МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) _Автоматизації і комп'ютерних систем_____
Кафедра _Інформаційних технологій, штучного інтелекту і кібербезпеки_

«До захисту в ЕК»		**«До захисту допущено»**	
Директор інституту(декан факультету)		Завідувач кафедри	
_____ (підпис)	___Андрій Форсюк__ (ім'я та прізвище)	_____ (підпис)	_Сергій Грибков__ (ім'я та прізвище)
«13» __лютого_____ 2023р.		«13» __лютого_____ 2023р.	

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

зі спеціальності__122 «Комп'ютерні науки»_____
(код та назва спеціальності)
освітньо-професійної програми_«Комп'ютерний еколого-економічний моніторинг»
_____
на тему:__«Дослідження та розроблення системи розрахунку терміну розкладання композитних пакувальних матеріалів»

Виконав: здобувач_2_ курсу, групи КМ-2-4М

____Мельниченко Роман Васильович_____ _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник ___Самсонов Валерій Васильович_____ _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач_____
(підпис)

Київ - 2023р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) __Автоматизації і комп'ютерних систем_____
Кафедра __Інформаційних технологій, штучного інтелекту і кібербезпеки___
Освітній ступінь__магістр_____
Спеціальність__122 «Комп'ютерні науки»_____
(код і назва)
Освітньо-професійна програма____«Комп'ютерний___еколого-економічний моніторинг»_____
(назва)

ЗАТВЕРДЖУЮ
Завідувач
кафедри Інформаційних технологій, штучного інтелекту і кібербезпеки_

_____Грибков С.В.
"_11_" ____листопада____2022року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

_____Мельниченко Роману Васильовичу_____
(прізвище, ім'я, по батькові)
1. Тема роботи:_«Дослідження та розроблення системи розрахунку терміну розкладання композитних пакувальних матеріалів»_____
керівник роботи_____ Самсонов Валерій Васильович, к.т.н. професор_,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від 11 листопада 2022року №820-кс
2. Строк подання здобувачем роботи_____20.01.2023 року_____
3. Вихідні дані до роботи _____Композитні матеріали, терміни розкладу___
_____
_____
_____
_____
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)___системний аналіз предметної області, постановка задачі, актуальність теми, опис процесу, опис реалізації._____
_____
_____
_____
_____
5. Перелік графічного матеріалу
_____фрагменти_коду_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	К.т.н. професор Самсонов В.В.		
2	К.т.н. професор Самсонов В.В.		
3	К.т.н. професор Самсонов В.В.		

7. Дата видачі завдання_____11 листопада 2022 року_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Перший етап - аналіз предметної області	11.11.22-28.11.22	Виконано
2	Другий етап – проектування загальної структури системи	29.11.22-03.01.23	Виконано
3	Третій етап – підбір програмних продуктів для реалізації системи та її реалізація	04.1.22-10.01.23	Виконано
4	Четвертий етап – оформлення магістерської роботи	11.01.23-19.01.23	Виконано

Здобувач _____ __Мельниченко Р.В.___
(підпис) (прізвище та ініціали)

Керівник роботи _____ __Самсонов В.В._____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Робота складається з 67 сторінок тексту, 6 рис., 7 табл, 57 джерел, 1 додаток.

Об’єкт дослідження: технологія забезпечення підтримки екологічної рівноваги у сфері переробки відходів за оптимізаційною моделлю.

Предмет дослідження: система для розрахунку терміну повного розкладання композитних пакувальних матеріалів.

Мета роботи: теоретично обґрунтувати, розробити та впровадити систему для розрахунку терміну повного розкладання композитних пакувальних матеріалів.

Перший розділ роботи присвячено аналізу основних питань предметної області.

Другий розділ роботи містить в собі огляд основних інструментальних засобів розробки та проектування загальної структури системи.

Третій розділ роботи орієнтований на проектування і розробку системи, через розроблення поведінкових сценаріїв, варіантів діяльності, побудову внутрішньої структури, розробку і тестування.

КЛЮЧОВІ СЛОВА: ПРОГРАМНИЙ КОМПЛЕКС, СИСТЕМНИЙ АНАЛІЗ, ПОРІВНЯЛЬНИЙ АНАЛІЗ, ВИМОГИ, ПРОГРАМА.

SUMMARY

The work consists of 67 pages of text, 6 figures, 7 tables, and 57 sources.

The object of the research: the technology of ensuring the maintenance of ecological balance in the field of waste processing according to the optimization model.

The subject of research: a system for calculating the term of complete decomposition of composite packaging materials.

The purpose of the work: to theoretically substantiate, develop and implement a system for calculating the term of complete decomposition of composite packaging materials.

The first section of the work is devoted to the analysis of the main issues of the subject area.

The second section of the work contains an overview of the main instrumental means of developing and designing the overall structure of the system.

The third section of the work is focused on the design and development of the system, through the development of behavioral scenarios, activity options, construction of the internal structure, development and testing.

KEY WORDS: SOFTWARE COMPLEX, SYSTEM ANALYSIS, COMPARATIVE ANALYSIS, REQUIREMENTS, PROGRAM.

ЗМІСТ

ВСТУП

В сучасному світі все більш активно набирає обертів процес діджиталізації і інтегрування все більшої кількості програмних продуктів в життя рядового члену соціуму.

Кожного дня з’являються нові додатки, сайти, тощо, які спрощують життя людини в усіх сферах.

Не оминув прогрес і сферу екології і збереження природи, створюються різні сервіси, які допомагають в екологічному моніторингу, покращенні стану природи, тощо.

