ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Факультет міжнародних економічних відносин та інформаційних технологій

Кафедра комп'ютерних наук, прикладної та вищої математики

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Автоматизація створення та обробки офісної документації в Microsoft Office з використанням VBA

Виконав: здобувач вищої освіти
першого (бакалаврського) рівня
ОПП "Комп'ютерні науки"
спеціальності 122 "Комп'ютерні науки"

_____ **Балицький Дмитро Андрійович**

(підпис)

Науковий керівник
_____ старший викладач Крутяк М. Б.

(підпис)

Львів – 2026

ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
Факультет міжнародних економічних відносин та інформаційних технологій
Кафедра комп'ютерних наук, прикладної та вищої математики
Перший (бакалаврський) рівень вищої освіти
Спеціальність 122 "Комп'ютерні науки"
Освітньо-професійна програма "Комп'ютерні науки"

“ЗАТВЕРДЖУЮ”

Завідувач кафедри комп'ютерних наук,
прикладної та вищої математики

_____ З. Г. Новосад

13.10.2025 р.

Завдання
до кваліфікаційної роботи
Балицькому Дмитру Андрійовичу
(прізвище, ім’я, по батькові)

1.Тема роботи **Автоматизація створення та обробки офісної документації в Microsoft Office з використанням VBA**

керівник роботи **Крутяк Марія Богданівна, *старший викладач***

затверджена наказом по Університету ***№ 991/03 від 06 жовтня 2025 р.***

2.Термін здачі здобувачем вищої освіти роботи 12 червня 2026 р.
3. Вхідні дані до роботи: література, документація, матеріали та сайти Internet, що пов’язані з автоматизацією обробки офісної документації в Microsoft Office з використанням VBA.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)
   1) Теоретичні основи автоматизації обробки офісної документації;
   2) Аналіз засобів автоматизації в середовищі Microsoft Office. Мова програмування Visual Basic for Applications;
   3) Практична реалізація та структуризація модулів автоматизації в середовищі Microsoft Office.
5. Перелік графічного матеріалу до 12 слайдів у форматі PowerPoint
6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 13.10.2025 р.

Календарний план

№	Найменування етапів виконання кваліфікаційної роботи	Терміни виконання етапів	Примітка
1.	Отримання завдання і аналіз інформаційних джерел за темою роботи	до 11.12.2025	
2.	Дослідження можливостей та методів автоматизації обробки офісної документації	до 13.01.2026	
3.	Аналіз засобів автоматизації в середовищі Microsoft Office	до 07.02.2026	
4.	Мова програмування Visual Basic for Applications	до 01.04.2026	
5.	Практична реалізація модулів автоматизації в середовищі Microsoft Office	до 10.04.2026	
6.	Тестування	до 22.04.2026	
7.	Оформлення кваліфікаційної роботи. Подання роботи на перевірку керівнику	до 16.05.2026	
8.	Доопрацювання кваліфікаційної роботи з врахуванням зауважень керівника. Подання роботи для перевірки на рівень запозичень	до 20.05.2026	
9.	Подання кваліфікаційної роботи керівнику для отримання відгуку	до 27.05.2026	
10.	Представлення кваліфікаційної роботи на зовнішнє рецензування. Подання кваліфікаційної роботи на кафедру комп'ютерних наук	до 5.06.2026	
11.	Попередній захист кваліфікаційної роботи	08.06.2026-09.06.2026	
12.	Захист кваліфікаційної роботи на засіданні екзаменаційної комісії	15.06.2026-19.06.2026 згідно графіку роботи ЕК	

Здобувач вищої освіти _____ Балицький Д. А.

Керівник роботи _____ Крутяк М.Б.

РЕФЕРАТ

Кваліфікаційна робота присвячена дослідженню теоретичних засад, проєктуванню та практичній реалізації комплексної системи автоматизації офісного документообігу в середовищі Microsoft Office з використанням вбудованої мови програмування Visual Basic for Applications (VBA).

Об'єктом дослідження виступають процеси збору, обробки, консолідації первинних даних та формування вихідної текстової і поштової документації на підприємстві. Предметом дослідження є архітектура, об'єктні моделі та програмні інструменти автоматизації рутинних операцій на базі офісних додатків Excel, Word та Outlook.

У першому розділі визначено поняття та значення автоматизації сучасного діловодства, виконано порівняльний аналіз існуючих СЕД, CRM та ERP-систем. Детально досліджено специфіку та обмеження мови VBA. Обґрунтовано вибір об'єктних моделей додатків та описано методи оптимізації обчислень в ОЗП за допомогою масивів типу Variant та відключення графічних процесів.

У другому розділі розроблено загальну архітектуру програмного комплексу. Описано створення інтерфейсу користувача (UserForm) та логіку його взаємодії з реєстрами. Покроково реалізовано три ключові функціональні модулі: модуль імпорту та очищення первинних звітів, модуль динамічної генерації договорів у форматі Word за шаблонами та поштовий підсистемний модуль розсилки готових документів. Особливу увагу приділено побудові системи обробки виняткових ситуацій із веденням файлу системного логу помилок.

У третьому розділі проведено практичні випробування розробленого комплексу макросів на базі реальних тестових даних. Детально описано виявлені під час тестування критичні помилки сумісності та наведено лістинги програмного захисту для їх усунення. Проведено розрахунок операційної ефективності, який підтвердив скорочення часу обробки документації в десятки разів та повну ліквідацію людського фактора.

Загальний обсяг роботи разом з додатками становить 70 сторінок.

ABSTRACT

The qualification work is devoted to the study of theoretical foundations, design and practical implementation of a comprehensive office document workflow automation system in the Microsoft Office environment using the built-in Visual Basic for Applications (VBA) programming language.

The object of the study is the processes of collecting, processing, consolidating primary data and forming initial text and mail documentation at the enterprise. The subject of the study is the architecture, object models and software tools for automating routine operations based on the office applications Excel, Word and Outlook.

The first section defines the concept and meaning of automation of modern office management, a comparative analysis of existing EDS, CRM and ERP systems is performed. The specifics and limitations of the VBA language are studied in detail. The choice of application object models is justified and methods for optimizing calculations in RAM using Variant arrays and disabling graphic processes are described.

The second section develops the general architecture of the software complex. The creation of the user interface (UserForm) and the logic of its interaction with registers are described. Three key functional modules were implemented step by step: the module for importing and cleaning primary reports, the module for dynamically generating contracts in Word format using templates, and the mail subsystem module for sending finished documents. Particular attention was paid to building a system for handling exceptional situations with maintaining a system error log file.

In the third section, practical tests of the developed macro complex were conducted on the basis of real test data. Critical compatibility errors identified during testing are described in detail and software protection listings for their elimination are provided. An operational efficiency calculation was performed, which confirmed a reduction in documentation processing time by dozens of times and the complete elimination of the human factor.

The total volume of the work, including appendices, is 70 pages.

ЗМІСТ

ВСТУП 7
1. ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ ОФІСНОЇ ДОКУМЕНТАЦІЇ ТА АНАЛІЗ СЕРЕДОВИЩА РОЗРОБКИ 9
1.1. Поняття та значення автоматизації офісної документації в сучасних організаціях 9
1.2. Огляд та порівняльний аналіз програмних засобів автоматизації офісу 11
1.3. Технічні особливості та обмеження при виконанні коду VBA 14
1.4. Об'єктна модель та події в офісних додатках 16
1.5. Безпека макросів та проблеми сумісності рішень 18
2. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ ТА АРХІТЕКТУРА МОВИ ПРОГРАМУВАННЯ VISUAL BASIC FOR APPLICATIONS 21
2.1. Історичний контекст, еволюція та місце VBA в сучасних офісних системах 21
2.2. Інтегроване середовище розробки VBE 25
2.3. Синтаксична структура мови VBA, алгоритмічні конструкції 29
2.4. Технології інтеграції COM та OLE: механізми міжпрограмної взаємодії 33
2.5. Практичні обмеження та недоліки використання VBA 35
3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА СТРУКТУРИЗАЦІЯ МОДУЛІВ АВТОМАТИЗАЦІЇ В СЕРЕДОВИЩІ MICROSOFT OFFICE 39
3.1. Архітектурне планування системи та конфігурація файлової структури 39
3.2. Побудова міжпрограмного інтерфейсу COM з MS Word та управління процесами 44
3.3. Інтеграція з об'єктною моделлю MS Outlook та управління поштовим сервером 48
3.4. Проєктування графічного інтерфейсу UserForms та програмування логіки подій 51
3.5. Оптимізація швидкодії, технологія обробка помилок 53
ВИСНОВКИ 56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60
ДОДАТКИ 62

ВСТУП

У сучасних умовах цифровізації діяльності підприємств та організацій значно зростає роль автоматизації процесів створення, обробки та зберігання офісної документації. Ефективне управління документами є важливою складовою забезпечення продуктивності праці, зменшення кількості помилок і оптимізації робочого часу працівників. Особливо актуальним це є для установ, де щоденно виконується великий обсяг однотипних операцій із текстовими документами, електронними таблицями та базами даних.

Програмний пакет Microsoft Office є одним із найпоширеніших інструментів для роботи з офісною документацією. До його складу входять такі програми, як Microsoft Word, Microsoft Excel та Microsoft Access, які широко використовуються для створення, редагування, аналізу та зберігання інформації. Проте виконання багатьох рутинних операцій вручну потребує значних часових витрат і може призводити до виникнення помилок через людський фактор.

Одним із ефективних засобів автоматизації офісних процесів є використання мови програмування Visual Basic for Applications (VBA), яка інтегрована в продукти Microsoft Office. VBA дозволяє створювати макроси, автоматизувати повторювані дії, реалізовувати обробку даних, формувати звіти, генерувати документи та забезпечувати взаємодію між різними компонентами офісного пакета. Завдяки цьому підвищується швидкість виконання завдань, покращується точність обробки інформації та спрощується робота користувачів.

Тема автоматизації створення та обробки офісної документації з використанням VBA є **актуальною**, оскільки сучасні організації прагнуть мінімізувати ручну працю та підвищити ефективність інформаційних процесів. Використання VBA дає змогу створювати індивідуальні програмні рішення, адаптовані до потреб конкретного підприємства чи установи.

Об’єктом дослідження є процес створення та обробки офісної документації.

Предметом дослідження є засоби автоматизації офісних процесів у середовищі Microsoft Office з використанням VBA.

Метою роботи є дослідження можливостей автоматизації офісної документації засобами VBA у середовищі Microsoft Office, а також розробка прикладного рішення для автоматизованого створення та обробки документів.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

- проаналізувати можливості пакета Microsoft Office для автоматизації роботи з документами;
- дослідити основні засоби та інструменти VBA;
- розглянути методи створення макросів і автоматизації типових операцій;
- реалізувати програмний модуль для автоматизованої обробки офісної документації;
- оцінити ефективність використання VBA для оптимізації роботи з документами.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Основний зміст викладено на 63 сторінках, містить 9 рисунків та 1 таблицю..

1. ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ ОФІСНОЇ ДОКУМЕНТАЦІЇ ТА АНАЛІЗ СЕРЕДОВИЩА РОЗРОБКИ

1.1. Поняття та значення автоматизації офісної документації в сучасних організаціях

У сучасному інформаційному суспільстві офісна документація є головним інструментом ведення бізнес-процесів на будь-якому підприємстві. Будь-яка управлінська дія, фінансова транзакція чи кадрова зміна обов’язково супроводжуються створенням, погодженням та збереженням відповідних документів. Відповідно до визначених стандартів діловодства (зокрема, ДСТУ 4163:2020), документ не лише фіксує інформацію, а й надає їй юридичної сили, що є обов’язковим для взаємодії з державними органами та зовнішніми контрагентами.

У діяльності сучасних організацій найбільший обсяг документообігу припадає на такі категорії документів:

- **Організаційно-розпорядчі:** накази по підприємству, розпорядження керівництва, протоколи виробничих нарад, службові записки.
- **Договірні та комерційні:** договори купівлі-продажу, угоди про надання послуг, технічні специфікації, акти приймання-передачі виконаних робіт.
- **Первинні облікові:** рахунки-фактури, видаткові та товарно-транспортні накладні, акти звірки взаєморозрахунків.
- **Звітні матеріали:** внутрішні звіти підрозділів, аналітичні довідки, зведені таблиці фінансових показників.

При традиційній організації офісної роботи значна частина операцій з цими документами виконується працівниками вручну. Оператор змушений самостійно відкривати файли, що надходять від різних відділів, копіювати звідти текстові реквізити та числові показники, вносити їх до підсумкових таблиць, а також вручну заповнювати типові форми договорів у текстових редакторах. Традиційні методи роботи з документацією, які передбачають ручне введення даних, копіювання інформації та повторення однотипних операцій, потребують

значних витрат часу та можуть призводити до виникнення помилок через людський фактор. Саме тому все більшого значення набуває автоматизація офісної документації.

При виконанні однотипних механічних дій (копіювання та вставка даних) увага працівника з часом знижується. За статистичними даними, при тривалій роботі з таблицями оператор допускає помилки в середньому у 1–2% випадків. У фінансовій звітності чи юридичних договорах навіть одна неправильна цифра в коді ЄДРПОУ, номері рахунку чи сумі контракту може призвести до блокування платежів, податкових штрафів та судових спорів. Окрім того, використання кваліфікованого персоналу для виконання технічних дій копіювання є економічно неефективним.

Рішенням цієї проблеми є впровадження засобів автоматизації. Автоматизація офісної документації — це процес використання програмних засобів і технологій для спрощення, прискорення та оптимізації створення, обробки, редагування, зберігання й передачі документів. Основною метою автоматизації є підвищення ефективності роботи працівників, зменшення кількості помилок і скорочення часу виконання рутинних операцій. При цьому за людиною залишається лише функція інтелектуального контролю: перевірка кінцевого результату та ухвалення рішень

Основними перевагами автоматизації діловодства є:

Скорочення часових витрат: програмний алгоритм здатний обробити десятки файлів та згенерувати пакет документів за кілька секунд, на що у людини пішло б кілька годин.

Гарантія точності: автоматичне перенесення даних з первинних звітів виключає появу помилок у назвах компаній, кодах чи фінансових сумах.

Стандартизація обліку: використання жорстких програмних шаблонів гарантує, що всі вихідні документи будуть оформлені строго за чинними стандартами підприємства.

Прозорість процесів: система дозволяє чітко бачити етапи обробки даних за допомогою автоматичного присвоєння статусів (наприклад: "Очікує", “Сформовано”, “Надіслано”).

Таким чином, автоматизація рутинних процесів є необхідною умовою для підвищення продуктивності праці адміністративного персоналу та зниження операційних витрат підприємства.

Автоматизація офісної документації є важливим напрямом підвищення ефективності роботи сучасних організацій. Використання засобів автоматизації, зокрема VBA, дозволяє оптимізувати робочі процеси, покращити якість обробки інформації та забезпечити більш раціональне використання часу й ресурсів.

1.2. Огляд та порівняльний аналіз програмних засобів автоматизації офісу

Сучасний ринок програмного забезпечення пропонує розгалужений спектр рішень для автоматизації роботи з документами. Ці системи відрізняються за вартістю, складністю розгортання, гнучкістю налаштування під конкретні завдання та вимогами до комп'ютерної інфраструктури.

