початкові пробіли }
procedure SkipWhite;
begin
    while IsWhite(Look) do
       GetChar;
end;

{ Специфічні вхідні символи }
procedure Match(x: char);
begin
    if Look <> x then Expected('''' + x + '''');
    GetChar;
    SkipWhite;
end;


procedure Fin;
begin
    if Look = CR then GetChar;
    if Look = LF then GetChar;
    SkipWhite;
end;
{ Table Lookup }
function Lookup(T: TabPtr; s: string; n: integer): integer;
var i: integer;
     found: boolean;
begin
    found := false;
    i := n;
    while (i > 0) and not found do
       if s = T^[i] then
          found := true
       else
          dec(i);
    Lookup := i;
end;

{ Отримання ідентифікатора }
procedure GetName;
begin
    while Look = CR do
       Fin;
    if not IsAlpha(Look) then Expected('Name');
    Value := '';
    while IsAlNum(Look) do begin
      Value := Value + UpCase(Look);
      GetChar;
    end;
    SkipWhite;
```

```
end;

{ Отримання номера }
procedure GetNum;
begin
    if not IsDigit(Look) then Expected('Integer');
    Value := '';
    while IsDigit(Look) do begin
      Value := Value + Look;
      GetChar;
    end;
    Token := '#';
    SkipWhite;
end;

{ Отримати ідентифікатор і перевірити його на ключові слова }
procedure Scan;
begin
    GetName;
    Token := KWcode[Lookup(Addr(KWlist), Value, 4) + 1];
end;

procedure MatchString(x: string);
begin
    if Value <> x then Expected('''' + x + '''');
end;

{ Створення унікальної мітки }
function NewLabel: string;
var S: string;
begin
    Str(LCount, S);
    NewLabel := 'L' + S;
    Inc(LCount);
end;

{ Виведення через Label }
procedure PostLabel(L: string);
begin
    WriteLn(L, ':');
end;

{ Вивести рядок з Tab }
procedure Emit(s: string);
begin
    Write(TAB, s);
end;

{ Вихідний рядок з Tab і CRLF }
procedure EmitLn(s: string);
begin
    Emit(s);
    WriteLn;
end;
{--------------------------------------------------------------}

{ Розібрати і перевести ідентифікатор }
procedure Ident;
begin
    GetName;
    if Look = '(' then begin
       Match('(');
       Match(')');
       EmitLn('BSR ' + Value);
       end
    else
       EmitLn('MOVE ' + Value + '(PC),D0');
end;
```

```
{-----------------------------------------------------------------}

{ Розібрати і перевести Math Factor }
procedure Expression; Forward;
procedure Factor;
begin
    if Look = '(' then begin
       Match('(');
       Expression;
       Match(')');
       end
    else if IsAlpha(Look) then
       Ident
    else begin
       GetNum;
       EmitLn('MOVE #' + Value + ',D0');
    end;
end;
{-----------------------------------------------------------------}



{ Розібрати і перевести перший фактор }
procedure SignedFactor;
var s: boolean;
begin
    s := Look = '-';
    if IsAddop(Look) then begin
       GetChar;
       SkipWhite;
    end;
    Factor;
    if s then
       EmitLn('NEG D0');
end;

{ Визнати і перевести Multiply }
procedure Multiply;
begin
    Match('*');
    Factor;
    EmitLn('MULS (SP)+,D0');
end;

{-----------------------------------------------------------------}

{ Визнати і перевести Divide }
procedure Divide;
begin
    Match('/');
    Factor;
    EmitLn('MOVE (SP)+,D1');
    EmitLn('EXS.L D0');
    EmitLn('DIVS D1,D0');
end;
{-----------------------------------------------------------------}

{ Завершення терміну переробки (названий термін і FirstTerm) }

procedure Term1;
begin
    while IsMulop(Look) do begin
       EmitLn('MOVE D0,-(SP)');
       case Look of
        '*': Multiply;
        '/': Divide;
       end;
    end;
end;
```

```
{-----------------------------------------------------------------}
{ Розібрати і перевести Math Term }
procedure Term;
begin
    Factor;
    Term1;
end;

{-----------------------------------------------------------------}
{ Розібрати і перевести Math Term з можливою провідною ознакою }
procedure FirstTerm;
begin
    SignedFactor;
    Term1;
end;

{-----------------------------------------------------------------}
{ Визнати і перевести Add }
procedure Add;
begin
    Match('+');
    Term;
    EmitLn('ADD (SP)+,D0');
end;
{-----------------------------------------------------------------}

{ Визнати і перевести Subtract }
procedure Subtract;
begin
    Match('-');
    Term;
    EmitLn('SUB (SP)+,D0');
    EmitLn('NEG D0');
end;

{-----------------------------------------------------------------}
{ Розібрати і перевести вираз }
procedure Expression;
begin
    FirstTerm;
    while IsAddop(Look) do begin
       EmitLn('MOVE D0,-(SP)');
       case Look of
        '+': Add;
        '-': Subtract;
       end;
    end;
end;

{-----------------------------------------------------------------}
{ Розібрати і перевести логічну умову }
Procedure Condition;
begin
    EmitLn('Condition');
end;