На базі актуальності описаних вище тематик було прийнято рішення про їх об’єднання з метою створення системи розрахунку строку розкладання композитних пакувальних матеріалів на базі сировини, яку вони містять в складі.

Мета роботи: теоретично обґрунтувати, розробити та впровадити систему для розрахунку терміну повного розкладання композитних пакувальних матеріалів.

Об’єкт дослідження: технологія забезпечення підтримки екологічної рівноваги у сфері переробки відходів за оптимізаційною моделлю.

Предмет дослідження: система для розрахунку терміну повного розкладання композитних пакувальних матеріалів.

Відповідно до предмету та мети дослідження визначено наступні **задачі**:

- Аналіз теоретичних підходів та стан наукових досліджень систем екологічної рівноваги у сфері переробки відходів..
- Розробка моделі системи для забезпечення підтримки екологічної рівноваги у сфері переробки відходів за оптимізаційною моделлю.
- З’ясування особливостей використання програмно-апаратних засобів системи для забезпечення підтримки екологічної рівноваги у сфері переробки відходів.
- Наукове обґрунтування, розробка та експериментальна перевірка технології розробки системи для забезпечення підтримки екологічної рівноваги у сфері переробки відходів.

Методи дослідження: порівняння, систематизація, класифікація, узагальнення, пошуковий, аналітичний, статистичний.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття переробки відходів

Переробка — це процес перетворення відходів на нові матеріали та предмети. Концепція зазвичай передбачає відновлення енергії з відходів. Обробка матеріалу залежить від його здатності відновлювати властивості первісного стану. [1] Це альтернатива «традиційній» утилізації відходів, економія матеріалів і скорочення викидів парникових газів. Це також запобігає відходам потенційно корисних матеріалів і зменшує споживання свіжої сировини за рахунок зменшення споживання енергії, забруднення повітря (від спалювання) і забруднення води (від звалищ).

Переробка є ключовим компонентом сучасного скорочення відходів і є третім компонентом ієрархії «зменшення, повторного використання та переробки». [2] [3] Він сприяє екологічній стійкості шляхом видобутку сировинних ресурсів і перенаправлення викидів відходів в економічну систему. [4] Існують стандарти ISO щодо переробки, наприклад ISO 15270:2008 для пластикових відходів і ISO 14001:2015 для екологічного контролю переробки.

Матеріали, які можна переробити, включають багато видів скла, паперу, картону, металу, пластику, шин, текстилю, акумуляторів та електроніки. Компостування та повторне використання інших біорозкладаних відходів, таких як харчові та садові відходи, також є формою переробки. [5] Відходи для вторинної переробки відправляються в домашні центри переробки або збираються з баків біля бордюрів, де їх сортують, очищають і переробляють у нові матеріали для виробництва нових продуктів

В ідеалі переробка матеріалів забезпечує свіжу поставку того самого матеріалу — наприклад, використаний офісний папір стане новим офісним папером, а використаний пінополістирол стане новим пінополістиролом. Певні типи матеріалів, наприклад металеві банки, можна переробляти знову і знову нескінченно довго, не втрачаючи їхньої чистоти. [6] Для інших матеріалів це часто важко або занадто дорого (порівняно з виробництвом того самого продукту із сировини чи

інших джерел), тому «переробка» багатьох продуктів і матеріалів передбачає їх повторне використання у виробництві іншого матеріалу (наприклад, картон). Іншою формою вторинної переробки є відновлення складових матеріалів зі складних продуктів, які мають власну цінність (наприклад, свинець в автомобільних акумуляторах і золото в друкованих платах) або небезпечні властивості (наприклад, видалення та повторне використання ртуті в термометрах і термостатах).

Постачання

Для того, щоб програми переробки працювали, велике стабільне постачання перероблених матеріалів має вирішальне значення. Для виготовлення таких матеріалів використовуються три законодавчі варіанти: обов'язковий збір вторинної сировини, законодавство про зберігання тари та заборона на сміття. Закони про обов'язковий збір встановлюють цілі переробки для міст, як правило, у вигляді певного відсотка матеріалів, які мають бути спрямовані з міського потоку відходів до цільової дати. Місто зобов'язане працювати над досягненням цієї мети. [5]

Законодавство щодо складської тари передбачає повернення коштів за певну тару (як правило, скляну, пластикову та металеву). При купівлі продуктів у таких контейнерах існує невелика доплата, яку споживачі можуть повернути після повернення контейнера в пункт прийому. Цим програмам вдалося створити середній рівень відновлення 80%. [34] Незважаючи на ці сприятливі наслідки, перенесення витрат на збір від місцевих органів влади до промисловості та споживачів викликало сильну опозицію в деяких областях [5] - наприклад, виробники відповідають за переробку своєї продукції. У Європейському Союзі Директива WEEE вимагає від виробників побутової електроніки компенсувати переробникам. [35]

Ще один спосіб збільшити пропозицію вторинної сировини — заборонити утилізацію певних матеріалів як відходів, зокрема відпрацьованого масла, старих акумуляторів, шин і садового сміття. Це може створити життєздатну економію для належної утилізації продукту. Необхідно подбати про те, щоб забезпечити достатню кількість служб утилізації для задоволення пропозиції, інакше такі заборони можуть призвести до збільшення незаконного звалища. [5]

На замовлення уряду

Чотири форми законодавства також використовувалися для збільшення та підтримки попиту на перероблені матеріали: мінімальні вимоги до переробленого вмісту, стандарти використання, закупівельна політика та маркування перероблених продуктів [5].