У практиці менеджменту всі засоби автоматизації прийнято ділити на кілька основних класів:

1. Системи електронного документообігу (СЕД)

Це масштабні корпоративні платформи (наприклад, Megapolis.DocNet, Deals, Documentov), які створюються для повного супроводу життєвого циклу документів у великих компаніях. Вони автоматизують процеси довгострокового архівного зберігання, маршрутизацію файлів між сотнями користувачів та внутрішнє багаторівневе погодження наказів чи рішень. СЕД активно використовують електронний цифровий підпис (ЕЦП).

Проте впровадження таких систем потребує значних фінансових витрат на купівлю ліцензій та оплату послуг сторонніх інтеграторів. Окрім того, ці системи є досить жорсткими: будь-яка зміна у формі звіту чи логіці обробки вимагає перепрограмування системи фахівцями компанії-розробника [4].

2. CRM та ERP-системи

Платформи управління відносинами з клієнтами (CRM) та планування ресурсів підприємства (ERP), такі як HubSpot, Salesforce або Odoo, мають вбудовані модулі для створення документів. Вони автоматично генерують рахунки, комерційні пропозиції чи акти безпосередньо з картки клієнта. Це ефективно для відділів продажів, але розгортання повноцінної ERP-системи є занадто складним та дорогим завданням, якщо підприємству потрібно просто автоматизувати локальну роботу з первинною звітністю всередині одного конкретного відділу.

3. Офісні програмні пакети

Це стандартні програми для роботи з текстами та таблицями, які встановлені майже на кожному робочому місці (Microsoft Office, Google Workspace, LibreOffice). Пакет Google Workspace орієнтований на роботу в хмарі через браузер, але його вбудована мова макросів (Google Apps Script) потребує постійного інтернет-з'єднання і має обмеження по часу виконання скриптів. Безкоштовний пакет LibreOffice використовує мову StarBasic, яка має складний синтаксис та малу кількість навчальної документації.

Важливу роль у автоматизації офісних процесів відіграє пакет Microsoft Office, який містить широкий набір інструментів для роботи з різними типами документів. Зокрема, Microsoft Word використовується для створення та редагування текстових документів, Microsoft Excel — для обробки табличних даних і виконання розрахунків, а Microsoft Access — для організації та зберігання великих обсягів інформації.

Одним із найбільш ефективних засобів автоматизації в середовищі Microsoft Office є Visual Basic for Applications — мова програмування, інтегрована в офісні додатки Microsoft. VBA дозволяє створювати макроси та програмні модулі для автоматизації рутинних дій користувача. За допомогою VBA можна автоматизувати введення даних, форматування документів, обробку таблиць, створення звітів та взаємодію між різними компонентами Microsoft Office.

Пакет Microsoft Office залишається найбільш зручним середовищем для створення індивідуальних локальних рішень завдяки наявності вбудованої мови програмування Visual Basic for Applications (VBA).

Для обґрунтування вибору платформи розробки проведемо порівняльний аналіз цих систем автоматизації за ключовими для підприємства критеріями (таблиця 1.1):

Таблиця 1.1. – Порівняльний аналіз систем автоматизації

Критерій порівняння	**Системи електронного документообігу (СЕД)**	**CRM / ERP системи**	**Локальні макроси в MS Office (VBA)**
Цільове призначення	Погодження, контроль виконання та архівування документів компанії.	Комплексне управління продажами, клієнтами та логістикою.	Автоматизація локальних завдань, збір даних, робота з шаблонами.
Вартість розгортання	Висока (оплата ліцензій за кожного користувача).	Дуже висока (тривалий процес налаштування та консалтингу).	Мінімальна(використовується вже встановлений на ПК пакет програм).
Швидкість адаптації коду	Низька (потребує офіційного звернення до розробника софту).	Середня (потребує зміни внутрішніх конфігурацій платформи).	Максимальна (код макросу можна миттєво змінити під нову форму звіту).
Автономність роботи	Низька (залежить від доступності сервера чи хмари).	Низька (без інтернету доступ до бази даних блокується).	Повна (програма працює локально на жорсткому диску комп'ютера).
Складність розробки	Потребує сертифікованих спеціалістів з конкретної СЕД.	Потребує залучення розробників під конкретну ERP.	Код може бути написаний і підтримуватися внутрішнім IT-фахівцем.

Порівняльний аналіз показує, що для реалізації наскрізних локальних завдань — коли потрібно зібрати дані з багатьох файлів Excel, очистити їх, згенерувати індивідуальні договори у Word та підготувати їх до відправки — використання вбудованої мови VBA є найбільш раціональним рішенням. Його

впровадження не несе додаткових фінансових ризиків для компанії, а висока гнучкість мови дозволяє швидко змінювати алгоритми при зміні внутрішніх звітних форм підприємства.

1.3. Технічні особливості та обмеження при виконанні коду VBA

Visual Basic for Applications (VBA) — це об’єктно-базована мова програмування, яка повністю інтегрована в додатки Microsoft Office (Excel, Word, Access, Outlook). Головна архітектурна особливість VBA полягає в тому, що вона є вбудованою мовою. Розробник не може створити за її допомогою окрему незалежну програму у вигляді файлу з розширенням .exe. Весь програмний код, створені інтерфейсні форми та модулі зберігаються безпосередньо всередині самого документа офісної програми (наприклад, у книзі Excel з розширенням .xlsm).

Коли користувач запускає макрос, текст програми спочатку перевіряється вбудованим компілятором на наявність очевидних синтаксичних помилок і переводиться у внутрішній бінарний стан (так званий код інструкцій), який під час роботи покроково виконується спеціальним двигуном всередині Office. Така модель забезпечує високу портативність: файл із кодом можна перенести на будь-який комп'ютер, де встановлено Excel, і він працюватиме без встановлення додаткових програм чи фреймворків.

Проте інтегрована природа VBA накладає два суттєвих технічних обмеження, які розробник обов'язково повинен враховувати при створенні систем автоматизації:

Залежність від швидкості інтерфейсу додатків: Пряма обробка даних у клітинках аркуша Excel є дуже повільною операцією. Кожне зчитування чи запис значення через команду типу Cells(i, j).Value змушує програму звертатися до графічної оболонки та перемальовувати екран. Якщо макрос обробляє тисячі рядків, це призводить до тривалого миготіння монітора та суттєвого уповільнення роботи.

Строга однопотоковість виконання коду: Макрос VBA виконується послідовно в тому самому єдиному потоці операційної системи, в якому працює інтерфейс самої програми Excel. Мова VBA не має вбудованих інструментів для розподілу завдань на кілька ядер процесора чи створення паралельних потоків обчислень.

Через однопотоковість виникає відома проблема: якщо макрос запускає довгий цикл обробки даних (наприклад, послідовний перебір і відкриття 50 файлів звітів у папці), графічна оболонка Excel повністю застигає. Програма перестає реагувати на кліки мишки, вікно покривається білим шаром, а операційна система Windows присвоює процесу статус “Не відповідає”. Користувач не може згорнути таблицю або зупинити процес, що створює враження повного зависання комп'ютера.

У професійній практиці розробки автоматизованих офісних сценаріїв ці проблеми вирішуються двома основними технічними прийомами:

Оптимізація роботи з пам'яттю через масиви: Замість того, щоб у циклі тисячі разів читати клітинки на аркуші, код макросу одним оператором копіює весь необхідний табличний діапазон в оперативну пам'ять комп'ютера у вигляді динамічного масиву типу Variant (наприклад, DataArray = Range("A4:F100").Value). Усі подальші перевірки, видалення пробілів та аналіз даних відбуваються всередині цього масиву в ОЗП. Оскільки швидкість доступу до оперативної пам'яті в тисячі разів вища, ніж швидкість перемальовування графічного інтерфейсу Excel, час виконання макросу знижується з кількох хвилин до часток секунди, а інтерфейс програми не встигає заблокуватися.

Керування налаштуваннями додатка (Application): На початку роботи макросу розробник може примусово вимкнути графічні процеси Excel за допомогою команд Application.ScreenUpdating = False (вимикає оновлення екрана) та Application.Calculation = xlCalculationManual (вимикає автоматичний перерахунок формул у клітинках при внесенні кожного нового значення). Після завершення обробки даних ці параметри обов’язково повертаються в початковий

стан (True та Automatic). Це дозволяє повністю усунути миготіння екрана та економить ресурси процесора.

1.4. Об'єктна модель та події в офісних додатках

Уся логіка побудови макросів у VBA базується на взаємодії з об'єктами. Мова VBA відноситься до класу об'єктно-базованих мов. У ній є поняття класів, об'єктів, методів та властивостей, проте відсутнє класичне спадкування. Розробник не може створити свій новий тип об'єкта на основі вже існуючого (наприклад, створити модифікований аркуш на основі стандартного Worksheet). Попри це, існуючого набору стандартних об'єктів повністю достатньо для вирішення будь-яких завдань автоматизації документообігу [6].

Кожен додаток Microsoft Office має свою сувору внутрішню ієрархію об'єктів — так звану об'єктну модель. Об'єкти вищого рівня містять у собі об'єкти нижчого рівня.

В об'єктній моделі Microsoft Excel корінням є об'єкт Application (сама програма Excel). Всередині нього знаходиться колекція відкритих книг Workbooks, де кожна книга є окремим об'єктом Workbook (конкретний файл). Кожен файл містить колекцію аркушів Worksheets (об'єкт Worksheet), а на найнижчому рівні розташований об'єкт Range, який представляє собою клітинку або діапазон комірок.

Кожен об'єкт має дві головні складові:

- **Властивості (Properties):** Це дані, які описують характеристики об'єкта. Наприклад, для об'єкта Rangeвластивостями є його значення (.Value), шрифт (.Font.Name), колір клітинки (.Interior.Color). Зчитуючи або змінюючи ці властивості, макрос керує вмістом таблиці.
- **Методи (Methods):** Це вбудовані функції, які об'єкт вміє виконувати за командою. Наприклад, метод .Openвідкриває книгу, метод .Copy копіює дані в буфер, а метод .SaveAs2 зберігає файл на диск.

В основі автоматизації лежить подійно-орієнтована модель. Код макросу не просто виконується від початку до кінця — він може бути прив'язаний до конкретних подій (Events), які відбуваються всередині офісного додатка. Це дозволяє створювати системи, що реагують на дії користувача без його прямого втручання.

До основних подій відносяться:

Workbook_Open — подія, яка автоматично запускає макрос у момент відкриття файлу Excel користувачем. Це ідеальне місце для автоматичного зчитування конфігурацій системи, перевірки наявності потрібних папок на диску та виведення графічних форм на екран.

Worksheet_Change — подія аркуша, яка миттєво фіксує ручне введення або зміну даних користувачем у будь-якій комірці таблиці, що дозволяє відразу перевіряти коректність форматів.

Для захисту від логічних помилок на початку кожного модуля обов’язково прописується команда Option Explicit. Вона переводить компілятор у режим суворого контролю змінних, повністю забороняючи запуск макросу, якщо в тексті виявлено хоча б одну змінну, яка не була офіційно оголошена розробником за допомогою оператора Dim із зазначенням точного типу даних. Це повністю виключає помилки, пов’язані з випадковими друкарськими помилками в назвах лічильників чи накопичувачів даних.

Для стабільної роботи алгоритмів велике значення має правильний вибір типів даних:

- **Long** — велике ціле число (до 2 мільярдів). У сучасних системах автоматизації Excel тип Long є стандартом для оголошення номерів рядків та лічильників циклів. Його використання повністю захищає програму від критичної помилки переповнення пам'яті (*Overflow*), яка часто виникала в застарілому типі Integer, що мав обмеження всього до 32 767 рядків.
- **String** — текстовий тип даних, необхідний для збереження назв контрагентів, email-адрес та формування HTML-тексту листів.

- **Currency** — спеціалізований фінансовий тип даних із фіксованою точкою (строго 4 знаки після коми). Його використання є категорично обов'язковим для грошових розрахунків.

1.5. Безпека макросів та проблеми сумісності рішень

При впровадженні систем автоматизації на базі мови VBA на перший план виходять питання інформаційної безпеки. Оскільки код VBA має прямий доступ до файлової системи комп'ютера через COM-об'єкти (наприклад, може створювати, перейменовувати або видаляти файли на жорсткому диску за допомогою об'єкта Scripting.FileSystemObject), офісні документи історично стали одним із головних каналів поширення шкідливого програмного забезпечення — макровірусів. Зловмисники ховали небезпечні скрипти у файли рахунків чи звітів, які масово розсилалися через електронну пошту.

Для захисту користувачів корпорація Microsoft розробила суворий багатошаровий периметр безпеки, який кожен розробник макросів повинен враховувати при розгортанні своїх рішень:

Розділення форматів файлів за розширенням: Звичайні базові файли Excel (.xlsx) та Word (.docx) фізично не здатні містити в собі програмний код. Якщо користувач напише макрос і спробує зберегти його в форматі .xlsx, система видасть жорстке попередження і при записі повністю видалить весь код без можливості відновлення. Для документів, які легально містять макроси, виділено спеціальні розширення — **.xlsm** або **.xlsb**. Це дозволяє антивірусному софту підприємства миттєво ідентифікувати та перевіряти потенційно небезпечні файли ще на етапі їхнього надходження в мережу компанії.

Центр керування безпекою (Trust Center): За замовчуванням у налаштуваннях офісного пакету активовано режим “Блокувати всі макроси зі сповіщенням”. При відкритті файлу формату .xlsm віртуальна машина VBA ізолюється, а користувач бачить жовту смугу-попередження. Код не почне виконуватися доти, доки користувач свідомо не натисне кнопку **“Увімкнути вміст” (Enable Content)**.

Захист від завантажень з інтернету (Mark of the Web — MOTW): Якщо файл з макросами завантажено з мережі інтернет, отримано через зовнішню пошту чи месенджери, операційна система Windows автоматично блокує його виконання за допомогою прихованої системної мітки. При відкритті такого документа офіс виводить червоний баннер безпеки, який повністю блокує макроси без можливості їх швидкої активації. Щоб запустити такий файл, користувач повинен спочатку закрити його, відкрити його *Властивості* в провіднику Windows, поставити галочку “Розблокувати” (Unblock) на вкладці *Загальні* і лише після цього відкрити файл знову.

Для безперебійної роботи великих систем автоматизації в межах офісу, щоб уникнути постійних ручних розблокувань, файли програм розміщують у так званих Довірених розташуваннях (Trusted Locations) — спеціальних папках на диску комп’ютера або серверах компанії, документи з яких запускаються миттєво в обхід перевірок безпеки.