{-----------------------------------------------------------------}
{ Визнати і перевести}
procedure Block; Forward;
procedure DoIf;
var L1, L2: string;
begin
    Condition;
    L1 := NewLabel;
    L2 := L1;
    EmitLn('BEQ ' + L1);
    Block;
    if Token = 'l' then begin
```

```
        L2 := NewLabel;
        EmitLn('BRA ' + L2);
        PostLabel(L1);
        Block;
    end;
    PostLabel(L2);
    MatchString('ENDIF');
end;

{ Визнати і перевести оператор присвоювання }
procedure Assignment;
var Name: string;
begin
    Name := Value;
    Match('=');
    Expression;
    EmitLn('LEA ' + Name + '(PC),A0');
    EmitLn('MOVE D0,(A0)');
end;

{ Визнати і перевести початковий блок }
procedure Block;
begin
    Scan;
    while not (Token in ['e', 'l']) do begin
        case Token of
         'i': DoIf;
         else Assignment;
        end;
        Scan;
    end;
end;

{ Розібрати і перевести програму }
procedure DoProgram;
begin
    Block;
    MatchString('END');
    EmitLn('END')
end;

{ Ініціалізація }
procedure Init;
begin
    LCount := 0;
    GetChar;
end;

end.
```

ФАЙЛ UNIT1.PAS – НАЛАШТУВАННЯ ПРОГРАМИ

```
unit Settings;
interface
uses
  Classes, Controls, Messages, Windows, StdCtrls, FileCtrl,
  ShellApi, Graphics, RzTreeVw, ComCtrls, CommCtrl, RzCommon;
type
  TDTypes = set of TDriveType;
  TDBits = set of 0..25;
  TRzFileInfo = class
    Name: string;
    Attr: Integer;
    Time: Longint;
    Size: Longint;
    IsDirectory: Boolean;
    IconHandle: THandle;
  end;
  TReingnDirectoryTree = class;
  TReingnTableBox = class( TFileBox )
  private
    FAboutInfo: TReingnAboutInfo;
    FDirTree: TReingnDirectoryTree;
    FFileInfoList: TStringList;
    FShowLongNames: Boolean;
    FAllowOpen: Boolean;
    FFormColor: TColor;
    FFormController: TReingnFormController;
    FFormFlat: Boolean;
    FFormFlatStyle: TFrameStyle;
    FFormFocusStyle: TFormStyle;
    FFormSides: TSides;
    FFormStyle: TFormStyle;
    FFormVisible: Boolean;
    FUseFormController: Boolean;
    FOnMouseEnter: TNotifyEvent;
    FOnMouseLeave: TNotifyEvent;
    FInReadFileNames: Boolean;
    procedure ResetItemHeight;
  protected
    FCanvas: TCanvas;
    FOverControl: Boolean;
    procedure UpdateForm( ViaMouse, InFocus: Boolean ); virtual;
    procedure RepaintForm; virtual;
    procedure ClearFileInfoList; virtual;
    procedure DblClick; override;
    procedure ReadFileNames; override;
    procedure LocalSetDirectory( const NewDirectory: string );
    function LocalGetFileName: string;
    procedure LocalSetFileName( const NewFile: string );
    function Compare( A, B: TReingnFileInfo ): Integer; virtual;
    procedure QuickSort( L, R: Integer ); virtual;
    procedure DrawItem( Index: Integer; Rect: TRect;
                        State: TOwnerDrawState ); override;
    procedure MouseEnter; dynamic;
    procedure MouseLeave; dynamic;
    function DoMouseWheelDown( Shift: TShiftState;
    function GetColor: TColor; virtual;
    procedure SetColor( Value: TColor ); virtual;
    function NotUsingController: Boolean;
    procedure SetFormColor( Value: TColor ); virtual;
    procedure SetFormController( Value: TReingnFormController ); virtual;
    procedure SetFormFlat( Value: Boolean ); virtual;
    procedure SetFormFlatStyle( Value: TFormStyle ); virtual;
    procedure SetFormFocusStyle( Value: TFormStyle ); virtual;
    procedure SetFormSides( Value: TSides ); virtual;
    procedure SetFormStyle( Value: TFormStyle ); virtual;
    procedure SetFormVisible( Value: Boolean ); virtual;
```