Мінімальні вимоги як до вмісту вторинної сировини, так і до утилізації підвищили попит, що змусило виробників інтегрувати переробку у свою діяльність. У вимогах до змісту зазначено, що певний відсоток нових продуктів має складатися з перероблених матеріалів. Утилізація є більш гнучким варіантом: галузь може досягти своїх цілей утилізації на будь-якому етапі своєї діяльності та навіть укладати контракти на переробку в обмін на ринкові кредити. Противники цих підходів зазначають, що вони значно підвищують вимоги до звітності та стверджують, що позбавляють галузь гнучкості. [5] [36]

Уряди використовують свою купівельну спроможність, щоб збільшити попит на переробку через «політику закупівель». Ці політики є або «фіксованою партією», яка відкладає певні витрати на перероблені продукти, або програмою «цінової переваги», яка передбачає більший бюджет для закупівлі вторинної сировини. Інші правила можуть бути специфічними: у Сполучених Штатах, наприклад, Агентство з охорони навколишнього середовища стверджує, що нафту, папір, шини та будівельну ізоляцію слід купувати з перероблених або перероблених джерел, коли це можливо. [5]

Останньою державною постановою для збільшення попиту є маркування продуктів переробки. Споживачі можуть зробити більш обґрунтований вибір, коли від виробників вимагається вказувати на своїй упаковці кількість переробленого матеріалу, який вона містить, включаючи упаковку. Споживачі з достатньою купівельною спроможністю можуть вибрати екологічніші варіанти, заохочуючи виробників збільшувати кількість перероблених матеріалів у своїх продуктах і збільшуючи попит. Стандартизоване маркування вторинної переробки також може позитивно вплинути на переробку, якщо воно визначає, як і де продукти можуть бути перероблені. [5]

Якість перероблених відходів

Якість перероблених відходів є однією з головних проблем у реалізації довгострокового бачення зеленої економіки та досягнення нульових відходів. Загалом це стосується того, яка частина його складу становить цільовий матеріал порівняно з нецільовим матеріалом та іншими матеріалами, які не підлягають переробці. [39] Сталь та інші метали мають вищу якість вторинної переробки; за оцінками, дві третини всієї нової сталі походить із вторинної сталі [40]. Тільки цільові матеріали мають потенціал для переробки, тому більше нецільових матеріалів і матеріалів, які не підлягають переробці, зменшують кількість перероблених продуктів. [39] Велика частка невикористаного матеріалу, який не підлягає повторній переробці, може ускладнити досягнення «високоякісної» переробки; якщо перероблений матеріал має низьку якість, він, швидше за все, потрапить на повторну переробку або, в більш крайніх випадках, в інше. варіанти переробки або відправлення на звалище. [39] Наприклад, щоб полегшити відновлення виробництва прозорого скла, існували суворі обмеження щодо кількості кольорового скла, яке надходить у процес переплавлення. Іншим прикладом є переробка пластику, коли такі продукти, як пластикова харчова упаковка, часто переробляються в продукти нижчої якості, а не в ту саму пластикову харчову упаковку.

Якість перероблених матеріалів не тільки сприяє високоякісній переробці, але також може принести значні переваги для навколишнього середовища завдяки зменшенню, повторному використанню та уникненню вивезення продукції на звалище. [39] Якісна переробка може сприяти економічному зростанню шляхом максимізації вартості відходів. [39] Більш високі доходи від продажу високоякісної вторинної сировини можуть генерувати значну віддачу від вартості для місцевих органів влади, домогосподарств і підприємств [39]. Прихильність до високоякісної переробки також може підвищити довіру споживачів і бізнесу до сектору управління відходами та ресурсами та заохотити інвестиції в нього.

У ланцюжку постачання вторинної сировини існує багато видів діяльності, кожна з яких впливає на якість вторинної переробки. [41] Виробники відходів, які поміщають нецільові відходи та відходи, що не підлягають повторній переробці, у збірники переробки, можуть вплинути на якість кінцевого потоку переробки та

вимагати Подальші зусилля докладаються для утилізації цих матеріалів на наступних стадіях процесу переробки [41]. Різні системи збору призводять до різного ступеня забруднення. Коли кілька матеріалів збираються разом, потрібні додаткові зусилля для сортування їх на окремі потоки, що значно знижує якість кінцевого продукту [41]. Транспортування та ущільнення матеріалів також може ускладнити ситуацію. Незважаючи на вдосконалення технологій переробки та якості, сортувальні установки все ще не на 100% ефективні для розділення матеріалів. [41] Коли матеріали зберігаються на відкритому повітрі, вони можуть намокнути, що також може спричинити проблеми для процесорів. Можливо, знадобляться додаткові етапи сортування, щоб задовільно зменшити кількість марного матеріалу, який не підлягає переробці. [41]

Відходи – речовини або предмети, що утворюються під час виробництва, виконання робіт, надання послуг чи споживання, переробляються, утилізуються чи зберігаються.

Загальна інформація

Протягом 20 століття кількість відходів виробництва та споживання зростала настільки швидко, що їх утворення стало важливою проблемою для великих міст і великих промислових підприємств.

З точки зору природничих наук, теоретично будь-яка речовина може бути використана тим чи іншим способом. Природною межею використання є економічна доцільність використання.

Виникнення відходів (сміття) як явища легко пояснити з точки зору теорії управління. Відходи виникають, коли людина перестає поводитися з непотрібними їй матеріальними об’єктами (викидає їх), і ці об’єкти переходять в автономний режим — сміття починає розкидатися і повільно розкладатися.

Причини та доцільність вторинної переробки відходів

- Ресурси багатьох матеріалів Землі обмежені і може бути заповнені у терміни, порівнянні з часом існування людської цивілізації.
- Потрапивши у довкілля, матеріали зазвичай стають забруднювачами.