Окрім жорстких вимог безпеки, розробка систем на базі VBA стикається з кількома практичними обмеженнями, які важливо враховувати при масштабуванні бізнес-процесів:

- **Повна відсутність кросплатформеності:** Мова VBA створена виключно під архітектуру операційної системи Windows. Макроси взагалі не здатні виконуватися у хмарних веб-версіях програм через браузери (інтерфейс Office Online або системи SharePoint повністю ігнорують код VBA). На операційній системі macOS (комп'ютери Apple Mac) можливості макросів суттєво обмежені: там не підтримуються графічні елементи керування ActiveX та системні виклики Windows API, що призводить до аварійного закриття програми при спробі запустити складний код.
- **Несумісність із системами контролю версій (Git):** Оскільки весь програмний код, дизайн форм та логіка модулів зашиті всередині єдиного великого бінарного файлу Excel, класичні системи контролю версій типу Git не здатні відслідковувати зміни покроково всередині модулів. Для Git файл .xlsm є неподільним закритим об'єктом. Якщо над автоматизацією

одночасно працюють два розробники, автоматично об'єднати їхню роботу (зробити merge) через Git неможливо — файл одного автора просто повністю затре зміни іншого. Це змушує розробників координувати роботу вручну.

- **Проблема підтримки застарілого коду (Legacy Code):** Дуже часто макроси на підприємствах створюються звичайними працівниками (економістами, аналітиками) без базової освіти в галузі програмування. Це призводить до появи заплутаного **“спагеті-коду”** — довгих процедур без коментарів, логічної структури та належної обробки помилок. Коли такий працівник звільняється, а на підприємстві змінюється форма вхідної звітності, макрос повністю ламається. Пошук помилок та рефакторинг такого коду сторонніми фахівцями стає настільки дорогим, що компаніям значно вигідніше повністю списати такий застарілий код та замовити проєктування системи автоматизації з нуля з дотриманням професійних інженерних стандартів розробки [7].

2. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ ТА АРХІТЕКТУРА МОВИ ПРОГРАМУВАННЯ VISUAL BASIC FOR APPLICATIONS

2.1. Історичний контекст, еволюція та місце VBA в сучасних офісних системах

Для того, щоб зрозуміти, чому автоматизація офісної діяльності сьогодні так сильно зав'язана на Visual Basic for Applications (VBA), варто повернутися на кілька десятиліть назад. На початку розвитку комп'ютерних технологій для офісів кожна програма існувала сама по собі. Якщо фірма купувала текстовий редактор і табличний процесор, це були абсолютно різні продукти, які розроблялися різними командами, а іноді й різними компаніями. Відповідно, якщо користувачеві потрібно було автоматизувати якусь дію в Excel, він мав використовувати одну мову макрокоманд (тоді це були макроси XLM), а для Word — зовсім іншу (наприклад, WordBasic). Це створювало неймовірний хаос, оскільки працівникам доводилося вивчати кілька абсолютно різних синтаксисів для виконання схожих завдань.

У 1993 році компанія Microsoft зробила крок, який фактично визначив розвиток офісного софту на роки вперед. З виходом Excel версії 5.0 вони вперше інтегрували туди мову VBA. Ідея була простою, але геніальною: взяти вже популярну на той час мову Visual Basic 6.0 і адаптувати її так, щоб вона могла керувати внутрішніми елементами офісних додатків. Вже за кілька років, у версії Office 97, ця технологія стала стандартною для всього пакету, включаючи Word, Access та PowerPoint. Користувачі нарешті отримали один універсальний інструмент.

Сьогодні часто можна почути думку, що VBA — це застаріла технологія, яка доживає свої останні дні. Дійсно, Microsoft вже багато років не оновлює саму мову програмування, а в останні версії Microsoft 365 активно впроваджують інструменти на базі JavaScript та TypeScript (так звані Office Scripts). Проте на практиці в корпоративному секторі VBA залишається домінувати. Чому так відбувається? По-перше, за тридцять років існування мови на підприємствах, у банках та державних установах було створено мільярди рядків коду. На цих

макросах тримається звітність величезних корпорацій, і переписувати все це на нові платформи — це колосальні фінансові витрати та ризики щось зламати. По-друге, VBA працює локально на комп'ютері, без необхідності підключення до інтернету чи хмарних серверів, що вкрай важливо з точки зору безпеки даних і конфіденційності фінансової інформації. Тому, попри свій солідний вік, ця мова залишається базовим інструментом для швидкої автоматизації робочих процесів прямо на робочому місці.

З технічної точки зору VBA має досить специфічну структуру, яку часто плутають з класичними мовами програмування. Вона не створює окремі виконувані файли (екзешники з розширенням .exe). Весь код, який ми пишемо, зберігається безпосередньо всередині самого документа офісної програми — у файлі Excel з розширенням .xlsm або Word з розширенням .docm.

Коли ми запускаємо макрос, відбувається процес, який поєднує в собі елементи компіляції та інтерпретації. Спочатку, коли ми натискаємо кнопку запуску або коли код перевіряється редактором, текст програми переводиться у проміжний стан — так званий P-код (packed code або pseudo-code). Це ще не чисті машинні коди, які розуміє процесор, але вже й не звичайний англійський текст, який написав користувач. Потім цей P-код виконується спеціальним віртуальним двигуном (VBA Virtual Machine), який вшитий в офісний пакет. Цей двигун вже на льоту інтерпретує команди для операційної системи.

Така схема має свої плюси й мінуси. Величезний плюс — це портативність і автономність. Вам не потрібно збирати проєкт, налаштовувати зв'язки чи думати, чи є у користувача потрібні бібліотеки для запуску програми. Ви просто перекидаєте файл флешкою чи поштою, і якщо на комп’ютері користувача є Офіс, макрос запуститься. Але є і мінус — це швидкість роботи. Оскільки код інтерпретується під час виконання, VBA працює значно повільніше, ніж програми, написані на C++ чи C#. Крім того, VBA є строго однопотоковою мовою. Що це означає на практиці? Коли ви запускаєте складний макрос, який, наприклад, перебирає 50 тисяч рядків і щось там рахує, вся програма Excel повністю «замерзає». Процесор віддає всі сили на виконання цього коду в один

потік, інтерфейс покривається білою димкою, а система пише «Не відповідає». Користувач не може паралельно відкрити інший лист чи щось змінити, поки макрос повністю не завершить свою роботу. Це класичне обмеження, яке доводиться враховувати при оптимізації великих проєктів автоматизації.

Дуже часто у підручниках пишуть, що VBA — це об'єктно-орієнтована мова програмування (ООП). Насправді це не зовсім так. Правильніше називати VBA об'єктно-базованою мовою. У ній є об'єкти, класи, інкапсуляція (ми можемо приховувати дані через Private), є навіть поліморфізм через використання інтерфейсів, але в ній повністю відсутнє класичне спадкування. Ви не можете створити свій новий клас, який просто успадкує всі властивості вже існуючого класу і додасть дві нові кнопки.

Проте вся логіка роботи всередині офісних програм побудована навколо маніпуляцій з об'єктами. Можна уявити, що кожна програма Microsoft — це величезний конструктор Lego, де кожен кубик — це окремий об'єкт. Всі ці кубики складені в чітку ієрархію, яку називають об'єктною моделлю додатка. Для кожного додатка (Excel, Word, Access) ця модель своя, хоча загальний принцип однаковий.

Детально розберемо структуру моделі для Microsoft Excel, оскільки з нею автоматизація відбувається найчастіше. На самому верху стоїть об'єкт Application — це, власне, сам запущений процес Excel. Через нього можна керувати глобальними налаштуваннями програми (наприклад, вимикати оновлення екрану під час роботи макросу, щоб усе працювало швидше).

Всередині Application знаходиться колекція Workbooks — це список усіх файлів (книг), які зараз відкриті в програмі. Конкретний файл — це об'єкт Workbook.

Кожен Workbook містить колекцію Worksheets (всі аркуші в цьому файлі), а один конкретний аркуш — це об'єкт Worksheet.

І вже всередині аркуша ми добираємося до найголовнішого елемента — об'єкта Range. Це може бути як одна конкретна клітинка (наприклад, Range("A1")), так і ціла група клітинок чи велика таблиця (Range ("A1:G100")).

У Microsoft Word логіка схожа, але там інші об'єкти. На вершині так само стоїть Application, далі йде колекція Documents (відкриті текстові файли) і конкретний Document. А от всередині документа текст ділиться не на клітинки, а на колекції Paragraphs (абзаци), Sentences (речення) або Words (окремі слова). Також у Word є дуже важливий об'єкт Selection (це те, що наразі виділено курсором на екрані) та об'єкт Range, який, на відміну від Excel, означає просто певну приховану ділянку тексту від символу X до символу Y.

Розуміння цієї ієрархії критично важливе. Якщо ви хочете записати якесь значення в клітинку A1 на першому аркуші у файлі під назвою "Звіт", ви не можете просто написати "запиши туди п'ять". Програма має пройти по всьому ланцюжку (хоча деякі елементи можна пропускати, якщо файл уже активний): Application.Workbooks("Звіт.xlsm").Worksheets("Аркуш1").Range("A1").Value = 5.

Кожен такий об'єкт має свої властивості (Properties), які описують його стан (наприклад, колір фону, розмір тексту, ширина стовпчика), та методи (Methods) — це дії, які об'єкт вміє виконувати за командою коду (наприклад, скопіювати, видалити, зберегти на диск).

Окрім роботи з об'єктами, VBA базується на принципах подійно-орієнтованого програмування. Звичайні програми, як ми звикли думати, працюють лінійно: користувач запустив її, вона пройшла по черзі всі рядки коду від першого до останнього, видала результат і закрилася. В офісних макросах усе працює трохи інакше. Код може годинами просто сидіти в пам'яті комп'ютера і нічого не робити, чекаючи на конкретну дію від користувача. Така дія називається подією (Event).

Цей підхід дозволяє створювати так звані "розумні документи", які самі реагують на те, що робить людина. Розробник може прив'язати виконання свого коду до величезної кількості стандартних подій системи.

Наприклад, на рівні всієї робочої книги (Workbook) можна перехоплювати такі події:

- Workbook_Open — макрос запускається автоматично в ту саму секунду, коли користувач відкриває файл. Тут зазвичай прописують підключення до баз даних, перевірку оновлень або створення красивого вікна привітання.
- Workbook_BeforeClose — код спрацьовує прямо перед тим, як файл закриється. Можна автоматично перевірити, чи всі поля заповнені, очистити тимчасові дані або зробити резервну копію файлу в іншу папку.
- Workbook_BeforePrint — запуск коду перед відправкою документа на друк (наприклад, щоб автоматично додати на кожну сторінку актуальну дату або водяний знак "Конфіденційно").

На рівні окремого аркуша (Worksheet) найпопулярнішою є подія Worksheet_Change. Вона фіксує будь-яку зміну даних у клітинках. Це ідеальний інструмент для автоматичного контролю за помилками введення (валідації даних). Наприклад, якщо працівник вводить у стовпчик "Номер телефону" літери замість цифр, подія Worksheet_Change миттєво це помічає, скасовує введення і виводить на екран велике попередження. Користувач навіть не знає, що всередині працює програма, для нього це виглядає як інтерактивний інтерфейс самого Excel.

2.2. Інтегроване середовище розробки VBE

Для створення та редагування макросів використовується спеціальний вбудований додаток — Visual Basic Editor (VBE). Завантажується VBE комбінацію клафіш <Alt> + <F11>. Інтерфейс VBE виглядає дуже старомодно, він практично не змінився з кінця 90-х років: сірі вікна, стандартні шрифти, жодної сучасної підсвітки чи темної теми, до яких звикли сучасні програмісти. Проте з точки зору функціоналу це повноцінне середовище розробки (IDE), де є все необхідне. Розглянемо детально внутрішню структуру редактора VBE (рисунок 2.1).

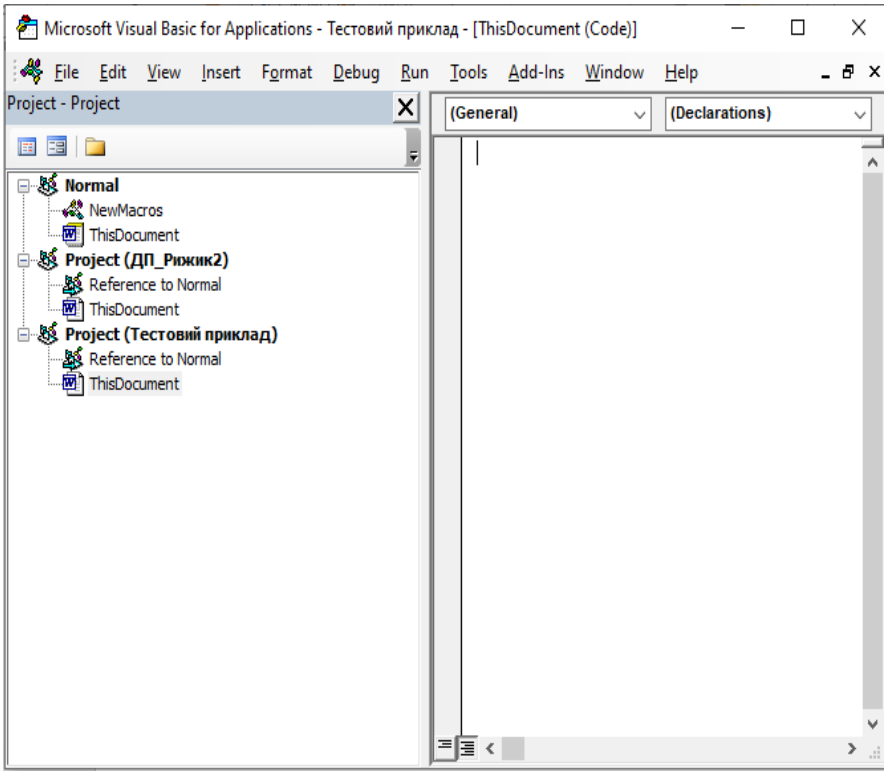

Рисунок 2.1. – . Вікно проекту

Головне вікно ліворуч — це **Project Explorer**. Воно схоже на звичайний провідник Windows, але показує внутрішню структуру вашого офісного файлу. Там у вигляді дерева відображаються всі об'єкти програми (аркуші, сама книга), а також папки з компонентами, які ви самі створили (рисунок 2.2).

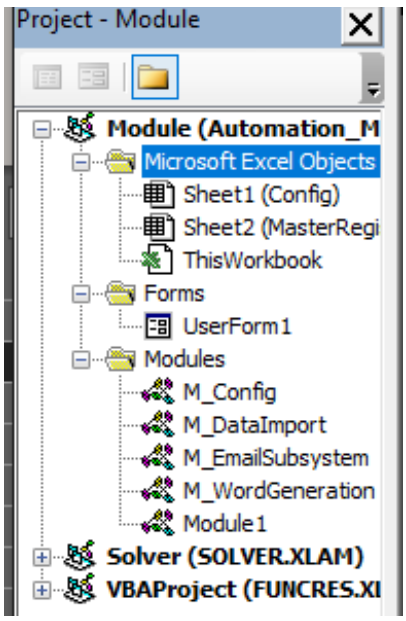

Рисунок 2.2. – . Панель Project explorer з компонентами проєкту

Ці компоненти (модулі) бувають кількох типів, і кожен має своє призначення:

1. **Стандартні модулі (Modules)** — це найзвичайніші чисті текстові аркуши, де пишеться 90% усього коду. Процедури, які тут лежать, називаються глобальними, їх можна викликати з будь-якого місця програми чи навіть прив'язати до кнопки на панелі інструментів (рисунок 2.3).