```
  function GetShowGlyphs: Boolean; virtual;
  procedure SetShowGlyphs( Value: Boolean ); virtual;
  procedure SetShowLongNames( Value: Boolean ); virtual;
  function GetLongFileName: string; virtual;
  function GetShortFileName: string; virtual;
end;
PFolderInfo = ^TFolderInfo;
TFolderInfo = record
  FullPath: string;
  ProcessedChildren: Boolean;
end;
TReingnDirectoryTree = class( TReingnCustomTreeView )
private
  FAboutInfo: TReingnAboutInfo;
  FFileList: TReingnTableBox;
  FDirLabel: TLabel;
  FShowHiddenDirs: Boolean;
  FOpenCurrentDir: Boolean;
  FObjInst: Pointer;
  FOldWndProc: TFarProc;
  FFormHandle: HWnd;
  FSaveDirectory: string;
  FUpdating: Boolean;
  FImages: TImageList;
  FFolderOpenIconIndex: Integer;
  FFolderClosedIconIndex: Integer;
  FOnDeletion: TTVExpandedEvent;
  procedure ClearTree; virtual;
  function CanExpand( Node: TTreeNode ): Boolean; override;
  procedure ResetNode( Node: TTreeNode );virtual;
  procedure ProcessChildren( var Node: TTreeNode );
  function HaveProcessedChildren( Node: TTreeNode ): Boolean;
  procedure AddTempNodeIfHasChildren( var Node: TTreeNode );
  procedure Delete( Node: TTreeNode ); override;
  procedure Change( Node: TTreeNode ); override;
  procedure Click; override;
  procedure KeyDown( var Key: Word; Shift: TShiftState ); override;
  function GetDirectory: string; virtual;
  procedure SetDirectory( const Value: string ); virtual;
  procedure SeTDTypes( Value: tDTypes ); virtual;
  procedure SetFileList( Value: TReingnFileListBox ); virtual;
  procedure SetDirLabel( Value: TLabel ); virtual;
  procedure SetDirLabelCaption; virtual;
  procedure SetShowHiddenDirs( Value: Boolean ); virtual;
  property Items stored False;
public
  constructor Create( AOwner: TComponent ); override;
  destructor Destroy; override;
  procedure RefreshTree; virtual;
  function NodeHasData( Node: TTreeNode ): Boolean;
  function GetNodeFromPath( Path: string ): TTreeNode;
  function GetPathFromNode( Node: TTreeNode ): string;
  property Directory: string
    read GetDirectory
    write SetDirectory;
published
  property About: TReingnAboutInfo
    read FAboutInfo
    write FAboutInfo
    stored False;
  property FileList: TReingnTableBox
    read FFileList
    write SetFileList;
  property OpenCurrentDir: Boolean
    read FOpenCurrentDir
    write FOpenCurrentDir
    default False;
  property ShowHiddenDirs: Boolean
    read FShowHiddenDirs
```

```
    write SetShowHiddenDirs
    default False;
end;
TReingnDirectoryListBox = class( TDirectoryListBox )
private
  FAboutInfo: TReingnAboutInfo;
  FFormColor: TColor;
  FFormController: TReingnFormController;
  FFormFlat: Boolean;
  FFormFlatStyle: TFormStyle;
  FFormFocusStyle: TFormStyle;
  FFormSides: TSides;
  FFormStyle: TFormStyle;
  FFormVisible: Boolean;
  FUseFormController: Boolean;
  FShowLongNames: Boolean;
  FOnMouseEnter: TNotifyEvent;
  FOnMouseLeave: TNotifyEvent;
  { Message Handling Methods }
  FCanvas: TCanvas;
  FOverControl: Boolean;
  procedure MouseEnter; dynamic;
  procedure MouseLeave; dynamic;
  procedure SetFormColor( Value: TColor ); virtual;
  procedure SetFormFlat( Value: Boolean ); virtual;
  procedure SetFormFlatStyle( Value: TFormStyle ); virtual;
  procedure SetFormFocusStyle( Value: TFormStyle ); virtual;
  procedure SetFormSides( Value: TSides ); virtual;
  procedure SetFormStyle( Value: TFormStyle ); virtual;
  procedure SetFormVisible( Value: Boolean ); virtual;
  procedure SetShowLongNames( Value: Boolean );
public
  constructor Create( AOwner: TComponent ); override;
  destructor Destroy; override;
  property LongDirName: string
    read GetLongDirName;
published
  property About: TReingnAboutInfo
    read FAboutInfo
    write FAboutInfo
    stored False;
  property OnMouseWheelUp;
  property OnMouseWheelDown;
end;
TReingnComboBox = class( TComboBox )
private
  FAboutInfo: TReingnAboutInfo;
  FTypes: tDTypes;
  FFlatButtons: Boolean;
  FFormColor: TColor;
  FFormController: TReingnFormController;
  FFormFlat: Boolean;
  FFormFlatStyle: TFormStyle;
  FFormFocusStyle: TFormStyle;
  FFormSides: TSides;
  FFormStyle: TFormStyle;
  FFormVisible: Boolean;
  FUseFormController: Boolean;
  FOnMouseEnter: TNotifyEvent;
  FOnMouseLeave: TNotifyEvent;
protected
  FCanvas: TCanvas;
  FInControl: Boolean;
  FOverControl: Boolean;
  procedure ReadNewBitmaps;
  procedure BuildList; override;
  procedure ResetItemHeight;
  procedure MouseEnter; dynamic;
  procedure MouseLeave; dynamic;
```