Загальні положення

Якщо говорити взагалі про відходи виробництва чи споживання по-сучасному, то в першу чергу варто сказати про тверді побутові відходи. Хоча під цією назвою в міжнародній і вітчизняній практиці розуміють не тільки тверді речовини, а й смолисті, пастоподібні, емульсійні та суспензійні речовини, а також речовини в рідкому та порошкоподібному агрегатних станах. [3]

Відходи виробничої діяльності називають промисловими (техногенними).

Сміття, що утворюється у сфері споживання людиною, відноситься до побутових відходів.

Загальна кількість світового сміття становить майже 800 мільярдів тонн, з яких тверді відходи перевищують 300 мільярдів тонн.

В Україні на початку 2000-х років питоме навантаження твердих побутових відходів, головним чином у гірських регіонах Дніпра та Донбасу, сягало 8-18 тис./км², порівняно з середнім показником по країні близько 3000/км².

Важливою особливістю структури відходоутворення України щодо сировинної орієнтації економіки є домінування в її складі відходів гірничодобувної промисловості (88%), тоді як частка відходів інших галузей становить близько 10%, тоді як побутові не перевищує 2%. [3]

1.2 Аналіз сучасних наукових публікацій

У 1986 році Всесвітня організація охорони здоров'я створила перелік правил і загальних принципів обробки та утилізації небезпечних відходів [50]. Цей збірник правил і основних принципів став основою для створення систем утилізації відходів у всьому світі.

Вільного доступу до наукових публікацій, пов’язаних з автоматизацією процесів у контексті підтримки екологічної рівноваги, не існує, оскільки сьогодні в більшості країн актуальнішою є проблема впровадження процесів переробки відходів, ніж їх автоматизація.

За підрахунками екологів, кожна людина «генерує» 200 кілограмів сміття на рік (найоптимістичніші підрахунки), з яких лише 40-50 кілограмів можна переробити. Отже, можна зробити висновок, що більшість публікацій, пов’язаних з переробкою та розкладанням відходів, безпосередньо присвячені впровадженню систем сортування та переробки відходів, а проблема автоматизації цього процесу взагалі не досліджена.

1.3 Дослідження інформаційного забезпечення для автоматизації процесу підрахунку строку розкладання відходів з метою подальшої переробки

З точки зору реалізації, найпростішим способом є автоматизація процесу шляхом створення програми, яка може агрегувати та обробляти певні дані, наприклад, дані розкладу різних матеріалів мультиупаковки та його вихід у розкладі.

Оскільки обраний метод впровадження включає розробку прикладного програмного забезпечення, концепцію прикладного програмного забезпечення та метод його розробки слід розглядати як частину загального обсягу робіт.

Програма (скорочено «застосунок» або «застосунок») — це комп'ютерна програма, призначена для виконання конкретного завдання, відмінного від роботи самого комп'ютера [1], зазвичай кінцевим користувачем. [2] Приклади включають текстові процесори, медіаплеєри та бухгалтерське програмне забезпечення. Збірні іменники вживаються всі разом. [3] Іншими основними категоріями програмного забезпечення є системне програмне забезпечення та службове програмне забезпечення («послуги»), пов'язані з роботою комп'ютера.

Програми можуть входити в комплект поставки комп'ютера та його системного програмного забезпечення або розповсюджуватися окремо, і можуть бути закодовані як пропрієтарні, з відкритим вихідним кодом або як проект. [4] Термін «програма» зазвичай відноситься до програм для мобільних пристроїв, таких як телефони.

термін

В інформаційних технологіях додаток, прикладна програма або прикладне програмне забезпечення — це комп'ютерна програма, розроблена для допомоги людям у виконанні певної діяльності. Залежно від діяльності, для якої вона призначена, програма може обробляти текст, числа, аудіо, графіку та комбінації цих елементів. Деякі програмні пакети зосереджені на одному завданні, наприклад на обробці тексту; інші, які називаються інтегрованим програмним забезпеченням, включають кілька програм. [5]

Написане користувачем програмне забезпечення адаптує систему до конкретних потреб користувача. Написане користувачем програмне забезпечення включає шаблони електронних таблиць, макроси текстового процесора, наукове

моделювання, аудіо, графіку та сценарії анімації. Навіть фільтри електронної пошти є програмним забезпеченням користувача. Користувачі створюють це програмне забезпечення самі, часто не усвідомлюючи його важливості.

Однак відмінність між системним програмним забезпеченням, таким як операційна система та прикладне програмне забезпечення, є неточним і іноді суперечливим [6]. Наприклад, одним із ключових питань у справі США проти Microsoft щодо антимонопольного законодавства є те, чи є веб-браузер Microsoft Internet Explorer частиною операційної системи Windows чи окремою програмою. Як інший приклад, суперечка щодо іменування GNU/Linux частково виникає через розбіжності щодо зв'язку між ядром Linux і операційною системою, побудованою на основі цього ядра. У певних типах вбудованих систем програмне забезпечення та ПЗ Користувачі можуть не розрізняти операційні системи, наприклад програмне забезпечення, яке використовується для керування відеомагнітофоном, DVD-програвачем або мікрохвильовою піччю. Наведене вище визначення може виключати деякі програми, які можуть існувати на деяких комп'ютерах у великих організаціях.