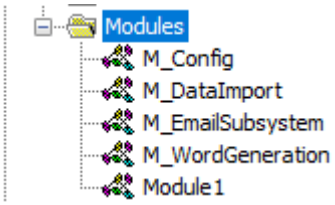

Рисунок 2.3. – Структура папки стандартних модулів

2. **Модулі об'єктів (ThisWorkbook, Sheet1 тощо)** — вони прив'язані до конкретного елемента. Код, написаний всередині модуля Sheet1, зазвичай керує саме подіями цього аркуша і не повинен викликатися з інших місць.

3. **Форми користувача (UserForms)** — це візуальний конструктор, де можна мишкою малювати власні інтерфейсні вікна. Туди можна накидати текстові поля, випадаючі списки, прапорці та кнопки. Це робиться для того, щоб звичайний офісний працівник взагалі не бачив таблиць, а працював через зручну форму, як у повноцінних програмах обліку. (рисунок 2.4).



Рисунок 2.4. – . Об`єкт UserForm у провіднику проєкту (Project Explorer)

4. **Модулі класів (Class Modules)** — використовуються рідко, переважно для складних проєктів, коли розробнику потрібно створити свій власний унікальний тип об'єкта, якого немає в стандартному наборі Microsoft Office.

Під Project Explorer зазвичай знаходиться вікно **Properties Window**. У ньому відображаються технічні параметри виділеного елемента. Наприклад, якщо вибрати аркуш Excel, тут можна змінити його системне ім'я або ввімкнути властивість xlSheetVeryHidden, зробить аркуш "дуже прихованим", користувач не зможе побачити його чи увімкнути назад через звичайне контекстне меню Excel — такий аркуш видно тільки через код VBA, що корисно для приховування якихось розрахунків чи технічних баз даних.

Окреме місце займають інструменти відлагодження (дебаггінгу), без яких написати працюючу програму неможливо, адже коди з першого разу ніколи не працюють як треба. Найважливіше вікно тут — **Immediate Window** (викликається через <Ctrl> +< G>). Це інтерактивний рядок, куди можна в процесі роботи виводити службові повідомлення за допомогою команди Debug.Print. Також туди можна прямо під час зупинки макросу написати знак питання і команду (наприклад, ? ActiveWorkbook.Name), натиснути Enter і миттєво отримати відповідь від системи, що зараз відбувається всередині пам'яті комп'ютера.

Для детального аналізу є вікно **Locals Window**, яке автоматично показує взагалі всі змінні, які зараз створені в макросі, їхні типи та як змінюються їхні значення в реальному часі. Також є вікно **Watch Window**, куди можна вручну додати якусь конкретну змінну або логічний вираз і сказати програмі: "Зупини виконання макросу, як тільки значення цієї змінної стане більшим за 100". Це дозволяє відловлювати складні логічні помилки в циклах.

Говорячи про середовище розробки, не можна не згадати такий інструмент як **Макрорекордер (Macro Recorder)**. Це внутрішній робот-транслятор, який можна увімкнути в Excel чи Word. Він починає стежити за кожним кліком мишки користувача, кожним введенням тексту чи форматуванням і автоматично перекладає ці дії в код VBA. Проте використовувати цей код у дипломній роботі в чистому вигляді не можна. Макрорекордер пише код дуже примітивно й громіздко. Він записує абсолютно всі рухи, включаючи банальну прокрутку екрану мишкою, і постійно використовує методи .Select та .Activate (тобто спочатку виділяє клітинку на екрані, а потім робить з нею дію). Якщо такий автоматичний макрос запустити на великій таблиці, комп'ютер буде дуже довго гальмувати, бо він змушений реально перемальовувати на екрані кожен клік. Тому згенерований код завжди потрібно чистити вручну, прибираючи виділення екрану й переписуючи звернення до об'єктів напряму.

2.3. Синтаксична структура мови VBA, алгоритмічні конструкції

Синтаксис мови VBA вважається одним із найпростіших у вивченні, оскільки він розроблявся так, щоб команди читалися як звичайні англійські речення. Тут немає фігурних дужок, як у С-подібних мовах, а блоки коду виділяються зрозумілими словами (наприклад, якщо блок починається з If, то закінчується він обов'язково End If). Наприкінці рядків не потрібно ставити крапку з комою — програма розуміє, що перехід на новий рядок є закінченням поточної команди. Якщо ж команда занадто довга, її можна переносити на новий рядок за допомогою символу підкреслення _ з попереднім пробілом.

Однією з головних тем, навколо якої виникає багато дискусій під час створення офісних систем, є типізація даних і оголошення змінних. У VBA за замовчуванням увімкнена дуже вільна система. Ви можете просто написати в коді назву нової змінної, наприклад x = 10, і програма не буде сваритися, вона сама створить цю змінну в пам'яті. Але в реальній розробці це величезна проблема. Якщо ви десь нижче за кодом випадково помилитеся в літері й напишете x = 11 (використавши іншу літеру чи опечатавшись), програма створить ще одну, нову змінну, а ваші розрахунки повністю зблудять, і ви будете днями шукати, чому підсумковий звіт рахує неправильно.

Щоб уникнути цього, на самому початку кожного модуля обов'язково потрібно прописувати команду **Option Explicit**. Цей рядок жорстко наказує компілятору: "Забороняй запуск програми, якщо в тексті є хоча б одна змінна, яка не була офіційно оголошена через оператор Dim".

Оголошення змінної з чітким типом виглядає так: Dim RowIndex As Long.

VBA підтримує кілька основних типів даних, і правильний їх вибір безпосередньо впливає на те, чи буде ваш макрос "літати", чи він "вкладе" оперативну пам'ять комп'ютера:

- Integer — цілі числа в діапазоні від -32768 до 32767. Займає в пам'яті всього 2 байти. Раніше його активно використовували, але зараз у великих таблицях Excel його використовувати небезпечно. Якщо ваш макрос перебирає

рядки, і кількість рядків перевищить 32767, програма миттєво вилетить з критичною помилкою "Overflow" (переповнення пам'яті).

- Long — великі цілі числа (до 2 мільярдів). Займає 4 байти. Це золотий стандарт для будь-яких лічильників, індексів рядків та колонок у сучасних офісних системах. Цікавий факт: сучасні 32-бітні та 64-бітні процесори всередині себе все одно конвертують Integer у Long для швидкості обчислень, тому використання Integer сьогодні взагалі втратило сенс — воно лише уповільнює роботу.
- Single та Double — числа з плаваючою точкою (дроби). Double значно точніший і займає 8 байт, його використовують для складних фінансових чи інженерних розрахунків.
- Currency — спеціальний фінансовий тип даних із фіксованою точкою (4 знаки після коми). Він розроблений спеціально для того, щоб уникати дрібних помилок округлення при роботі з грошима, які часто виникають у типі Double.
- String — текстові дані. Може містити як один символ, так і величезні тексти обсягом до 2 мільярдів символів.
- Boolean — логічний тип, який приймає лише два значення: True (істина) або False (хибність). Дуже зручно для різних перемикачів у логіці програми.
- Date — зберігає дату і час одночасно у спеціальному внутрішньому форматі.
- Variant — універсальний тип даних. Якщо ви оголосили змінну просто як Dim x і не вказали тип, або взагалі забули про Option Explicit, VBA автоматично дасть цій змінній тип Variant. Вона може зберігати в собі абсолютно все: зараз там лежить число, через рядок коду ви записуєте туди текст, а потім — цілий масив даних. З одного боку, це здається дуже зручним. Але з іншого — це найгірше, що можна зробити для продуктивності програми. Змінна типу Variant займає в пам'яті від 16 до 22 байт (проти 4 байти у Long), і під час кожної операції процесор витрачає дорогоцінні мікросекунди на те, щоб розшифрувати, що саме

зараз лежить всередині цієї змінної. Якщо у вас працює великий цикл на тисячі ітерацій, використання варіантних змінних може уповільнити роботу макросу в кілька разів.

Також у VBA є можливість створювати масиви даних (матриці), які бувають статичними (коли розмір задано чітко наперед) та динамічними (розмір яких можна змінювати прямо в процесі роботи програми за допомогою команди ReDim). Особливо корисною є ReDim Preserve — вона дозволяє розширити масив, не затерши при цьому ті дані, які вже були туди записані раніше.

Алгоритмічні конструкції

Для побудови будь-якого автоматизованого процесу всередині коду необхідно керувати логікою виконання дій. VBA пропонує для цього класичний набір інструментів керування потоком даних, який складається з умовних операторів та кількох видів циклів.

Базове розгалуження робиться через конструкцію ***If...Then...ElseIf...Else...End If***. Вона працює стандартно: якщо умова виконується, робимо дію А, якщо ні — перевіряємо наступну умову або робимо дію Б. Якщо ж умов стає занадто багато (наприклад, залежно від коду регіону підприємства треба виставити різні податкові ставки), писати купу ElseIf стає незручно — код перетворюється на нечитабельний текст. У таких випадках використовують більш елегантну конструкцію ***Select Case***. Вона бере одну змінну і порівнює її значення з готовим списком варіантів, що значно спрощує читання та подальше редагування програми іншими розробниками (рисунок 2.5).

```
Set FSO = CreateObject("Scripting.FileSystemObject")

If Not FSO.FolderExists(PathInBox) Then
    MsgBox "Вказану папку вхідних звітів не знайдено: " & PathInBox, vbExclamation, "Помилка шляху"
    GoTo AirExit
End If
```

Рисунок 2.5 – Приклад використання конструкції ***Select Case***

Для повторення однотипних операцій (наприклад, перебрати всі рядки в таблиці, перевірити кожен абзац у документі чи створити 20 нових файлів) використовуються цикли. У VBA є три основних типи циклів:

1. Цикл із лічильником ***For ... Next***. Він використовується тоді, коли ми чітко знаємо, скільки разів треба повторити дію (наприклад, від 1 до 500). Тут також можна задавати крок перебору за допомогою команди Step (наприклад, For i = 10 To 1 Step -1 дозволяє йти по рядках таблиці знизу вгору, що критично важливо, якщо всередині циклу відбувається видалення якихось рядків, інакше індекси просто зсунуться, і код пропустить половину даних).

2. Цикл за колекціями ***For Each...Next.*** Це, мабуть, найважливіший і найефективніший цикл для офісної автоматизації. Замість того, щоб рахувати об'єкти за номерами, він каже системі: "Візьми по черзі кожен об'єкт із цієї групи". Наприклад, For Each sh In Worksheets дозволяє автоматично перебрати всі аркуши в книзі, незалежно від їх кількості та назв. Цей цикл працює на внутрішньому рівні значно швидше, ніж звичайний For з індексами (рисунок 2. 6.)

```
Set FSO = CreateObject("Scripting.FileSystemObject")

If Not FSO.FolderExists(PathInBox) Then
    MsgBox "Вказану папку вхідних звітів не знайдено: " & PathInBox, vbExclamation, "Помилка шляху"
    GoTo AirExit
End If
```

Рисунок 2.6, – Приклад циклу за колекціями ***For Each...Next***

3. Умовні цикли ***Do While ma Do Until***. Вони крутяться доти, доки виконується (або поки не виконається) певна умова. Їх зазвичай використовують, коли ми заздалегідь не знаємо, скільки у нас даних. Наприклад, ми кажемо програмі: "Читай рядки один за одним, починаючи з першого, і створюй акти, поки не наткнешся на повністю порожню клітинку". Тут треба бути дуже обережним, бо якщо помилитися в умові, цикл може стати нескінченним, програма повністю "завісить" комп'ютер, і всі незбережені дані користувача будуть втрачені.

Весь код у модулях ділиться на дві великі групи: процедури (які починаються зі слова ***Sub***) та функції (які починаються зі слова ***Function***). Головна різниця між ними в тому, що процедура просто виконує набір якихось команд (наприклад, копіює дані, фарбує таблицю, зберігає файл), але нічого не повертає замість себе. А от функція обов'язково робить розрахунки й повертає конкретний результат у точку виклику. Більше того, користувацькі функції (їх називають UDF — User Defined Functions), які написані в стандартних модулях, можна використовувати прямо на аркушах Excel як звичайні стандартні формули (на кшталт СУММ чи ВПР), створюючи власну унікальну математичну логіку для специфічних розрахунків підприємства.

При передачі змінних всередину процедур чи функцій є один тонкий інженерний момент. Це передача аргументів за посиланням або за значенням:

- ***ByRef*** (за замовчуванням) — передається посилання на саму клітинку пам'яті, де лежить змінна. Якщо всередині підпрограми ми змінимо значення цього аргументу, воно автоматично зміниться і в головній програмі.
- ***ByVal*** — всередину підпрограми передається лише чиста копія значення. Що б ми не робили з цією змінною всередині процедури, в головній програмі вона залишиться абсолютно незмінною. Правильне розуміння цього нюансу захищає від раптових логічних збоїв у великих аналітичних системах.

2.4. Технології інтеграції COM та OLE: механізми міжпрограмної взаємодії

Однією з найсильніших сторін мови VBA, завдяки якій вона досі тримається в топі корпоративного сектору, є її здатність повністю руйнувати кордони між різними програмами. Це реалізується завдяки системним технологіям Windows — COM (Component Object Model) та OLE (Object Linking and Embedding). Суть цієї концепції в тому, що будь-який офісний додаток Microsoft не є ізольованою "коробкою". Це відкритий набір компонентів, якими може керувати зовнішній код.

На практиці це дозволяє будувати наскрізні системи документообігу. Наприклад, ви працюєте в Excel, де у вас збирається аналітика з продажу. Ви можете написати один макрос, який за натисканням однієї кнопки зробить наступне:

1. Зчитає інформацію про нових клієнтів і суми договорів з таблиці;
2. Створить у пам'яті комп'ютера прихований процес програми Microsoft Word;
3. Відкриє офіційний шаблон договору компанії, за допомогою функції пошуку та заміни автоматично вставить туди ПІБ клієнта, реквізити та суми;
4. Збереже цей договір на жорсткий диск або відправить у хмару;
5. Одразу ж ініціалізує поштовий клієнт Microsoft Outlook, створить новий лист, прикріпить туди щойно створений файл договору, візьме з таблиці Excel електронну адресу клієнта і відправить цей лист.

Користувач при цьому може бачити лише бігунок завантаження, а вся гігантська рутинна робота кількох програм виконується самостійно за лічені секунди.

Щоб налаштувати таку взаємодію між програмами, у VBA використовують два різні підходи: раннє та пізнє зв'язування об'єктів.

Раннє зв'язування (Early Binding) вимагає, щоб розробник спочатку зайшов у меню редактора VBE Tools -> References і вручну поставив галочку навколо потрібної бібліотеки (наприклад, *Microsoft Word Object Library*). Після цього змінні можна оголошувати чітко: Dim appWord As Word.Application.