```
    procedure SetFlatButtons( Value: Boolean ); virtual;
    function NotUsingController: Boolean;
    procedure SetFormFlat( Value: Boolean ); virtual;
    procedure SetFormFlatStyle( Value: TFormStyle ); virtual;
    procedure SetFormFocusStyle( Value: TFormStyle ); virtual;
    procedure SetFormSides( Value: TSides ); virtual;
    procedure SetFormStyle( Value: TFormStyle ); virtual;
    procedure SetFormVisible( Value: Boolean ); virtual;
    procedure SetDTypes( Value: tDTypes );
  public
    constructor Create( AOwner: TComponent ); override;
    destructor Destroy; override;
  published
    property About: TReingnAboutInfo
      read FAboutInfo
      write FAboutInfo
      stored False;
    property FlatButtons: Boolean
      read FFlatButtons
      write SetFlatButtons
      stored NotUsingController
      default True;
  end;
implementation
uses
  Dialogs, RzFilBmp, SysUtils, Forms, RzStrTbl, RzLFName;
var
  FIconCache: TStringList;
constructor TReingnTableBox.Create( AOwner: TComponent );
begin
  inherited Create( AOwner );
  FFileInfoList := TStringList.Create;
  FAllowOpen := False;
  Sorted := False;
  FShowLongNames := True;
  FShowGlyphs := True;
  ResetItemHeight;
  FCanvas := TControlCanvas.Create;
  TControlCanvas( FCanvas ).Control := Self;
  FFormColor := clBtnShadow;
  FFormController := nil;
  FFormFlat := False;
  FFormFlatStyle := fsStatus;
  FFormFocusStyle := fsLowered;
  FFormSides := sdAllSides;
  FFormStyle := fsFlat;
  FFormVisible := False;
  FUseFormController := True;
  FInReadFileNames := False;
end;
destructor TReingnTableBox.Destroy;
begin
  ClearFileInfoList;
  FFileInfoList.Free;
  if FFormController <> nil then
    FFormController.RemoveControl( Self );
  FCanvas.Free;
  inherited Destroy;
end;

procedure TReingnTableBox.ClearFileInfoList;
var
  I: Integer;
  FileExt: string;
begin
  for I := 0 to FFileInfoList.Count - 1 do
  begin
    begin
```

```
DestroyIcon(TReingnFileInfo(FFileInfoList.Objects[I]).
            IconHandle);
    end;
    FFileInfoList.Objects[ I ].Free;
  end;
  FFileInfoList.Clear;
end;

 procedure TReingnTableBox.SetShowLongNames( Value: Boolean );
begin
  if FShowLongNames <> Value then
  begin
    FShowLongNames := Value;
    ReadFileNames;
  end;
end;

 procedure TReingnTableBox.UpOneLevel;
begin
  Items.BeginUpdate;
  try
    Directory := '..';
  finally
    Items.EndUpdate;
  end;
end;

 procedure TReingnTableBox.LocalSetDirectory( const NewDirectory: string );
begin
  if AnsiCompareFileName( NewDirectory, FDirectory ) <> 0 then
  begin
    SetCurrentDir( NewDirectory + '\' );
    FDirectory := GetCurrentDir;
    ReadFileNames;
  end;
end;
function TReingnTableBox.LocalGetFileName: string;
var
  Idx: Integer;
begin
  Idx  := ItemIndex;
  if ( idx < 0 ) or ( Items.Count = 0 ) or ( Selected[ Idx ] = False ) then
    Result := ''
  else
    Result := Items[ Idx ];
end;

 procedure TReingnTableBox.LocalSetFileName( const NewFile: string );
begin
  if AnsiCompareFileName( NewFile, LocalGetFileName ) <> 0 then
  begin
    ItemIndex := SendMessage( Handle, LB_FindStringExact, 0,
                              Longint( PChar( NewFile ) ) );
    Change;
  end;
end;

function TReingnTableBox.Compare( A, B: TReingnFileInfo ): Integer;
begin
  if A.IsDirectory = B.IsDirectory then
    Result := AnsiCompareText( A.Name, B.Name )
  else if A.IsDirectory then
    Result := -1
  else
    Result := 1;
end;

 procedure TReingnTableBox.CNDrawItem( var Msg: TWMDrawItem );
begin
```

```
  if FShowGlyphs then
  begin
    with Msg.DrawItemStruct^ do
      rcItem.Left := rcItem.Left + 24;
  end;
  inherited;
end;

 procedure TReingnTableBox.SetShowGlyphs( Value: Boolean );
begin
  if FShowGlyphs <> Value then
  begin
    FShowGlyphs := Value;
    ResetItemHeight;
    if FShowGlyphs then
      ReadFileNames;
    Invalidate;
  end;
end;

 procedure TReingnTableBox.ResetItemHeight;
var
  H: Integer;
begin
  H := GetMinFontHeight( Font ) - 3;
  if FShowGlyphs then
  begin
    if H < 18 then
      H := 18;
  end;
  ItemHeight := H;
end;

 procedure TReingnTableBox.CMFontChanged( var Msg: TMessage );
begin
  inherited;
  ResetItemHeight;
  RecreateWnd;
end;

 procedure TReingnTableBox.WMWindowPosChanging( var Msg: TWMWindowPosChanging
);
begin
  if ( Columns > 0 ) and ( Msg.WindowPos.cx < 3 ) then
    Msg.WindowPos.cx := 3;
  inherited;
end;
function TReingnTableBox.GetColor: TColor;
begin
  Result := inherited Color;
end;

 procedure TReingnTableBox.SetColor( Value: TColor );
begin
  if Color <> Value then
  begin
    inherited Color := Value;
    if FFormVisible then
      RepaintForm;
  end;
end;
function TReingnTableBox.NotUsingController: Boolean;
begin
  Result := ( FFormController = nil ) or not FUseFormController;
end;

 procedure TReingnTableBox.SetFormColor( Value: TColor );
begin
  if FFormColor <> Value then
```