Метонімія

Термін «додаток», який використовується як прикметник, не обмежується значенням «належить або стосується прикладного програмного забезпечення» [7]. Наприклад, такі поняття, як інтерфейси прикладного програмування (API), сервери додатків, віртуалізація додатків, керування життєвим циклом додатків і портативні додатки однаково застосовуються до всіх комп'ютерних програм, а не лише до прикладного програмного забезпечення.

Класифікація

Існує багато різних способів класифікації програмного забезпечення.

З юридичної точки зору програмні додатки насамперед класифікують права своїх кінцевих користувачів або передплатників (можливо, із проміжними та багаторівневими рівнями підписки) за допомогою підходу «чорної скриньки».

Програмне забезпечення також класифікується за мовою програмування, на якій написаний або виконується їхній вихідний код, а також за їх призначенням і результатом.

на правах власності та користування

Прикладне програмне забезпечення зазвичай поділяється на дві широкі категорії: програмне забезпечення із закритим вихідним кодом проти програмного забезпечення з відкритим вихідним кодом, а також безкоштовне програмне забезпечення та пропрієтарне програмне забезпечення.

Власне програмне забезпечення захищено виключно авторським правом, і ліцензія на програмне забезпечення надає обмежені права на використання. Принцип «відкрито-закрито» стверджує, що програмне забезпечення може бути «відкритим лише для розширення, а не для модифікації». Такі програми можуть отримувати лише доповнення від сторонніх розробників.

Безкоштовне програмне забезпечення з відкритим вихідним кодом можна запускати, розповсюджувати, продавати або розширювати для будь-яких цілей, і його потрібно змінювати або змінювати таким же чином, коли воно відкрито.

Програмне забезпечення FOSS, випущене за безкоштовною ліцензією, може бути безстроковим або безкоштовним. Власник, власник або третя сторона, яка здійснює правозастосування будь-якого права (авторського права, торговельної марки, патенту чи права реального права) може мати право додавати винятки, обмеження, терміни дії або терміни дії до Ліцензійних умов використання.

Програмне забезпечення загального користування є безкоштовним і відкритим або зарезервованим типом FOSS, яке можна запускати, розповсюджувати, модифікувати, модифікувати, перевидавати або створювати в похідних роботах без згадки про авторське право та, отже, відкликання. Її навіть можна продати без передачі комунальної власності іншим суб’єктам. Програмне забезпечення, що є суспільним надбанням, може бути випущено відповідно до юридичної (скасованої) ліцензійної заяви, яка забезпечує виконання цих умов протягом невизначеного періоду (довічного або назавжди).

використовувати мову кодування

Зі зростанням і майже повсюдним розповсюдженням Інтернету існували важливі відмінності між написанням веб-додатків за допомогою HTML, JavaScript та інших веб-технологій, які зазвичай вимагають підключення до Інтернету, і використанням веб-браузера. Більш традиційні рідні програми, написані будь-якою

мовою, доступні для певних типів комп'ютерів. У комп'ютерному співтоваристві точаться дебати щодо того, що веб-додатки замінюють рідні програми для багатьох цілей, особливо на мобільних пристроях, таких як смартфони та планшети. Веб-програми дійсно набувають популярності для деяких цілей, але переваги програм роблять їх навряд чи зникнуть найближчим часом, якщо взагалі зникнуть. Крім того, вони можуть доповнювати і навіть інтегрувати. [9][10][11]

За записом та виїздом

Аплікації також можна вважати горизонтальними або вертикальними. [12][13] Бічні програми є більш популярними та поширеними, тому що вони загального призначення, такі як текстові процесори або бази даних. Вертикальні програми — це нішеві продукти, розроблені для певного типу галузі, бізнесу чи підрозділу в організації. Інтегрований пакет намагатиметься впоратися з усіма можливими аспектами, такими як виробництво чи банківський персонал, бухгалтерія чи обслуговування клієнтів.

Існує багато типів прикладного програмного забезпечення:[14]

- Набір програм складається з кількох програм, об'єднаних разом. Зазвичай вони мають пов'язані функції, функції та інтерфейси користувача і можуть взаємодіяти один з одним, наприклад. відкривати файли один одного. Бізнес-додатки часто поставляються в комплектах, напр. Microsoft Office, LibreOffice та iWork, які об'єднують текстовий процесор, електронну таблицю тощо; але люкси існують для інших цілей, напр. графіка або музика.
- Корпоративне програмне забезпечення відповідає потребам процесів і потоків даних цілої організації в кількох відділах, часто у великому розподіленому середовищі. Приклади включають системи планування ресурсів підприємства, системи управління відносинами з клієнтами (CRM), механізми реплікації даних та програмне забезпечення для управління ланцюгом поставок. Департаментальне програмне забезпечення — це підтип корпоративного програмного забезпечення, зосередженого на менших організаціях або групах у великій організації. (Приклади включають управління витратами на відрядження та службу підтримки IT.)