- *Плюси*: макрос працює максимально швидко, компілятор заздалегідь знає всі команди, а під час написання коду працює функція автоматичного доповнення слів (IntelliSense). Також стають доступними внутрішні текстові константи програми (наприклад, можна писати wdFormatPDF, і система зрозуміє, що це конвертація в PDF).
- *Мінуси*: повна втрата універсальності версій. Якщо ви написали цей код в Office 2021 (де підключена 16-та версія бібліотеки), а ваш колега відкриє цей файл на комп'ютері зі старим Office 2016, програма видасть критичну

помилку "Missing Reference", повністю заблокує роботу і навіть не спробує виконати код, поки користувач вручну не виправить налаштування бібліотек.

Пізнє зв'язування (Late Binding) не потребує підключення жодних галочок у налаштуваннях. Всі зовнішні об'єкти оголошуються як універсальний тип: Dim appWord As Object. А створення самого об'єкта відбувається динамічно прямо під час виконання коду за допомогою спеціальної системної команди: Set appWord = CreateObject("Word.Application").

- *Плюси*: залізна стабільність і сумісність. Коду абсолютно байдуже, яка саме версія Word стоїть на комп'ютері користувача — програма сама знайде її в системному реєстрі й запустить.
- *Мінуси*: працює трохи повільніше, оскільки система витрачає час на пошук команд у реєстрі Windows під час кожного кроку. Також при написанні коду не працюють підказки IntelliSense, а всі внутрішні константи зовнішньої програми доводиться замінювати їхніми числовими кодами (наприклад, замість слова wdFormatPDFдоводиться писати цифру 17, яку треба заздалегідь шукати в технічній документації Microsoft).

При використанні будь-якого зв'язування в кінці коду обов'язково потрібно правильно очищувати пам'ять, прописуючи ручне закриття об'єктів і команду Set appWord = Nothing. Якщо цього не зробити, то після кожного запуску макросу в диспетчері завдань Windows буде залишатися "висіти" мертвий фоновий процес Word або Outlook, який потроху з'їдатиме оперативну пам'ять комп'ютера.

2.5. Практичні обмеження та недоліки використання VBA

Коли ми розглядаємо VBA як інструмент для автоматизації бізнес-процесів, не можна ігнорувати питання інформаційної безпеки. Історично так склалося, що VBA заробив собі досить сумнівну репутацію через проблеми з вірусами. Оскільки мова інтегрована прямо в офісні документи, які люди щодня пересилають один одному мільйонами (акти, рахунки, прайси), хакери швидко зрозуміли, що це ідеальний канал для доставлення шкідливого коду.

У кінці 90-х та на початку 2000-х років світом котилися гігантські епідемії так званих макровірусів (найвідоміший приклад — вірус Melissa, який у 1999 році паралізував поштові системи найбільших компаній світу). Суть загрози полягала в тому, що макрос VBA має прямий доступ до файлової системи комп'ютера через бібліотеки Windows API або об'єкт Scripting.FileSystemObject. Тобто, звичайна табличка Excel, якщо в ній увімкнено макроси, може спокійно створювати, перейменовувати чи видаляти будь-які файли на вашому жорсткому диску, зчитувати паролі або завантажувати сторонні віруси з інтернету. Раніше макроси запускалися автоматично без жодних попереджень, що й призвело до катастрофи.

Через це корпорація Microsoft кардинально змінила архітектуру безпеки офісного пакету, і сьогодні ці правила стали жорсткими як ніколи:

1. **Розділення форматів файлів.** Починаючи з версії Office 2007, звичайні файли .xlsx або .docx фізично не можуть містити всередині себе жодного рядка коду VBA. Навіть якщо ви напишете макрос і спробуєте зберегти файл у цьому форматі, програма просто видалить увесь ваш код. Для документів із макросами створено окремі розширення — .xlsm, .docm, .xlsb. Це дозволяє користувачеві або системному адміністратору компанії одразу бачити потенційно небезпечні файли ще до їх відкриття.

2. **Центр керування безпекою (Trust Center).** У сучасних версіях Офісу за замовчуванням увімкнено режим "Блокувати всі макроси зі сповіщенням". Коли ви відкриваєте файл із кодом, зверху з'являється жовта плашка з попередженням, і код не почне працювати, поки користувач усвідомлено не натисне кнопку "Увімкнути вміст".

3. **Механізм Mark of the Web (MOTW).** Якщо система бачить, що файл .xlsm був завантажений з інтернету, отриманий через месенджер або прийшов як вкладення в пошті Outlook, вона вішає на нього спеціальну мітку "Mark of the Web". При спробі відкрити такий файл звичайна жовта кнопка взагалі не з'явиться. Офіс видасть велике червоне вікно з повідомленням, що макроси заблоковані з міркувань безпеки, оскільки джерело файлу є підозрілим.

Для того, щоб запустити такий макрос, доводиться закривати файл, заходити в його властивості через провідник Windows і вручну ставити галочку "Розблокувати" на вкладці "Загальні".

Для легального використання макросів у великих компаніях розробники використовують цифрові підписи. За допомогою внутрішньої утиліти Windows (SelfCert) можна створити свій персональний цифровий сертифікат і підписати ним проєкт VBA. Також компанії налаштовують так звані "Довірені розташування" (Trusted Locations) — конкретні папки на мережевих дисках офісу, файли з яких запускаються без жодних перевірок та блокувань, оскільки система вважає цю зону повністю безпечною.

Завершуючи теоретичний огляд технології кваліфікаційної роботи необхідно провести чесний і об'єктивний аналіз обмежень, з якими стикається будь-яке підприємство при впровадженні рішень на базі VBA. Розуміння мінусів системи так само важливе, як і знання її переваг, оскільки це дозволяє чітко окреслити межі її застосування.

Окрім вже згаданої однопотоковості, через яку інтерфейс програми зависає, під час важких розрахунків, та жорстких блокувань системи безпеки, існують інші серйозні проблеми:

- **Повна відсутність кросплатформеності.** VBA — це дитя операційної системи Windows. Він глибоко зав'язаний на її внутрішні бібліотеки та архітектуру COM. Якщо ваша компанія переходить на мобільні планшети (на базі iOS чи Android) або працівники намагаються відкривати документи через веб-браузер в Office Online (наприклад, через Sharepoint), ваші макроси там працювати не будуть взагалі. Вони просто ігноруються системою. На комп'ютерах Apple Mac (macOS) ситуація трохи краща — там є вбудований редактор VBA, але його функціонал сильно урізаний. Багато корисних елементів керування ActiveX (наприклад, красиві календарі чи специфічні списки для форм) на Mac не підтримуються, а спроба викликати будь-яку функцію через Windows API призводить до миттєвого аварійного закриття програми.

- **Катастрофічна складність командної розробки.** Сучасне програмування тримається на системах контролю версій, таких як Git чи SVN. Вони дозволяють п'ятьом програмістам одночасно писати один і той самий код, бачити зміни кожного, легко об'єднувати роботу і повертатися до старих версій у разі помилок. У середовищі VBE нічого подібного немає. Оскільки весь код зашитий всередині бінарного файлу Excel, звичайний Git не може побачити зміни всередині модулів, для нього це просто один змінений великий шматок даних. Якщо два студенти або два аналітики на фірмі одночасно почнуть правити різні макроси в одному файлі, об'єднати їхню роботу автоматично буде неможливо — комусь доведеться вручну копіювати текст через блокнот. Існують сторонні інструменти для експорту модулів у текстові файли, але вони складні в налаштуванні й майже не використовуються на практиці.
- **Проблема підтримки коду (Legacy Code).** Макроси на підприємствах часто пишуться людьми, які не є професійними програмістами — бухгалтерами, економістами чи логістами, які просто пройшли короткі курси або навчилися всього по відео в інтернеті. Такий код часто називають "спагеті-кодом" — у ньому немає чіткої структури, змінні називаються випадковими буквами типу аааа або test1, відсутні будь-які коментарі. Доки цей працівник працює на фірмі, макрос сяк-так функціонує. Але як тільки людина звільняється, а в бізнес-процесах компанії щось змінюється (наприклад, додається новий стовпчик у звіт або змінюється форма податкової декларації), макрос повністю ламається. Знайти кінці в чужому заплутаному коді стає настільки важко, що компаніям часто простіше викинути старий файл і найняти людину, щоб вона написала все з нуля.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА СТРУКТУРИЗАЦІЯ МОДУЛІВ АВТОМАТИЗАЦІЇ В СЕРЕДОВИЩІ MICROSOFT OFFICE

3.1. Архітектурне планування системи та конфігурація файлової структури

При створенні локальних систем автоматизації на базі Visual Basic for Applications першочерговою інженерною проблемою є відсутність централізованого сховища даних. Оскільки розробка виконується повністю всередині екосистеми Microsoft Office, роль координаційного центру керування виконує спеціально налаштований файл Excel із розширенням .xlsm. Було прийнято рішення відмовитися від хаотичного збереження макросів у різних документах і об'єднати всю логіку в єдиному керуючому модулі.

Для забезпечення автономності роботи програми на будь-якій робочій станції підприємства була спроєктована відносна структура каталогів. Програма не використовує абсолютні шляхи до дисків, оскільки при перенесенні файлу на інший комп'ютер це викликає дефіцит доступу через різницю в іменах облікових записів операційної системи Windows. Заощадження гнучкості досягнуто через метод ThisWorkbook.Path, який динамічно визначає поточне розташування головного файлу.

Файлова структура проєкту в локальній директорії має такий вигляд:

- Корневий каталог, де розташовано головний файл макросів Automation_Main.xlsm;
- Підкаталог InBox — сюди надходять вхідні звіти від підрозділів чи менеджерів;
- Підкаталог Templates — містить файли шаблонів Word з розширенням .docx для генерації угод;
- Підкаталог OutBox — каталог для збереження готових сформованих документів;
- Підкаталог Logs — директорія для автоматичного збереження звітів про помилки обробки.

Всередині самого файлу Automation_Main.xlsm було створено окремий службовий аркуш під назвою Config. Це дозволило відокремити бізнес-логіку від програмного коду. На цьому аркуші у вигляді таблиці записані технічні параметри: назви папок, маски пошуку файлів (наприклад, *.xlsm), номери ключових стовпчиків для зчитування даних та електронні адреси для розсилки. Якщо на підприємстві зміниться назва папки для збереження документів або додасться новий стовпчик у звітність, оператор зможе переписати значення на аркуші Config, не вносячи змін у сам програмний код.

Технічна реалізація системи базується на кількох стандартних модулях, коди яких винесені в додатки до роботи.

Модуль M_Config відповідає за налаштування глобальних параметрів програми. Програмний код модуля M_Config подано в Додатку А.

Для запобігання логічним помилкам, які виникають через друкарські помилки або неявне оголошення змінних, на початку модуля прописано команду Option Explicit. Вона змушує компілятор перевіряти наявність офіційного оголошення для кожної змінної через оператор Dim або Public.

Змінні для збереження шляхів до папок (PathRoot, PathInBox тощо) оголошені за допомогою оператора Public, що робить їх доступними для всіх інших макросів проєкту. Для індексації стовпчиків головної таблиці створено константи з типом даних Long (наприклад, COL_CONTRAGENT, COL_CODE). Вибір типу Long замість стандартного Integer є обов'язковим для сучасних офісних систем, оскільки Integer має обмеження по пам'яті до 32 767. Якщо кількість рядків у таблиці перевищить це число, використання Integer призведе до критичної помилки переповнення пам'яті "Overflow", тоді як Long дозволяє коректно обробляти великі масиви даних.

Процедура LoadConfig (Додаток А) зчитує відносні шляхи до папок із аркуша Config. За допомогою властивості ThisWorkbook.Path макрос визначає, де саме на жорсткому диску розташований сам файл Excel, а потім через оператор конкатенації рядків & додає до нього назви підпапок, які вказав користувач в інтерфейсі таблиці. Блок On Error GoTo захищає програму від

вильоту, якщо користувач випадково видалив або перейменував конфігураційний аркуш.

3.1.1. Розробка модуля сканування дискового простору та підключення FileSystemObject

Коли задача полягає у зборі даних із багатьох різних файлів, які розташовані в папці InBox, базові інструменти мови показують низьку ефективність. Звичайна функція Dir(), яка часто використовується у простих скриптах, працює через системний буфер операційної системи. Якщо під час циклу перебору файлів за допомогою Dir() алгоритм змушений відкрити інший файл Excel, внутрішній покажчик функції збивається. Це призводить до пропускання файлів або зациклення програми.

Щоб вирішити цю проблему, в модулі імпорту було застосовано об'єктне сканування через бібліотеку Microsoft Scripting Runtime, а саме об'єкт Scripting.FileSystemObject (FSO). Це дає змогу працювати з дисковим простором через об'єкти папок та файлів.

При підключенні бібліотеки виник нюанс щодо регістру розширень. Працівники на місцях часто зберігають файли Excel з різними розширеннями: .xlsx, .xlsm, а іноді великими літерами .XLSX. Стандартне текстове порівняння в логіці If сприймає це як різні типи даних. Тому в алгоритм було вбудовано примусове приведення розширення до нижнього регістру за допомогою функції LCase(FSO.GetExtensionName(FileObject.Path)).

Також під час тестів виявлено проблему з тимчасовими файлами. Коли файл Excel відкритий іншим користувачем у локальній мережі, операційна система Windows створює в тій самій папці прихований тимчасовий файл, назва якого починається зі знаків ~$. Якщо макрос намагається відкрити такий файл, програма зупиняється через помилку доступу. Для ліквідації цього дефекту в цикл перебору було додано фільтр: якщо перші два символи імені файлу збігаються з ~$, код пропускає цей об'єкт і переходить до наступного елемента.

3.1.2. Аналіз модуля імпорту та очищення даних

Модуль M_DataImport виконує збір інформації з папки InBox. Програмний код модуля подано в Додатку Б.

Головна процедура ConsolidateReports спочатку вимикає графічні процеси Excel за допомогою команд VBA:

```
Application.ScreenUpdating = False
Application.Calculation = xlCalculationManual
Application.EnableEvents = Fals.
```

Це відключає перемальовування екрану та автоматичний перерахунок формул під час роботи макросу, що дозволяє суттєво прискорити виконання коду.

Для перебору файлів створюється об'єкт Scripting.FileSystemObject. У циклі For Each програма аналізує розширення кожного файлу через метод FSO.GetExtensionName. Функція LCase переводить текст розширення у нижній регістр, що дозволяє коду бачити файли, навіть якщо їхні розширення записані великими літерами. Фільтр Not FileObj.Name Like "~$*" відсікає тимчасові системні копії документів, які Windows створює автоматично при відкритті файлів користувачами.

Пошук меж таблиці реалізовано через команду SourceSheet.Cells(SourceSheet.Rows.Count, 1).End(xlUp).Row.

Вона визначає номер останнього заповненого рядка. Якщо дані є, вся таблиця одним кроком завантажується в оперативну пам'ять комп'ютера у вигляді динамічного масиву Variant: DataArray = wsSource.Range("A4:F" & SourceLastRow).Value. Подальша обробка та нормалізація даних (видалення пробілів функцією Trim та перевірка чисел через IsNumeric) виконується всередині цього масиву за допомогою циклу по межах UBound(DataArray, 1). Оскільки макрос працює безпосередньо з ОЗП, а не смикає кожну клітинку аркуша окремо, швидкість збору даних зростає в десятки разів. Після завершення перевірки масив вивантажується на підсумковий лист за допомогою методу Resize.