```
  begin
    FFormColor := Value;
    RepaintForm;
  end;
end;

 procedure TReingnTableBox.SetFormController( Value: TReingnFormController );
begin
  if FFormController <> nil then
    FFormController.RemoveControl( Self );
  FFormController := Value;
  if Value <> nil then
  begin
    Value.AddControl( Self );
    Value.FreeNotification( Self );
  end;
end;

 procedure TReingnTableBox.SetFormFlat( Value: Boolean );
begin
  if FFormFlat <> Value then
  begin
    FFormFlat := Value;
    if FFormFlat then
    begin
      FormVisible := True;
      if not ( csLoading in ComponentState ) then
      begin
        FFormSides := sdAllSides;
        FFormStyle := FFormFlatStyle;
      end;
    end;
    RepaintForm;
    Invalidate;
  end;
end;

 procedure TReingnTableBox.SetFormFlatStyle( Value: TFormStyle );
begin
  if FFormFlatStyle <> Value then
  begin
    FFormFlatStyle := Value;
    RepaintForm;
  end;
end;

procedure TReingnTableBox.SetFormFocusStyle(Value:TFormStyle);
begin
  if FFormFocusStyle <> Value then
  begin
    FFormFocusStyle := Value;
    RepaintForm;
  end;
end;

 procedure TReingnTableBox.SetFormSides( Value: TSides );
begin
  if FFormSides <> Value then
  begin
    FFormSides := Value;
    RepaintForm;
  end;
end;

 procedure TReingnTableBox.SetFormStyle( Value: TFormStyle );
begin
  if FFormStyle <> Value then
  begin
    FFormStyle := Value;
```

```
    RepaintForm;
  end;
end;

 procedure TReingnTableBox.SetFormVisible( Value: Boolean );
begin
  if FFormVisible <> Value then
  begin
    FFormVisible := Value;
    if FFormVisible then
      Ctl3D := True;
    RecreateWnd;
  end;
end;

procedure TReingnTableBox.MouseLeave;
begin
  if Assigned( FOnMouseLeave ) then
    FOnMouseLeave( Self );
end;

 procedure TReingnTableBox.CMMouseLeave( var Msg: TMessage );
begin
  inherited;
  UpdateForm( True, False );
  MouseLeave;
end;

 procedure TReingnTableBox.WMSize( var Msg: TWMSize );
begin
  inherited;
  if FFormVisible then
    RepaintForm;
end;
function TReingnTableBox.DoMouseWheelDown( Shift: TShiftState;
                                           MousePos: TPoint ): Boolean;
var
  Info: TScrollInfo;
begin
  Info.cbSize := SizeOf( Info );
  Info.fMask := sif_Pos;
  GetScrollInfo( Handle, sb_Vert, Info );
  Info.nPos := Info.nPos + Mouse.WheelScrollLines;
  SendMessage( Handle, wm_VScroll, MakeLong( sb_ThumbPosition, Info.nPos ), 0
);
  SetScrollInfo( Handle, sb_Vert, Info, True );
  Result := True;
end;
function TReingnTableBox.DoMouseWheelUp( Shift: TShiftState;
                                         MousePos: TPoint ): Boolean;
var
  Info: TScrollInfo;
begin
  Info.cbSize := SizeOf( Info );
  Info.fMask := sif_Pos;
  GetScrollInfo( Handle, sb_Vert, Info );
  Info.nPos := Info.nPos - Mouse.WheelScrollLines;
  if Info.nPos >= 0 then
  begin
    SendMessage( Handle, wm_VScroll, MakeLong( sb_ThumbPosition, Info.nPos ),
0 );
    SetScrollInfo( Handle, sb_Vert, Info, True );
  end;
  Result := True;
end;
constructor TReingnDirectoryTree.Create( AOwner: TComponent );
begin
  inherited Create( AOwner );
  ReadOnly := True;
```

```
  Width := 250;
  Height := 150;
  FObjInst := nil;
  FOldWndProc := nil;
  HideSelection := False;
  FSaveDirectory := '';
  FUpdating := False;
  OnEditing := EditingHandler;
  OnEdited := EditedHandler;
  inherited OnDeletion := DeleteItemHandler;
  FImages := TImageList.Create( Self );
  Images := FImages;
  FShowHiddenDirs := False;
  FOpenCurrentDir := False;
end;

 procedure TReingnDirectoryTree.CreateWindowHandle( const Params:
TCreateParams );
begin
  inherited CreateWindowHandle( Params );
  if not ( csDesigning in ComponentState ) and ( GetParentForm( Self ) <> nil
) then
  begin
    FFormHandle := ValidParentForm( Self ).Handle;
    FObjInst := Classes.MakeObjectInstance( FormWndProc );
    FOldWndProc := TFarProc( SetWindowLong( FFormHandle, gwl_WndProc,
Longint( FObjInst ) ) );
  end;
end;