- Програмне забезпечення інфраструктури підприємства надає загальні можливості, необхідні для підтримки систем корпоративного програмного забезпечення. (Приклади включають бази даних, сервери електронної пошти та системи для керування мережами та безпекою.)
- Платформа додатків як послуга (aPaaS) — це служба хмарних обчислень, яка пропонує середовища розробки та розгортання служб додатків.
- Програмне забезпечення для інформаційних працівників дозволяє користувачам створювати інформацію та керувати нею, часто для окремих проектів у відділі, на відміну від управління підприємством. Приклади включають управління часом, управління ресурсами, аналітичні інструменти, інструменти для співпраці та документації. Текстові процесори, електронні таблиці, клієнти електронної пошти та блогів, персональні інформаційні системи та окремі медіаредактори можуть допомогти у виконанні багатьох завдань інформаційного працівника.
- Програмне забезпечення для доступу до вмісту використовується переважно для доступу до вмісту без редагування, але може включати програмне забезпечення, яке дозволяє редагувати вміст. Таке програмне забезпечення задовольняє потреби окремих осіб і груп у споживанні цифрових розваг і опублікованого цифрового контенту. (Приклади включають медіапрогравачі, веб-браузери та довідкові браузери.)
- Освітнє програмне забезпечення пов’язане з програмним забезпеченням для доступу до вмісту, але має вміст або функції, адаптовані для використання викладачами чи студентами. Наприклад, він може надавати оцінки (тести), відстежувати прогрес за матеріалами або включати можливості спільної роботи.
- Програмне забезпечення для моделювання моделює фізичні або абстрактні системи для дослідницьких, навчальних або розважальних цілей.
- Програмне забезпечення для розробки медіа створює друковані та електронні засоби масової інформації для використання іншими, найчастіше в комерційних або освітніх умовах. Це включає графічне програмне забезпечення, програмне забезпечення для настільних видавництв, програмне

забезпечення для розробки мультимедіа, редактори HTML, редактори цифрової анімації, цифрові аудіо- та відеокомпозиції та багато інших.[15]

- Програмне забезпечення для розробки продуктів використовується при розробці апаратних і програмних продуктів. Це включає в себе комп'ютерне проектування (CAD), комп'ютерну інженерію (CAE), засоби редагування та компіляції комп'ютерної мови, інтегровані середовища розробки та інтерфейси програмістів прикладних програм.
- Програмне забезпечення для розваг може посилатися на відеоігри, заставки, програми для відображення фільмів або відтворення записаної музики та інші види розваг, які можна відчути за допомогою комп'ютерного пристрою.

за платформою

Програми також можна класифікувати за обчислювальною платформою, наприклад, настільні програми для певної операційної системи [16], мережу доставки (наприклад, хмарні обчислення та програми Web 2.0) або пристрій доставки (наприклад, мобільні програми для мобільних пристроїв).

Сама операційна система може вважатися прикладним програмним забезпеченням для виконання простих обчислень, вимірювань, візуалізації та обробки тексту, які не використовуються для керування обладнанням через інтерфейс командного рядка або графічний інтерфейс користувача. Це не включає прикладне програмне забезпечення, що входить до складу операційної системи, наприклад програмні калькулятори або текстові редактори.

1.4 Постановка задачі

Основна задача даної роботи полягає в створенні прикладного додатку, який забезпечить користувача можливістю розрахувати строк розкладання композитного пакувального матеріалу в залежності від його складових частин, побачити строк розкладання кожної окремої частини і матеріалу в цілому, як у вигляді тексту, так і у вигляді графічного представлення отриманих результатів.

Висновки до розділу 1

В першому розділі проведено огляд предметної області, який допоможе зрозуміти суть задачі, проведено аналіз наукових публікацій по обраній темі, в ході якого виявлена абсолютна недослідженість тематики.

Загалом, перший розділ служить для розуміння обраної тематики роботи, оцінки поточної ситуації в дослідженні проблеми і необхідних засобах, які будуть використовуватися в ході вирішення поставленої задачі.

РОЗДІЛ 2
РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДЛЯ РОЗРАХУНКУ СТРОКУ РОЗКЛАДАННЯ КОМПОЗИТНИХ ПАКУВАЛЬНИХ МАТЕРІАЛІВ

2.1 Загальний опис архітектури системи

Архітектура програмного забезпечення відноситься до базової структури програмної системи та дисциплін для створення таких структур і систем. Кожна структура містить елементи програми, зв’язки між ними, а також атрибути елементів і зв’язки. Архітектура програмної системи є метафорою, подібною до архітектури будівлі. Він слугує концептуальною основою для системи та проекту, що розробляється, описуючи те, що має виконати команда проекту.

Архітектура програмного забезпечення полягає в тому, щоб зробити фундаментальний структурний вибір, і зміна цих виборів після реалізації коштує дорого. Вибір архітектури програмного забезпечення включає в себе конкретні структурні параметри, отримані від можливостей розробки програмного забезпечення. Наприклад, системи, які керують ракетою-ракетою космічного човника, мають бути дуже швидкими та надійними. Тому вам потрібно вибрати відповідну мову обчислень у реальному часі. Крім того, для потреб надійності можна вибрати кілька резервних і незалежно створених копій програми та запустити ці копії на незалежному обладнанні під час перехресної перевірки результатів.

Документація архітектури програмного забезпечення полегшує спілкування між зацікавленими сторонами, фіксує ранні рішення щодо проектування високого рівня та дозволяє повторно використовувати компоненти дизайну в проектах.

Думки щодо архітектури програмного забезпечення неоднозначні:

Структура макросистеми: це стосується архітектури як абстракції програмного забезпечення високого рівня, що складається з набору обчислювальних компонентів і з’єднувачів, які описують взаємодію між цими компонентами.

Важливі речі – що завгодно: йдеться про те, що архітекторам програмного забезпечення доводиться мати справу з рішеннями, які мають значний вплив на систему та її зацікавлених сторін.

Що є основою для розуміння системи в її контексті

Те, що люди думають, важко змінити: оскільки архітектурне проектування відбувається на початку життєвого циклу програмної системи, архітектори повинні зосередитися на рішеннях, які «мають» бути правильними з першого разу. Дотримуючись цієї лінії мислення, коли незворотність архітектурного дизайну подолано, вони можуть стати неархітектурними проблемами.