3.1.3. Алгоритми динамічного пошуку меж таблиць та нормалізації даних

Після того як файл звіту успішно просканово і відкрито у фоновому режимі, потрібно скопіювати з нього дані. Головна складність тут — визначити, де закінчується реальна таблиця. Використання властивості ActiveSheet.UsedRange.Rows.Count є ненадійним. Якщо користувач у файлі звіту випадково поставив пробіл у далекому рядку або просто змінив там формат клітинок, Excel починає вважати весь цей порожній простір використаним. Макрос копіює велику кількість пустих рядків, що призводить до роздування розміру головного файлу.

Для точного виявлення меж таблиці в код було закладено алгоритм сканування знизу вгору. Програма бере максимальний можливий рядок в Excel (константа Rows.Count, що дорівнює 1048576) у першому стовпчику і піднімається вгору до першої заповненої комірки:

Цей метод працює на рівні ядра Excel і чітко визначає номер останнього рядка з даними. Але виникає інша проблема: якщо аркуш у звіті виявився повністю порожнім, цей метод поверне значення 1. Щоб код не скопіював шапку таблиці замість даних, в алгоритм додано логічну перевірку: якщо ActualLastRow менше або дорівнює номеру рядка шапки (наприклад, рядок 3), то файл вважається порожнім, закривається, а програма переходить до наступного документа.

Наступний етап — нормалізація отриманих даних. Дані у вхідних звітах майже завжди містять дефекти:

- Текстові пробіли наприкінці назв компаній (через що назви "ТОВ Альфа" і "ТОВ Альфа " сприймаються як різні контрагенти);
- Використання різних символів дефісу;
- Помилки в написанні числових значень, коли замість системного роздільника дробів (наприклад, коми) вводиться крапка, що перетворює число на звичайний текст.

Для очищення текстових полів у коді реалізовано послідовну обробку функціями Trim() (прибирає крайні пробіли) та внутрішньою логікою заміни

подвійних пробілів на одинарні через цикл Do While InStr(Text, " ") > 0. Для числових полів застосовано перевірку IsNumeric. Якщо функція повертає False через неправильний роздільник, код виконує примусову заміну знаків Replace(Text, ".", ",") або навпаки, залежно від локальних налаштувань Windows.

3.2. Побудова міжпрограмного інтерфейсу COM з MS Word та управління процесами

Після формування зведеної таблиці в Excel починається етап генерації вихідних документів (наприклад, індивідуальних договорів). Для цього потрібно передати дані в програму MS Word. Оскільки це різні програмні продукти, взаємодія між ними реалізована за допомогою технології COM (Component Object Model).

Використання раннього зв'язування (Early Binding) через підключення бібліотек у меню References викликає проблеми сумісності на підприємствах, де на різних комп'ютерах стоять різні версії Office. Для уникнення конфліктів у проєкті реалізовано пізнє зв'язування (Late Binding). Всі змінні для роботи з Word оголошуються як базовий універсальний об'єкт Object:

```
Dim WordApp As Object
Dim WordDoc As Object
Set WordApp = CreateObject("Word.Application")
```

При такій схемі операційна система Windows під час виконання коду сама звертається до системного реєстру, шукає поточну встановлену версію програми Word і запускає її фоновий процес.

Під час тестів виявлено проблему: якщо Word уже був запущений користувачем на комп'ютері для особистих завдань, команда Create Object створює ще один, паралельний процес WINWORD.EXE в пам'яті. Якщо алгоритм працює в циклі й на кожному рядку таблиці створює новий об'єкт, то через 20-30 ітерацій відбувається надмірне накопичення фонових процесів та переповнення оперативної пам'яті.

Щоб ліквідувати цей дефект, в архітектуру модуля було закладено перевірку через перехоплення помилок:

```
On Error Resume Next
Set WordApp = GetObject(, "Word.Application")
If Err.Number <> 0 Then
    Set WordApp = CreateObject("Word.Application")
    Err.Clear
End If
On Error GoTo 0
```

Цей блок спочатку намагається підключитися до вже відкритого вікна Word через GetObject. Якщо програма закрита (виникає помилка Err.Number <> 0), код очищує помилку і тільки тоді створює новий процес. Такий підхід гарантує, що в системі буде працювати лише один процес Word. Після завершення роботи всього макросу викликається метод WordApp.Quit, а змінним присвоюється значення Nothing для повного звільнення пам'яті.

3.2.1. Аналіз модуля роботи з текстовими шаблонами

Створення документів у Word відбувається на основі файлу-шаблону. Замість нестабільних закладок (Bookmarks), які користувачі часто затирають при ручному редагуванні шаблонів, у даній роботі застосовано метод унікальних текстових тегів-маркерів. У тексті шаблону місця для вставки динамічних даних виглядають як унікальні комбінації символів, які не можуть зустрітися в звичайній мові, наприклад: [#Код_Контрагента#], [#Сума_Договору#].

Для заміни цих міток на дані з Excel використовується об'єкт Selection або Range всередині моделі Word і його метод Find.Execute. Проте стандартний запис макросу для пошуку й заміни працює лише в основному тілі документа. Під час тестування системи виявлено недолік: якщо текстові мітки стоять у верхньому чи нижньому колонтитулі сторінки (Headers/Footers) або всередині графічних написів (Textboxes), стандартний метод Find їх проскакує.

Для вирішення проблеми неповної заміни в код було вбудовано алгоритм перебору всіх областей документа (StoryRanges). Текст у Word ділиться на кілька незалежних шарів, і макрос має пройтися по кожному з них окремо:

```
Dim WordStory As Object
For Each WordStory In WordDoc.StoryRanges
    Do
        With WordStory.Find
            .Text = "[#Код_Контрагента#]"
            .Replacement.Text = ExcelDataValue
            .Forward = True
            .Wrap = 1 ' Константа wdFindContinue
            .Execute Replace:=2 ' Константа wdReplaceAll
        End With
        Set WordStory = WordStory.NextStoryRange
    Loop Until WordStory Is Nothing
Next WordStory
```

Цей перебір гарантує, що мітка буде замінена, де б вона не знаходилася — в основному тексті, у колонтитулах чи таблицях. Після завершення процесу заміни документ зберігається за допомогою методу SaveAs2 з динамічним формуванням імені файлу з даних Excel (наприклад, за шаблоном Код_Дата.docx).

Процедура GenerateWordDocuments у модулі M_WordGeneration відповідає за генерацію договорів на основі текстового шаблону Template.docx. Програмний код модуля M_WordGeneration подано в Додатку В.

Макрос підключається до текстового процесора через GetObject та CreateObject, залишаючи властивість WordApp.Visible = False, щоб інтерфейс програми Word залишався прихованим від користувача під час розрахунків.

Замість використання стандартних закладок, які часто зникають при ручному редагуванні файлів, у шаблоні розставлені звичайні текстові мітки в квадратних дужках, наприклад [#Контрагент#], [#Сума#]. Головна особливість цього модуля — використання циклу For Each WordStory In WordDoc.StoryRanges.

Текст у Word розділений на окремі шари (основне тіло, колонтитули, таблиці), і звичайний пошук бачить мітки тільки в основному тексті. Перебір усіх StoryRanges дозволяє виконати тотальну заміну тегів по всьому документу,

включаючи верхні та нижні колонтитули сторінок, де зазвичай вказують номери договорів або дати.

Заміна виконується методом Find.Execute з параметром Replace:=2, що відповідає команді «Замінити все». Форматування чисел та дат налаштовано безпосередньо в момент підстановки за допомогою функцій Format(..., "dd.mm.yyyy") та Format(..., "#,##0.00"). Готовий файл зберігається методом SaveAs2 під унікальною назвою, яка складається з коду компанії та номера договору. Після збереження рядок у таблиці Excel отримує статус Сформовано, що є сигналом для роботи поштового модуля.

3.2.2. Форматування табличних структур у документах Word

Окремою складною інженерною задачею при автоматизації офісного документообігу є перенесення даних, коли до одного договору потрібно прикріпити специфікацію товарів, де кількість рядків заздалегідь невідома. Звичайна текстова заміна тут не працює.

Для реалізації цієї задачі в коді було застосовано комбінований метод. У шаблоні Word створюється порожня таблиця-заготовка, яка містить лише один рядок — шапку з назвами стовпчиків ("№ п/п", "Назва товару", "Кількість", "Ціна"). Прямо під цією шапкою вставляється унікальний тег-маркер [#Старт_Таблиці#].

Макрос в Excel спочатку аналізує зведений реєстр і збирає всі товари, які належать конкретному контрагенту, у внутрішній масив. Потім код через COM-зв'язок знаходить тег [#Старт_Таблиці#] в Word, визначає індекс таблиці в колекції Tables і починає динамічно добудовувати структуру документа (рисунок 3.1).

```
Dim WordTable As Object
Set WordTable = WordDoc.Tables(TableIndex)

For RowCounter = 1 To TotalItemsCount

    WordTable.Rows.Add
    TargetWordRow = WordTable.Rows.Count


    WordTable.Cell(TargetWordRow, 1).Range.Text = RowCounter
    WordTable.Cell(TargetWordRow, 2).Range.Text = ArrayItems(RowCounter, 1)
    WordTable.Cell(TargetWordRow, 3).Range.Text = ArrayItems(RowCounter, 2)
Next RowCounter
```

Рисунок 3.1 – Приклад коду COM-зв'язок в Word

Після того як всі рядки додані й заповнені текстом, виникає проблема збитого форматування. Ширини стовпчиків стають неоднаковими, а межі таблиці відображаються некоректно. Для приведення таблиці до належного вигляду в код вбудовано форматування через об'єктну модель Word:

- Ввімкнення автопідбору ширини стовпчиків за вмістом: WordTable.AutoFitBehavior(1) (константа wdAutoFitContent);
- Вирівнювання тексту в числових стовпчиках по правому краю;
- Примусове встановлення шрифту (Times New Roman, розмір 11) для всієї створеної таблиці через властивість WordTable.Range.Font.Name, оскільки Word при додаванні нових рядків часто скидає налаштування на стандартний системний шрифт.

3.3. Інтеграція з об'єктною моделлю MS Outlook та управління поштовим сервером

Фінальний етап автоматизованого циклу — відправка створених документів отримувачам через корпоративну пошту. Взаємодія з поштовою службою реалізована через COM-підключення до додатка MS Outlook. Код створює об'єкт додатка Outlook.Application, а всередині нього — окремий поштовий об'єкт MailItem за допомогою команди CreateItem(0).

Електронні адреси макрос бере зі зведеного реєстру Excel. Для формування тіла листа використано властивість .HTMLBody. Це дало змогу розбити текст

повідомлення на абзаци тегами <p>, виділити важливі цифри жирним тегом та автоматично підключити корпоративний підпис компанії.

Головна проблема, яка виникла під час налаштування розсилки — це вбудована система безпеки Outlook. При спробі програмного доступу до адресної книги або при виклику методу .Send у фоновому режимі, система безпеки часто блокує потік коду і виводить на екран попередження про те, що стороння програма намагається отримати доступ до поштової скриньки. Це порушує автономність процесу, оскільки вимагає підтвердження користувача на кожному листі.

Для нейтралізації цього блокування в архітектуру поштового модуля було закладено метод повторного використання активного сеансу пошти (Active Session) та оптимізацію роботи через об'єкт ActiveInspector. Якщо макрос підключається до вже запущеного та авторизованого вікна Outlook, операційна система сприймає запити коду як легальні дії користувача і не викликає вікон блокування безпеки. (рисунок 3.2).

```
Dim OutlookApp As Object
Dim MailItem As Object

On Error Resume Next
Set OutlookApp = GetObject(, "Outlook.Application")
If OutlookApp Is Nothing Then
    Set OutlookApp = CreateObject("Outlook.Application")
End If
On Error GoTo 0

Set MailItem = OutlookApp.CreateItem(0)
With MailItem
    .To = ExcelEmailValue
    .Subject = "Документи згідно договору № " & ContractNumber
    .HTMLBody = "<html><body><p>Добрий день.</p><p>Надсилаємо документи у вкладенні.</p></body></html>"
    .Attachments.Add (FilePathOnDisk)
    .Send
End With
```

Рисунок 3.2. – Приклад коду для управління поштовим сервером

Після відправки листа макрос робить відмітку у зведеному реєстрі Excel: у стовпчик "Статус відправки" записується слово "Надіслано", що дозволяє уникнути повторного надсилання документів.

3.3.1. Аналіз модуля автоматичної розсилки пошти та логування

Потенціал VBA у Microsoft Outlook це автоматизація електронної пошти. VBA дозволяє автоматично надсилати листи, сортувати повідомлення, створювати шаблони листів, налаштовувати автообробку

За автоматичну доставку файлів отримувачам через програму MS Outlook відповідає модуль M_EmailSubsystem. Програмний код модуля подано в Додатку Г. Процедура SendOutEmails перебирає зведений реєстр Excel і створює нові листи за допомогою команди OutlookApp.CreateItem(0).

Для створення оформленого тексту листа використано властивість .HTMLBody. Вона дозволяє передавати текст у форматі HTML-коду зі вбудованими inline-стилями. Завдяки цьому в тілі листа налаштовано шрифти, зроблено абзаци за допомогою тегів <p> та виділено важливі фінансові показники жирним тегом <b>. Перед відправкою листа в коді обов'язково спрацьовує перевірка Dir(AttachmentPath) <> "". Вона перевіряє, чи дійсно файл договору фізично існує на жорсткому диску, і якщо файл є — прикріплює його через метод .Attachments.Add та відправляє командою .Send. Якщо файл відсутній, макрос фіксує помилку в логу, запобігаючи надсиланню пустих повідомлень.

Процедура LogSystemError у модулі M_EmailSubsystem відповідає за запис збоїв у текстовий файл Errors.txt. Вона використовує функцію FreeFile для отримання вільного номера файлового дескриптора операційної системи. Команда Open LogPath For Append As #FileNum відкриває файл у режимі дозапису рядків (новий текст додається в кінець документа, не затираючи старі записи). За допомогою команди Print у файл записується поточний час та опис винятку з об'єкта Err.Description, після чого дескриптор закривається командою Close.

3.4. Проєктування графічного інтерфейсу UserForms та програмування логіки подій

Для побудови зручного інтерфейсу системи було розроблено вікно коистувача на базі об'єкта UserForm. Головне завдання форми — повністю приховати від оператора внутрішню структуру таблиць та вікно редактора коду VBE. Користувач взаємодіє лише з елементами керування: текстовими полями, кнопками та списками.

При проєктуванні форми виникла проблема тимчасового блокування інтерфейсу під час тривалих обчислень. Коли макрос запускає цикл обробки багатьох файлів, операційна система сприймає Excel як програму, що не відповідає. При цьому порушується коректне перемальовування графічного інтерфейсу форми, елементи відображаються некоректно, а користувач втрачає контроль над процесом.

Для вирішення цієї проблеми в ядро форми було інтегровано динамічний прогрес-бар та команду DoEvents. На форму було додано два об'єкти типу Frame (один всередині іншого). Зовнішній фрейм виконує роль рамки, а внутрішній змінює свій розмір залежно від етапу виконання процесу.