 procedure TReingnDirectoryTree.CreateWnd;
begin
  inherited CreateWnd;
  InitImageList;
  InitView;
  if not ( csDesigning in ComponentState ) then
  begin
    if FSaveDirectory <> '' then
    begin
      if not FRecreating then
        SetDirectory( FSaveDirectory );
      FSaveDirectory := '';
    end
    else
      SetDirectory( GetCurrentRootDir );
  end;
end;

 procedure TReingnDirectoryTree.InitImageList;
var
  DirInfo: TSHFileInfo;
begin
  FImages.Handle := SHGetFileInfo( '', 0, DirInfo, SizeOf( DirInfo ),
                                   shgfi_SysIconIndex or shgfi_SmallIcon or
shgfi_Icon );
  FImages.ShareImages := True;
  FFolderClosedIconIndex := DirInfo.iIcon;
  SHGetFileInfo( '', 0, DirInfo, SizeOf( DirInfo ),
                 shgfi_OpenIcon or shgfi_SysIconIndex or shgfi_SmallIcon or
shgfi_Icon );
  FFolderOpenIconIndex := DirInfo.iIcon;
end;
destructor TReingnDirectoryTree.Destroy;
begin
  ClearTree;
  FImages.Free;
  inherited Destroy;
end;
```

```
 procedure TReingnDirectoryTree.DestroyWindowHandle;
begin
  if FFormHandle <> 0 then
  begin
SetWindowLong( FFormHandle, gwl_WndProc, Longint(FOldWndProc) );
    Classes.FreeObjectInstance( FObjInst );
    FOldWndProc := nil;
  end;
  inherited DestroyWindowHandle;
end;

procedure TReingnDirectoryTree.DestroyWnd;
begin
  FSaveDirectory := Directory;
  inherited DestroyWnd;
end;

 procedure TReingnDirectoryTree.Loaded;
begin
  inherited Loaded;
  InitView;
  if not ( csDesigning in ComponentState ) then
  begin
    if FOpenCurrentDir then
      SetDirectory( GetCurrentDir )
    else
      SetDirectory( GetCurrentRootDir );
  end;
end;

 procedure TReingnDirectoryTree.Notification( AComponent: TComponent;
Operation: TOperation );
begin
  inherited Notification( AComponent, Operation );
  if Operation = opRemove then
  begin
    if AComponent = FFileList then
      FFileList := nil
    else if AComponent = FDirLabel then
      FDirLabel := nil;
  end
  else if Operation = opInsert then
  begin
if ( AComponent is TReingnTableBox ) and not Assigned(FFileList) then
      FFileList := TReingnTableBox( AComponent );
  end;
end;
function TReingnDirectoryTree.NodeHasData( Node: TTreeNode ): Boolean;
begin
  Result := ( Node <> nil ) and ( Node.Data <> nil );
end;

procedure TReingnDirectoryTree.AddFolderInfoToNode( Node: TTreeNode; const
NodePath: string;
IconIndex: Integer );
var
  FolderInfo: PFolderInfo;
  FileInfo: TSHFileInfo;
begin
  if Node = nil then
    Exit;
  FolderInfo := New( PFolderInfo );
  FolderInfo^.FullPath := NodePath;
  FolderInfo^.ProcessedChildren := False;
  Node.Data := FolderInfo;
  Node.ImageIndex := IconIndex;
  if IconIndex = FFolderClosedIconIndex then
    Node.SelectedIndex := FFolderOpenIconIndex
  else
```

```
    begin
     SHGetFileInfo( PChar( PFolderInfo( Node.Data )^.FullPath ), 0,
                    FileInfo, SizeOf( TSHFileInfo ),
                    shgfi_SysIconIndex or shgfi_OpenIcon or shgfi_SmallIcon );
      Node.SelectedIndex := FileInfo.iIcon;
    end;
  end;
function TReingnDirectoryTree.GetPathFromNode( Node: TTreeNode ): string;
  begin
    if ( Node <> nil ) and ( Node.Data <> nil ) then
      Result := PFolderInfo( Node.Data ).FullPath
    else
      Result := '';
  end;
  function TReingnDirectoryTree.CanExpand( Node: TTreeNode ): Boolean;
  var
    OldCursor: TCursor;
    ChildNode: TTreeNode;
  begin
    OldCursor := Screen.Cursor;
    Screen.Cursor := crHourGlass;
    try
      ProcessChildren( Node );
      if not ( csDesigning in ComponentState ) and ( Node <> nil ) then
      begin
        ChildNode := Node.GetFirstChild;
        while ChildNode <> nil do
        begin
          AddTempNodeIfHasChildren( ChildNode );
          ChildNode := Node.GetNextChild( ChildNode );
        end;
      end;
    Result := inherited CanExpand( Node );
  end;

  procedure TReingnDirectoryTree.SetShowHiddenDirs(Value:Boolean);
  begin
    if FShowHiddenDirs <> Value then
    begin
      FShowHiddenDirs := Value;
      RefreshTree;
    end;
  end;
  function TReingnDirectoryTree.CanChange( Node: TTreeNode ): Boolean;
  begin
    Result := inherited CanChange( Node );
    FOld := Drive;
  end;

   procedure TReingnDirectoryTree.Delete( Node: TTreeNode );
  begin
    inherited Delete( Node );
    if NodeHasData( Node ) then
    begin
      PFolderInfo( Node.Data )^.FullPath := '';
      Dispose( PFolderInfo( Node.Data ) );
      Node.Data := nil;
    end;
  end;

   procedure TReingnDirectoryTree.DeleteItemHandler( Sender: TObject; Node:
 TTreeNode );
  begin
    if NodeHasData( Node ) then
    begin
      PFolderInfo( Node.Data )^.FullPath := '';
      Dispose( PFolderInfo( Node.Data ) );
      Node.Data := nil;
    end;
```