Набір рішень щодо архітектурного проектування: архітектуру програмного забезпечення не слід розглядати просто як набір моделей або структур, а скоріше включати рішення, які призводять до цих конкретних структур, і обґрунтування, що стоїть за ними. Це розуміння призвело до важливих досліджень управління знаннями про архітектуру програмного забезпечення.

Немає чіткої різниці між архітектурою та дизайном програмного забезпечення та вимогами (див. відповідні поля нижче). Усі вони є частиною «ланцюжка намірів» від намірів високого рівня до деталей низького рівня.

Архітектура програмного забезпечення демонструє наступне [17]:

Кілька зацікавлених сторін: програмна система повинна обслуговувати багато зацікавлених сторін, таких як бізнес-менеджери, власники, користувачі та оператори. Усі ці зацікавлені сторони висловили стурбованість системою. Збалансування цих проблем і демонстрація способів їх вирішення є частиною проектування системи. Це означає, що архітектура передбачає вирішення різних проблем і зацікавлених сторін і є багатодисциплінарною за своєю природою.

Розділення проблем: відомий метод для архітекторів зменшити складність полягає в тому, щоб відокремити проблеми, які керують дизайном. Документація щодо архітектури показує, що всі проблеми зацікавлених сторін вирішуються шляхом моделювання та опису архітектури з різних точок зору, пов’язаних із проблемами різних зацікавлених сторін. Ці окремі описи називаються представленнями схеми

Основа на якості: Класичні підходи до проектування програмного забезпечення (такі як структурне програмування Джексона) визначаються бажаною функціональністю та потоком даних через систему, але сучасне розуміння полягає в тому, що архітектура програмної системи тісніше пов’язана з її якісними характеристиками, такими як як продуктивність відмовостійкості, зворотна сумісність, розширюваність, надійність, ремонтопридатність, зручність використання, безпека, зручність використання та інші подібні характеристики. Занепокоєння зацікавлених сторін часто перетворюються на вимоги до цих атрибутів якості, якіРізні вимоги називаються нефункціональними вимогами, нефункціональними вимогами, поведінковими вимогами або атрибутами якості.

Повторювані стилі: Подібно до архітектури будівель, дисципліна архітектури програмного забезпечення розробила стандартні способи вирішення нових проблем. Ці «стандартні методи» мають різні назви на різних рівнях абстракції. Загальними термінами для ітераційних рішень є архітектурний стиль, стратегія, еталонна архітектура та архітектурний шаблон.

Концептуальна цілісність: термін, введений Фредом Бруксом у «Міфічній людині на Місяці» для позначення того факту, що архітектура програмної системи представляє спільне бачення того, що вона має робити та як вона має це робити. Це бачення слід відокремити від його реалізації. Архітектор діє як «охоронець бачення», гарантуючи, що доповнення до системи узгоджуються з архітектурою, таким чином зберігаючи цілісність концепції.

Когнітивні обмеження: спостереження, яке вперше зробив комп’ютерний програміст Мелвін Конвей у статті 1967 року, що організації, які розробляють системи, змушені створювати проекти, які повторюють комунікаційні структури цих організацій. Що стосується концептуальної цілісності, то Фред Брукс представив цю тезу та цю ідею ширшій аудиторії, коли процитував цю статтю та цю ідею у своїй елегантній класиці «Міфічна людина на Місяці», назвавши це «законом Конвея».

Model-View-Controller (зазвичай відомий як MVC) — це шаблон проектування програмного забезпечення, який зазвичай використовується для розробки користувальницьких інтерфейсів, який розділяє відповідну логіку програми на три взаємопов’язані елементи. Це робиться для того, щоб відокремити внутрішнє

представлення інформації від того, як користувач подає та отримує її. Цей тип візерунка використовується для розробки макетів сторінок.

Ця модель, яка традиційно використовується для настільних графічних інтерфейсів користувача (GUI), стала популярною для розробки веб-додатків. Такі популярні мови програмування, як JavaScript, Python, Ruby, PHP, Java, C# і Swift, мають фреймворки MVC для розробки веб-або мобільних додатків прямо з вікна.

Модель

Основні компоненти сутруктури. Це динамічна структура даних програми, яка не залежить від інтерфейсу користувача. [5] Він безпосередньо керує даними, логікою та правилами програми.