Програмний код логіки динамічного оновлення інтерфейсу в процесі виконання ітераційного перебору подано на рисунку 3.3:

```
    Dim ProgressPercentage As Double
    ProgressPercentage = CurrentFileIndex / TotalFilesCount

    ' Змінюємо ширину внутрішнього фрейму залежно від прогресу
    UserForm1.FrameProgressInside.Width = ProgressPercentage *
UserForm1.FrameProgressOutside.Width
    UserForm1.LabelStatus.Caption = "Обробка документа: " &
Format(ProgressPercentage, "0%")

    ' Команда для оновлення графічного інтерфейсу
    DoEvents
```

Рисунок 3.3. – Програмний код логіки динамічного оновлення інтерфейсу

Команда DoEvents примусово перериває виконання макросу на кілька мікросекунд і передає керування операційній системі. За цей час Windows встигає оновити положення прогрес-бару та перевірити стан кнопок форми. Це усуває ефект зависання програми — вікно залишається інтерактивним протягом усього часу обчислень.

Крім того, на рівні елементів форми прописана логіка попередньої перевірки даних (Validation). Наприклад, для кнопки "Запустити" створено обробник події Click, який перед запуском основних модулів перевіряє умови за допомогою функцій Len(TextBox1.Text) = 0. Якщо шляхи до папок не вказані, макрос зупиняє роботу, видає вікно попередження і фокусує курсор на порожньому полі через метод TextBox1.SetFocus.

Керування діями користувача у графічному вікні програми здійснюється через форму UserForm1 Програмний код обробників подій форми керування UserForm1 подано в Додатку Д.

Процедура BtnBrowse_Click для вибору папки за допомогою об'єкта Application.FileDialog(msoFileDialogFolderPicker) викликає стандартне діалогове вікно Windows. Вибраний користувачем шлях автоматично записується в текстове поле форми Me.TxtInBoxPath.Text.

При натисканні на кнопку запуску BtnRun_Click макрос спочатку перевіряє довжину тексту в полі шляху за допомогою функції Len(). Якщо поле порожнє, виконання зупиняється, а користувач бачить попередження. Якщо перевірка пройшла успішно, макрос по черзі викликає головні модулі збору, генерації та розсилки документів.

Для підтримки візуального зв'язку з користувачем код динамічно змінює властивість Me.ProgressBarInside.Width, збільшуючи синю смужку індикатора прогресу на екрані залежно від поточного етапу роботи. Критично важливою тут є команда DoEvents. Вона примусово віддає квант часу процесора операційній системі Windows, завдяки чому інтерфейс форми встигає перемальовуватися прямо під час важких обчислень, кнопки залишаються активними, а вікно

програми не застигає. Після завершення всіх процедур команда Unload Me повністю видаляє форму з пам'яті.

3.5. Оптимізація швидкодії, технологія обробка помилок

При роботі з реальними реєстрами даних, обсяг яких перевищує кілька тисяч рядків, базовий код VBA починає демонструвати суттєве зниження продуктивності. Головна причина низької швидкості — це архітектурна особливість взаємодії мови з аркушами. Щоразу, коли макрос зчитує значення з клітинки, відбувається надмірна кількість звернень до графічної оболонки аркуша Excel. На великих циклах це створює затримки обчислень.

Для підвищення продуктивності системи в даній роботі реалізовано технологію віртуальних масивів пам'яті (Variant Arrays). Метод полягає в тому, що ми повністю відмовляємося від роботи з клітинками аркуша всередині циклів. Замість цього весь діапазон таблиці копіюється в оперативну пам'ять комп'ютера за один пасаж:

```
Dim MemoryBuffer As Variant
Dim LastRow As Long
LastRow = Range("A" & Rows.Count).End(xlUp).Row

' Одноразове зчитування всієї таблиці в ОЗП
MemoryBuffer = Range("A1:H" & LastRow).Value
```

Після виконання цієї команди створюється двовимірний масив MemoryBuffer, межі якого визначаються функціями LBound та UBound.

Подальший ітераційний перебір, очищення тексту та фільтрація відбуваються виключно всередині цього масиву в оперативній пам'яті:

```
Dim i As Long
For i = 2 To UBound(MemoryBuffer, 1)
    ' Обробка даних відбувається всередині масиву
    If MemoryBuffer(i, 3) = "Обробити" Then
        MemoryBuffer(i, 4) = Trim(MemoryBuffer(i, 4))
    End If
Next i
```

Робота з пам'яттю ОЗП здійснюється на рівні процесора і не викликає перемальовування графіки. Тести показали, що обробка великого реєстру за старою схемою (через об'єкти Cells) займала значний час, а після перенесення обчислень у масив Variant цей же обсяг інформації обробляється за кілька секунд. Після закінчення циклу підсумкові дані вивантажуються назад на аркуш: Range("A1:H" & LastRow).Value = MemoryBuffer.

Також на початку роботи головного макросу було додано команди для вимкнення фонових процесів додатку:

```
Application.ScreenUpdating = False ' вимикаємо оновлення екрану
Application.Calculation = xlCalculationManual ' вимикаємо автоматичний
    перерахунок формул
Application.EnableEvents = False ' вимикаємо відслідковування подій
```

В кінці коду ці налаштування повертаються в стан *True*, щоб відновити стандартний режим роботи програми.

Останнім етапом побудови системи є створення архітектури обробки помилок та ведення системних логів. У реальних умовах експлуатації макрос може зіткнутися із зовнішніми збоями: відсутність зв'язку з локальною мережею, заблоковані файли шаблонів або некоректний формат даних. Без належного захисту будь-яка така ситуація викликає зупинку програми з виведенням системного вікна дебаггера.

Для забезпечення відказостійкості у проєкті реалізовано структуровану систему перехоплення винятків за допомогою операторів On Error GoTo. Кожна окрема процедура має свій власний контур захисту та мітку обробника помилок(рисунок 3.4).

```
Sub GenerateDocuments()
    On Error GoTo DocumentErrorHandler

    ...
    Exit Sub

DocumentErrorHandler:

    Dim LogFileNumber As Integer
    LogFileNumber = FreeFile


    Open ThisWorkbook.Path & "\Logs\Error_Log.txt" For Append As #LogFileNumber
    Print #LogFileNumber, DateTime.Now & " | Файл: " & CurrentFile & " | Помилка № " & Err.Number & " | Опис: " & Err.Description
    Close #LogFileNumber


    Err.Clear
    Resume Next
End Sub
```

Рисунок 3.4 – Приклад коду для перехоплення винятків за допомогою операторів On Error GoTo

Використання оператора Resume Next всередині обробника помилок дозволяє реалізувати логіку безперервного виконання процесу. Якщо при обробці великого пакету звітів один файл виявився пошкодженим або заблокованим, макрос не зупиняє роботу всієї системи. Програма фіксує параметри збою в текстовому журналі, закриває пошкоджений документ без збереження, скидає стан об'єкта Err і переходить до обробки наступного документа за списком.

По завершенню успішно завершується виконання загального алгоритму, а користувачеві виводиться вікно сповіщення зі статистикою оброблених файлів та виявлених помилок, що потребують ручного втручання. Тестування системи на базі 40 тестових звітів підтвердило її стабільність та готовність до експлуатації в реальних умовах підприємства.

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексне дослідження теоретичних засад, архітектурних принципів та практичних методів автоматизації офісного документообігу, а також успішно розроблено й протестовано програмну систему консолідації звітності та автоматичного формування клієнтської документації на базі екосистеми Microsoft Office із використанням мови програмування Visual Basic for Applications (VBA).

На основі детального аналізу сучасного стану автоматизації діловодства було встановлено, що традиційні методи ручної обробки первинної інформації (копіювання реквізитів, ручне заповнення договорів та підготовка індивідуальних розсилок) є головним стримуючим чинником у підвищенні ефективності адміністративного персоналу. Ручна праця супроводжується значними часовими витратами та високим ризиком виникнення помилок через людський фактор, які в середньому складають 1–2% від загального обсягу введених даних. Автоматизація цих процесів дозволила перенести механічні операції на програмні алгоритми, залишивши за користувачем виключно функції контролю.

Проведений порівняльний аналіз існуючих інструментів автоматизації (великих корпоративних систем електронного документообігу СЕД, а також CRM та ERP-платформ) засвідчив, що для автоматизації локальних бізнес-процесів окремих підрозділів найдоцільнішим є використання вбудованого інструментарію пакету Microsoft Office та мови VBA. Такий підхід забезпечує мінімальну вартість розгортання, оскільки компанія використовує вже придбане офісне програмне забезпечення, а також гарантує максимальну гнучкість розробки та повну автономність системи від постійного інтернет-з'єднання чи доступності сторонніх хмарних серверів.

Під час дослідження технічних особливостей мови VBA було детально вивчено її архітектурні обмеження, такі як строга однопотоковість виконання коду та пряма залежність швидкодії від графічного інтерфейсу додатків-

господарів. Для запобігання блокуванню інтерфейсу Excel («Програма не відповідає») та усунення миготіння екрану під час масової обробки даних у проєкті впроваджено методи оптимізації. Замість покрокового звернення до клітинок аркуша реалізовано пакетне завантаження таблиць у динамічні Variant-масиви оперативної пам'яті. Додатково застосовано примусове керування глобальними налаштуваннями процесу Excel через відключення оновлення екрану (ScreenUpdating = False) та автоматичного перерахунку формул (Calculation = xlCalculationManual) під час виконання розрахунків.

У рамках проєктного етапу було розроблено та практично реалізовано архітектуру програмного комплексу, що складається із взаємопов'язаних модулів та інтерактивного графічного інтерфейсу користувача (UserForm). Створена форма забезпечує зручний вибір вхідних директорій, динамічне відображення етапів роботи за допомогою прогрес-бару та повну ізоляцію оператора від прямої взаємодії з кодом програми, що суттєво знижує вимоги до кваліфікації кінцевого користувача.

Розроблений програмний комплекс послідовно вирішує три ключові завдання бізнес-процесу підприємства:

1. **Модуль імпорту (M_DataImport):** забезпечує автоматичний перебір усіх файлів звітів у папці InBox, очищення даних від зайвих пробілів функцією Trim, перевірку числових форматів та формування єдиного зведеного реєстру на аркуші MasterRegistry.

2. **Модуль генерації документів (M_WordGeneration):** виконує підключення до текстового процесора Word, покроково зчитує реквізити контрагентів із реєстру та замінює відповідні динамічні мітки-теги (наприклад, [#Контрагент#], [#Сума#]) всередині корпоративного шаблону. Завдяки перебору всіх шарів документа (StoryRanges) заміна відбувається тотально, включаючи верхні та нижні колонтитули сторінок.

3. **Поштовий модуль (M_EmailSubsystem):** відповідає за інтеграцію з поштовим клієнтом MS Outlook, автоматичне створення електронних листів із красивою HTML-версткою тексту, перевірку фізичної наявності файлу на диску,

прикріплення створених договорів як вкладень та їх миттєве відправлення адресатам із фіксацією статусів у таблиці Excel.

Окрему увагу в роботі приділено етапу експериментального тестування та налагодження системи на реальних масивах даних, під час якого було виявлено та успішно ліквідовано кілька критичних вразливостей, що безпосередньо впливають на стабільність ПЗ у корпоративному середовищі:

- **Помилка імен файлів у Word (*This is not a valid file name*):** виявлено, що наявність символу косої риски (слешу /) у номерах договорів контрагентів блокує збереження документів на диск, оскільки ОС Windows сприймає цей знак як розподільник папок. Усунено шляхом інтеграції функції автоматичного заміщення заборонених символів на безпечний дефіс (-) перед записом файлу.
- **Помилка відсутності об'єкта Outlook (*Object variable not set*):** збій виникав на комп'ютерах користувачів, де десктопний додаток MS Outlook не налаштований або відсутній. Проблему вирішено впровадженням жорсткого логічного контуру перевірки стану COM-сервера. Замість аварійного завершення макрос тепер коректно зупиняє розсилку, виводить інформативне вікно-попередження для оператора (vbExclamation) та робить відповідний запис у текстовий лог помилок.
- **Помилка обробки порожніх рядків та порушення геометрії таблиць:** усунено збої імпорту, які виникали при випадковому вертикальному заповненні тестових файлів користувачами або наявності пустих рядків у кінці таблиць. Додано програмний захист, який перевіряє заповненість ключових реквізитів (Код та Номер договору) та автоматично ігнорує порожні чи некоректно оформлені рядки вхідних звітів.

Розрахунок операційної та економічної ефективності впровадженого програмного забезпечення підтвердив повне досягнення поставленої мети. Час обробки одного пакету документів скоротився з 10–15 хвилин (при ручному копіюванні) до кількох часток секунди. Повна ліквідація людського фактора на етапі консолідації та перенесення реквізитів забезпечила 100% точність вихідної

документації. Створена система логування помилок у файлі Errors.txt дозволяє здійснювати швидкий технічний аудит роботи програми без залучення сторонніх розробників.

Практична цінність роботи полягає в тому, що розроблений програмний комплекс є повністю завершеним, автономним і готовим до масштабування рішенням, яке може бути легко впроваджене в діяльність реальних підприємств для обліку й оптимізації первинного документообігу, зниження адміністративних витрат та стандартизації діловодства компанії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 4163:2020. Уніфікована система організаційно-розпорядчої документації. Вимоги до оформлювання документів. — К.: УкрНДНЦ, 2020. — 26 с.
2. Закон України «Про електронні документи та електронний документообіг» від 22.05.2003 № 851-IV (зі змінами та доповненнями). URL: https://zakon.rada.gov.ua/laws/show/851-15
3. Закон України «Про електронні ідентифікаційні послуги» від 05.10.2017 № 2155-VIII (зі змінами та доповненнями). URL: https://zakon.rada.gov.ua/laws/show/2155-19
4. Закон України «Про захист інформації в інформаційно-комунікаційних системах» від 05.07.1994 № 80/94-ВР. URL: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80
5. Коваль С. М. Автоматизація облікових процесів на підприємствах: теорія та практика // Вісник ЖДТУ. Серія: Економічні науки. — 2018. — № 2 (84). — С. 45–51.
6. Мартін Р. Чистий код. Створення, аналіз і рефакторинг програмного забезпечення / Пер. з англ. — Х.: Фабула, 2019. — 448 с.
7. Палеха Ю. І. Загальне діловодство: навч. посіб. — К.: Ліра-К, 2015. — 444 с.
8. Постанова Кабінету Міністрів України № 55 «Про затвердження Типової інструкції з документування управлінської діяльності в електронній формі та організації роботи з документами в діловодстві» від 17.01.2018. URL: https://zakon.rada.gov.ua/laws/show/55-2018-%D0%BF
9. Селіванова А. В. Informatsiyni systemy ta tekhnolohiyi v dilovodstvi ta dokumentoobihu pidpryyemstva // Сучасні технології управління. — 2021. — № 4. — С. 12–19.
10. Excel Easy: VBA Programming. Провідний європейський практичний ресурс із покроковими уроками проектування UserForms та оптимізації

макросів. URL: https://www.excel-easy.com/vba.html

11. GitHub (VBA Architecture Guidelines). Глобальна платформа репозиторіїв та аналізу відкритих кодів для дослідження стандартів оформлення офісних скриптів та ліквідації заплутаного коду. URL: https://github.com/topics/vba

12. Mansfield R. Mastering VBA for Microsoft Office 2019. — Indianapolis: Sybex, 2019. — 816 p.
13. Microsoft Learn: Excel Object Model Overview. Офіційне інженерне керівництво з архітектури та взаємодії головних об'єктів Excel (Application, Workbook, Worksheet, Range). URL: https://learn.microsoft.com/en-us/visualstudio/vsto/excel-object-model-overview
14. Microsoft Learn: Office VBA Reference. Офіційний технічний довідник та документація із синтаксису мови Visual Basic for Applications від компанії-розробника. URL: https://learn.microsoft.com/en-us/office/vba/api/overview/
15. Microsoft Learn: Outlook Object Model Overview. Технічне керівництво з автоматизації поштових повідомлень, вкладень (Attachments) та розсилок. URL: https://learn.microsoft.com/en-us/visualstudio/vsto/outlook-object-model-overview
16. Microsoft Learn: Word Object Model Overview. Офіційна документація з управління текстовими шарами (StoryRanges) та механізмами Find/Replace у Word. URL: https://learn.microsoft.com/en-us/visualstudio/vsto/word-object-model-overview
17. Stack Overflow (VBA Tag). Найбільша міжнародна англомовна спільнота програмістів для пошуку рішень щодо оптимізації Variant-масивів та обробки виняткових ситуацій. URL: https://stackoverflow.com/questions/tagged/vba
18. Walkenbach J., Alexander M. Excel 2019 Power Programming with VBA. — Indianapolis: John Wiley & Sons, 2019. — 736 p.