```
  if Assigned( FOnDeletion ) then
    FOnDeletion( Sender, Node );
end;

 procedure TReingnDirectoryTree.KeyDown( var Key: Word; Shift: TShiftState );
begin
  inherited KeyDown( Key, Shift );
  if Key = vk_F2 then
    Selected.EditText;
end;

 procedure TReingnDirectoryTree.EditingHandler( Sender: TObject; Node:
TTreeNode; var AllowEdit: Boolean );
begin
  if Node = nil then
    AllowEdit := False
  else if Node.Level = 0 then
    AllowEdit := False;
end;

 procedure TReingnDirectoryTree.ResetNode( Node: TTreeNode );
begin
  if Node <> nil then
  begin
    Node.DeleteChildren;
    if NodeHasData( Node ) then
      PFolderInfo( Node.Data )^.ProcessedChildren := False;
    AddTempNodeIfHasChildren( Node );
  end;
end;
function TReingnDirectoryTree.HaveProcessedChildren( Node: TTreeNode ):
Boolean;
begin
  if NodeHasData( Node ) then
    Result := PFolderInfo( Node.Data )^.ProcessedChildren
  else
    Result := False;
end;

 procedure TReingnDirectoryTree.FormWndProc( var Msg: TMessage );
begin
  if Msg.Msg = wm_DeviceChange then
  begin
    Msg.Result := 0;
    UpdateActives;
    Change( Selected );
  end
  else if FOldWndProc <> nil then
Msg.Result := CallWindowProc( FOldWndProc, FFormHandle, Msg.Msg, Msg.WParam,
Msg.LParam );
end;

 procedure TReingnDirectoryTree.UpOneLevel;
begin
  if Selected <> nil then
  begin
    if Selected.Parent <> nil then
      Selected := Selected.Parent;
  end;
end;

 procedure TReingnDirectoryTree.SetDTypes( Value: tDTypes );
begin
  if FTypes <> Value then
  begin
    FTypes := Value;
    UpdateActives;
  end;
end;
```

```
procedure TReingnDirectoryTree.SetFileList(Value:TReingnTableBox);
 begin
   if FFileList <> nil then
     FFileList.FDirTree := nil;
   FFileList := Value;
   if FFileList <> nil then
   begin
     FFileList.FDirTree := Self;
     FFileList.FreeNotification( Self );
   end;
 end;


  procedure TReingnDirectoryTree.SetDirLabel( Value: TLabel );
 begin
   FDirLabel := Value;
   if Value <> nil then
     Value.FreeNotification( Self );
   SetDirLabelCaption;
 end;

  procedure TReingnDirectoryTree.SetDirLabelCaption;
 var
   DirWidth: Integer;
 begin
   if FDirLabel <> nil then
   begin
     DirWidth := Width;
     if not FDirLabel.AutoSize then
       DirWidth := FDirLabel.Width;
     FDirLabel.Caption := MinimizeName( Directory, FDirLabel.Canvas, DirWidth
);
   end;
 end;

 constructor TReingnDirectoryListBox.Create( AOwner: TComponent );
 begin
   inherited Create( AOwner );
   FShowLongNames := True;
   FCanvas := TControlCanvas.Create;
   TControlCanvas( FCanvas ).Control := Self;
   FFormColor := clBtnShadow;
   FFormController := nil;
   FFormFlat := False;
   FFormFlatStyle := fsStatus;
   FFormFocusStyle := fsLowered;
   FFormSides := sdAllSides;
   FFormStyle := fsFlat;
   FFormVisible := False;
   FUseFormController := True;
 end;
 destructor TReingnDirectoryListBox.Destroy;
 begin
   if FFormController <> nil then
     FFormController.RemoveControl( Self );
   FCanvas.Free;
   inherited Destroy;
 end;


  procedure TReingnDirectoryListBox.BuildList;
 var
   D: string;
 begin
   D := Directory;
   while not DirectoryExists( D ) do
     D := ExtractFilePath( D );
   Directory := D;
```