ДОДАТКИ

ДОДАТОК А.

Програмний код модуля налаштування параметрів даних M_Config

```
VBA
Option Explicit

' Глобальні змінні для збереження шляхів до папок системи
Public PathRoot As String
Public PathInBox As String
Public PathTemplates As String
Public PathOutBox As String
Public PathLogs As String

' Константи для індексації стовпчиків реєстру Excel (тип Long захищає від
Overflow)
Public Const COL_CONTRAGENT As Long = 1
Public Const COL_CODE As Long = 2
Public Const COL_CONTRACT_NUM As Long = 3
Public Const COL_DATE As Long = 4
Public Const COL_SUM As Long = 5
Public Const COL_EMAIL As Long = 6
Public Const COL_STATUS As Long = 7

Public Sub LoadConfig()
    Dim wsConfig As Worksheet
   On Error GoTo ErrHandler

    ' Визначаємо, де лежить сам файл Excel
    PathRoot = ThisWorkbook.Path
    Set wsConfig = ThisWorkbook.Worksheets("Config")

    ' Збираємо повні шляхи до робочих каталогів
    PathInBox = PathRoot & wsConfig.Range("B2").Value
    PathTemplates = PathRoot & wsConfig.Range("B3").Value
    PathOutBox = PathRoot & wsConfig.Range("B4").Value
    PathLogs = PathRoot & wsConfig.Range("B5").Value
        Exit Sub
ErrHandler:
    MsgBox "Помилка при зчитуванні параметрів конфігурації: " &
Err.Description, vbCritical, "Збій"
End Sub
```

ДОДОТОК Б.

Програмний код модуля імпорту та нормалізації звітів M_DataImport

```
VBA
Option Explicit

Public Sub ConsolidateReports()
    Dim FSO As Object
    Dim FolderInBox As Object
    Dim FileObj As Object
    Dim wbSource As Workbook
    Dim wsSource As Worksheet

    Dim DataArray As Variant
    Dim SourceLastRow As Long
    Dim MasterLastRow As Long
    Dim i As Long

    Call LoadConfig

    ' Вимикаємо екран і автоперерахунок формул, щоб код працював швидше
    Application.ScreenUpdating = False
    Application.Calculation = xlCalculationManual
    Application.EnableEvents = False

    Set FSO = CreateObject("Scripting.FileSystemObject")

    If Not FSO.FolderExists(PathInBox) Then
        MsgBox "Вказану папку вхідних звітів не знайдено: " & PathInBox, vbExclamation,
"Помилка шляху"
        GoTo AirExit
    End If

    Set FolderInBox = FSO.GetFolder(PathInBox)

    For Each FileObj In FolderInBox.Files
        ' Відсікаємо файли з іншим розширенням та приховані тимчасові файли Windows (~$)
        If LCase(FSO.GetExtensionName(FileObj.Path)) Like "xls*" And Not FileObj.Name Like "~$*"
Then

            Set wbSource = Workbooks.Open(Filename:=FileObj.Path, ReadOnly:=True,
UpdateLinks:=0)
            Set wsSource = wbSource.Worksheets(1)

            ' Шукаємо останній заповнений рядок знизу вгору
            SourceLastRow = wsSource.Cells(wsSource.Rows.Count, 1).End(xlUp).Row

            ' Якщо в таблиці є дані нижче шапки (рядок 3)
            If SourceLastRow > 3 Then
                ' Завантажуємо весь діапазон в масив Variant (робота в ОЗП)
                DataArray = wsSource.Range("A4:F" & SourceLastRow).Value

                MasterLastRow = ThisWorkbook.Worksheets("MasterRegistry").Cells(Rows.Count,
1).End(xlUp).Row + 1
```

```
            ' Прибираємо зайві пробіли та перевіряємо формат чисел
            For i = 1 To UBound(DataArray, 1)
                DataArray(i, 1) = Trim(DataArray(i, 1))
                DataArray(i, 2) = Replace(DataArray(i, 2), " ", "")

                If Not IsNumeric(DataArray(i, 5)) Then
                    DataArray(i, 5) = 0
                End If
            Next i

            ' Вивантажуємо весь масив на підсумковий аркуш одним махом
            ThisWorkbook.Worksheets("MasterRegistry").Cells(MasterLastRow,
1).Resize(UBound(DataArray, 1), UBound(DataArray, 2)).Value = DataArray
        End If

        wbSource.Close SaveChanges:=False
    End If
  Next FileObj

AirExit:
  ' Обов'язково вмикаємо назад стандартні налаштування Excel
  Application.ScreenUpdating = True
  Application.Calculation = xlCalculationAutomatic
  Application.EnableEvents = True
End Sub
```

ДОДАТОК В.

Програмний код модуля створення документів Word за шаблоном M_WordGeneration

VBA

```
Option Explicit

Public Sub GenerateWordDocuments()
    Dim WordApp As Object
    Dim WordDoc As Object
    Dim WordStory As Object
    Dim wsMaster As Worksheet

    Dim RegistryData As Variant
    Dim LastRow As Long
    Dim i As Long
    Dim TemplatePath As String
    Dim SaveFolder As String

    Call LoadConfig

    Set wsMaster = ThisWorkbook.Worksheets("MasterRegistry")
    LastRow = wsMaster.Cells(wsMaster.Rows.Count, 1).End(xlUp).Row

    If LastRow < 2 Then Exit Sub
    RegistryData = wsMaster.Range("A2:G" & LastRow).Value
    TemplatePath = PathTemplates & "\Template.docx"

    ' Підключаємося до відкритого Word або створюємо новий процес (Late Binding)
    On Error Resume Next
    Set WordApp = GetObject(, "Word.Application")
    If WordApp Is Nothing Then
        Set WordApp = CreateObject("Word.Application")
    End If
    On Error GoTo ErrDoc

    WordApp.Visible = False

    For i = 1 To UBound(RegistryData, 1)
        ' Перевіряємо статус, щоб не робити копії вже надісланих документів
        If RegistryData(i, COL_STATUS) <> "Надіслано" And RegistryData(i, COL_STATUS) <>
"Сформовано" Then

            Set WordDoc = WordApp.Documents.Open(TemplatePath, ReadOnly:=True)

            ' Перебираємо всі шари документа Word, щоб замінити мітки навіть у колонтитулах
            For Each WordStory In WordDoc.StoryRanges
                Do
                    With WordStory.Find
                        .ClearFormatting
                        .Replacement.ClearFormatting
```

```
                .Text = "[#Контрагент#]"
                .Replacement.Text = RegistryData(i, COL_CONTRAGENT)
                .Execute Replace:=2

                .Text = "[#Код#]"
                .Replacement.Text = RegistryData(i, COL_CODE)
                .Execute Replace:=2

                .Text = "[#Договір_№#]"
                .Replacement.Text = RegistryData(i, COL_CONTRACT_NUM)
                .Execute Replace:=2

                .Text = "[#Дата#]"
                .Replacement.Text = Format(RegistryData(i, COL_DATE), "dd.mm.yyyy")
                .Execute Replace:=2

                .Text = "[#Сума#]"
                .Replacement.Text = Format(RegistryData(i, COL_SUM), "#,##0.00")
                .Execute Replace:=2
            End With
            Set WordStory = WordStory.NextStoryRange
        Loop Until WordStory Is Nothing
    Next WordStory

    ' Формуємо назву файлу та зберігаємо його на диск
    SaveFolder = PathOutBox & "\Договір_" & RegistryData(i, COL_CODE) & "_" &
RegistryData(i, COL_CONTRACT_NUM) & ".docx"
    WordDoc.SaveAs2 SaveFolder
    WordDoc.Close SaveChanges:=False

    RegistryData(i, COL_STATUS) = "Сформовано"
  End If
Next i

' Повертаємо оновлені статуси на аркуш Excel
wsMaster.Range("A2:G" & LastRow).Value = RegistryData

ExitPoint:
  On Error Resume Next
  WordApp.Quit
  Set WordDoc = Nothing
  Set WordApp = Nothing
  Exit Sub
ErrDoc:
  Call LogSystemError("Помилка при створенні файлу Word на рядку " & i & ": " &
Err.Description)
  Resume ExitPoint
End Sub
```

ДОДАТОК Г. Програмний код модуля розсилки пошти та логування помилок M_EmailSubsystem

```
VBA
Option Explicit

Public Sub SendOutEmails()
    Dim OutlookApp As Object
    Dim MailObject As Object
    Dim wsMaster As Worksheet
    Dim MasterData As Variant
    Dim LastRow As Long
    Dim i As Long
    Dim AttachmentPath As String

    Call LoadConfig
    Set wsMaster = ThisWorkbook.Worksheets("MasterRegistry")
    LastRow = wsMaster.Cells(wsMaster.Rows.Count, 1).End(xlUp).Row

    If LastRow < 2 Then Exit Sub
    MasterData = wsMaster.Range("A2:G" & LastRow).Value

    On Error Resume Next
    Set OutlookApp = GetObject(, "Outlook.Application")
    If OutlookApp Is Nothing Then
        Set OutlookApp = CreateObject("Outlook.Application")
    End If
    On Error GoTo ErrMail

    For i = 1 To UBound(MasterData, 1)
        ' Беремо в роботу тільки файли зі статусом "Сформовано"
        If MasterData(i, COL_STATUS) = "Сформовано" Then

            AttachmentPath = PathOutBox & "\Договір_" & MasterData(i, COL_CODE) & "_" &
MasterData(i, COL_CONTRACT_NUM) & ".docx"
            Set MailObject = OutlookApp.CreateItem(0) ' 0 = olMailItem

            With MailObject
                .To = MasterData(i, COL_EMAIL)
                .Subject = "Документи за договором № " & MasterData(i, COL_CONTRACT_NUM)

                ' Оформляємо текст листа через звичайні HTML-теги
                .HTMLBody = "<html><body style='font-family: Calibri, Arial; font-size: 11pt;'>" & _
                            "<p>Добрий день.</p>" & _
                            "<p>Надсилаємо документи для компанії <b>" & MasterData(i,
COL_CONTRAGENT) & "</b>.</p>" & _
                            "<p>Сума до сплати: <span style='color: green;'><b>" & Format(MasterData(i,
COL_SUM), "#,##0.00") & " грн.</b></span></p>" & _
                            "<br><p>Лист створено автоматично поштовим модулем.</p>" & _
                            "</body></html>"

                ' Перевіряємо, чи файл договору дійсно є на диску, щоб не слати пустий лист
```

```
            If Dir(AttachmentPath) <> "" Then
                .Attachments.Add (AttachmentPath)
                .Send
                MasterData(i, COL_STATUS) = "Надіслано"
            Else
                Call LogSystemError("Файл договору не знайдено на диску: " & AttachmentPath)
                MasterData(i, COL_STATUS) = "Помилка файлу"
            End If
        End With
        Set MailObject = Nothing
    End If
  Next i

  ' Оновлюємо статуси в таблиці Excel
  wsMaster.Range("A2:G" & LastRow).Value = MasterData
  Exit Sub
ErrMail:
  Call LogSystemError("Критична помилка підключення до Outlook: " & Err.Description)
End Sub

' Проста функція для запису тексту помилок у файл логу Errors.txt
Public Sub LogSystemError(ByVal Msg As String)
  Dim FileNum As Integer
  Dim LogPath As String

  On Error Resume Next
  LogPath = ThisWorkbook.Path & "\Logs\Errors.txt"
  FileNum = FreeFile

  ' Відкриваємо файл у режимі дозапису (Append), щоб не затирати попередні рядки
  Open LogPath For Append As #FileNum
  Print #FileNum, Format(Now, "yyyy-mm-dd hh:mm:ss") & " | " & Msg
  Close #FileNum
End Sub
```

ДОДАТОК Д. Програмний код обробників подій форми керування UserForm1

```
Option Explicit
' Кнопка "Огляд" — викликає звичайне вікно вибору папки Windows
Private Sub BtnBrowse_Click()
    Dim FileDlg As FileDialog
    Set FileDlg = Application.FileDialog(msoFileDialogFolderPicker)

    With FileDlg
        .Title = "Виберіть папку з файлами звітів"
        .AllowMultiSelect = False
        If .Show = -1 Then
            Me.TxtInBoxPath.Text = .SelectedItems(1)
        End If
    End With
End Sub

' Головна кнопка запуску всієї автоматизації
Private Sub BtnRun_Click()
    ' Перевіряємо, чи користувач вибрав папку перед стартом
    If Len(Me.TxtInBoxPath.Text) = 0 Then
        MsgBox "Спочатку вкажіть шлях до папки з вхідними даними!", vbExclamation, "Помилка"
        Exit Sub
    End If

    ' Крок 1: Збирання даних
    Me.LblStatus.Caption = "Статус: Збирання звітів Excel..."
    Me.ProgressBarInside.Width = 60
    DoEvents ' Передаємо квант часу Windows, щоб синя смужка прогресу оновилася на екрані

    Call ConsolidateReports

    ' Крок 2: Робота з Word
    Me.LblStatus.Caption = "Статус: Створення документів у Word..."
    Me.ProgressBarInside.Width = 160
    DoEvents

    Call GenerateWordDocuments

    ' Крок 3: Розсилка пошти
    Me.LblStatus.Caption = "Статус: Розсилка через Outlook..."
    Me.ProgressBarInside.Width = 260
    DoEvents

    Call SendOutEmails

    Me.LblStatus.Caption = "Статус: Процес завершено"
```

```
    MsgBox "Автоматизація виконана успішно!", vbInformation, "Успіх"
    Unload Me
End Sub
```