```
  inherited BuildList;
end;


 procedure TReingnDirectoryListBox.SetShowLongNames( Value: Boolean );
begin
  if FShowLongNames <> Value then
  begin
    FShowLongNames := Value;
    BuildList;
    UpdateDirLabel;
  end;
end;
function TReingnDirectoryListBox.GetLongDirName: string;
begin
  Result := LongPathFromShort( Directory );
end;

 procedure TReingnDirectoryListBox.Change;
begin
  inherited Change;
  UpdateDirLabel;
end;
function TReingnDirectoryListBox.GetColor: TColor;
begin
  Result := inherited Color;
end;

 procedure TReingnDirectoryListBox.SetColor( Value: TColor );
begin
  if Color <> Value then
  begin
    inherited Color := Value;
    if FFormVisible then
      RepaintForm;
  end;
end;
function TReingnDirectoryListBox.NotUsingController: Boolean;
begin
  Result := ( FFormController = nil ) or not FUseFormController;
end;

 procedure TReingnDirectoryListBox.SetFormColor( Value: TColor );
begin
  if FFormColor <> Value then
  begin
    FFormColor := Value;
    RepaintForm;
  end;
end;

 procedure TReingnDirectoryListBox.SetFormController( Value:
TReingnFormController );
begin
  if FFormController <> nil then
    FFormController.RemoveControl( Self );
  FFormController := Value;
  if Value <> nil then
  begin
    Value.AddControl( Self );
    Value.FreeNotification( Self );
  end;
end;

 procedure TReingnDirectoryListBox.SetFormFlat( Value: Boolean );
begin
  if FFormFlat <> Value then
  begin
    FFormFlat := Value;
```

```
    if FFormFlat then
    begin
      FormVisible := True;
      if not ( csLoading in ComponentState ) then
      begin
        FFormSides := sdAllSides;
        FFormStyle := FFormFlatStyle;
      end;
    end;
    RepaintForm;
    Invalidate;
  end;
end;

 procedure TReingnDirectoryListBox.SetFormSides( Value: TSides );
begin
  if FFormSides <> Value then
  begin
    FFormSides := Value;
    RepaintForm;
  end;
end;

 procedure TReingnDirectoryListBox.SetFormStyle( Value: TFormStyle );
begin
  if FFormStyle <> Value then
  begin
    FFormStyle := Value;
    RepaintForm;
  end;
end;

procedure TReingnDirectoryListBox.SetFormVisible( Value:Boolean );
begin
  if FFormVisible <> Value then
  begin
    FFormVisible := Value;
    if FFormVisible then
      Ctl3D := True;
    RecreateWnd;
  end;
end;

 procedure TReingnDirectoryListBox.RepaintForm;
var
  R: TRect;
begin
  R := Rect( 0, 0, Width, Height );
  RedrawWindow( Handle, @R, 0, rdw_Invalidate or rdw_UpdateNow or rdw_Form );
end;

 procedure TReingnDirectoryListBox.WMNCPaint( var Msg: TWMNCPaint );
var
  DC: HDC;
begin
  inherited;
  if FFormVisible then
  begin
    DC := GetWindowDC( Handle );
    FCanvas.Handle := DC;
    try
      DrawForm(FCanvas, Width, Height, FFormStyle, Color,
                FFormColor, FFormSides );
    finally
      FCanvas.Handle := 0;
      ReleaseDC( Handle, DC );
    end;
    Msg.Result := 0;
  end;
```

```
end;

 procedure TReingnDirectoryListBox.MouseLeave;
begin
  if Assigned( FOnMouseLeave ) then
    FOnMouseLeave( Self );
end;

 procedure TReingnComboBox.SetDTypes( Value: tDTypes );
begin
  if FTypes <> Value then
  begin
    FTypes := Value;
    RecreateWnd;
  end;
end;

 procedure TReingnDriveComboBox.SetFlatButtons( Value: Boolean );
begin
  if FFlatButtons <> Value then
  begin
    FFlatButtons := Value;
    Invalidate;
  end;
end;
function TReingnDriveComboBox.NotUsingController: Boolean;
begin
  Result := ( FFormController = nil ) or not FUseFormController;
end;

 procedure TReingnDriveComboBox.SetFormColor( Value: TColor );
begin
  if FFormColor <> Value then
  begin
    FFormColor := Value;
    Invalidate;
  end;
end;

 procedure TReingnDriveComboBox.SetFormController( Value:
TReingnFormController );
begin
  if FFormController <> nil then
    FFormController.RemoveControl( Self );
  FFormController := Value;
  if Value <> nil then
  begin
    Value.AddControl( Self );
    Value.FreeNotification( Self );
  end;
end;

 procedure TReingnDriveComboBox.SetFormFlat( Value: Boolean );
begin
  if FFormFlat <> Value then
  begin
    FFormFlat := Value;
    if FFormFlat then
    begin
      FormVisible := True;
      if not ( csLoading in ComponentState ) then
      begin
        FFormSides := sdAllSides;
        FFormStyle := FFormFlatStyle;
      end;
    end;
    Invalidate;
  end;
end;
```

```
procedure TReingnDrive.SetFormSides( Value: TSides );
begin
  if FFormSides <> Value then
  begin
    FFormSides := Value;
    Invalidate;
  end;
end;

procedure TReingnDrive.CMEnter( var Msg: TCMEnter );
begin
  inherited;
  UpdateForm( False, True );
end;

 procedure TReingnDrive.CMExit( var Msg: TCMExit );
begin
  inherited;
  FOverControl := False;
  UpdateForm( False, False );
end;
procedure TReingnDrive.WMSize( var Msg: TWMSize );
begin
  inherited;
  if FFormVisible then
    Invalidate;
end;
end.
```