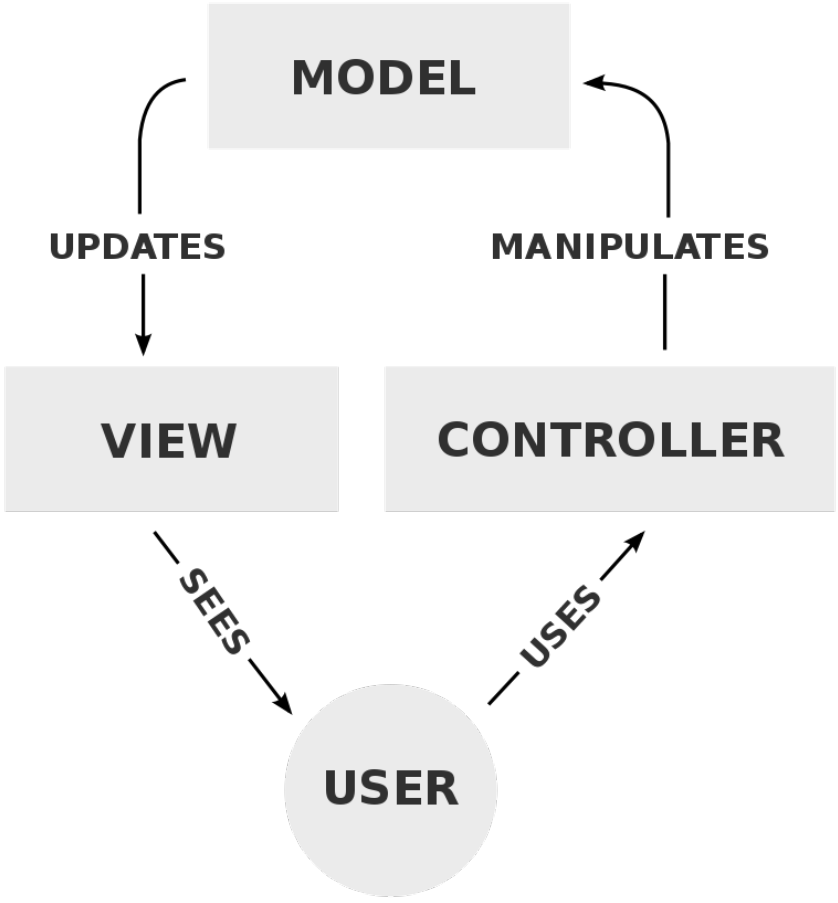

Рисунок 1.1 Діаграма роботи MVC

Вид

Будь-яке представлення інформації, наприклад діаграми, діаграми або таблиці. Можливі кілька переглядів однієї й тієї самої інформації, такі як гістограма для менеджменту та таблична таблиця для бухгалтерів.

Контролер

Приймає введення та перетворює його в команди для моделі чи перегляду.

На додаток до поділу програми на ці компоненти, модель модель-перегляд – контролер визначає взаємодію між ними.

- Модель відповідає за управління даними програми. Він отримує введення користувача від контролера.
- Перегляд означає представлення моделі у певному форматі.
- Контролер відповідає на введення користувача та здійснює взаємодію з об'єктами моделі даних. Контролер отримує вхід, додатково перевіряє його, а потім передає вхід моделі.

Як і інші моделі програмного забезпечення, MVC виражає «основне рішення» проблеми, дозволяючи налаштувати кожну систему. Конкретні конструкції MVC можуть суттєво відрізнятися від традиційних описів тут.

Незважаючи на те, що MVC спочатку був розроблений для настільних комп’ютерів, він отримав широке поширення як основна мова програмування для розробки додатків World Wide Web. Для реалізації цієї схеми було створено кілька веб-фреймворків. Ці програмні рамки інтерпретуються по-різному, головним чином через те, що обов’язки MVC розподілені між клієнтом і сервером.

Деякі веб-платформи MVC використовують підхід тонкого клієнта, розміщуючи майже всю логіку моделі, представлення та контролера на сервері. Це відображено в таких фреймворках, як Django, Rails і ASP.NET MVC. У цьому підході клієнт надсилає гіперпосилання або запит на надсилання форми до контролера, і повна та оновлена веб-сторінка (або інший документ) отримується з подання; модель повністю живе на сервері. Інші фреймворки, такі як AngularJS, EmberJS, JavaScriptMVC і Backbone, дозволяють компонентам MVC частково виконуватися на стороні клієнта.

Оскільки MVC роз’єднує різні компоненти програми, розробники можуть працювати над різними компонентами паралельно, не впливаючи та не блокуючи один одного. Наприклад, команда може розділити своїх розробників на зовнішніх і внутрішніх розробників. Розробники бекенда можуть проектувати структуру даних і взаємодію з ними без додаткового інтерфейсу користувача. Натомість розробники

передумов можуть спроектувати та перевірити макети додатків до того, як стануть доступними структури даних.

Ті самі (або схожі) представлення однієї програми можна відтворити для іншої програми з іншими даними, оскільки представлення стосуються лише того, як дані відображаються користувачеві. На жаль, це не працює, якщо цей код також можна використовувати для обробки даних користувача. Наприклад, код DOM (включно з його власною абстракцією програми) корисний як для графічного відображення, так і для введення користувачем. (Зверніть увагу, що незважаючи на назву Document Object Model, DOM насправді не є моделлю MVC, оскільки це інтерфейс користувача програми) [6].

Щоб вирішити ці проблеми, MVC (та подібні шаблони) часто поєднуються з архітектурою компонентів, яка забезпечує набір елементів інтерфейсу. Кожен елемент інтерфейсу є компонентом верхнього рівня, який поєднує 3 необхідні компоненти MVC в один пакет. Створюючи незалежні компоненти верхнього рівня, розробники можуть швидко та легко використовувати ці компоненти в інших програмах.

2.2 Аналіз вимог до інформаційної системи

IEEE (Institute of Electrical and Electronics Engineers – Інститут інженерів електротехніки і електроніки) Standard Glossary of Software Engineering Terminology визначає вимоги як:

- умови або можливості, необхідні користувачеві для вирішення проблем або досягнення цілей;
- умови або можливості, якими повинна володіти система або системні компоненти, щоб виконати контракт або задовольняти стандартам, специфікаціям або іншим формальним документам;

Бізнес-правило — це політика, положення, стандарт, правило або формула розрахунку, які визначають або обмежують певні аспекти бізнес-процесу.

Бізнес-вимога — це набір інформації, яка разом описує необхідність ініціювати один або більше проектів, спрямованих на надання рішення та досягнення бажаного кінцевого бізнес-результату. Бізнес-вимоги включають бізнес-можливості, бізнес-цілі, показники успіху, концепції та межі та обмеження.

Обмеження - можливості проектування або будівництва, накладені на розробників. Інші типи обмежень можуть обмежувати можливості, доступні керівникам проектів. Бізнес-правила часто накладають обмеження на бізнес-операції і, отже, на програмні системи.

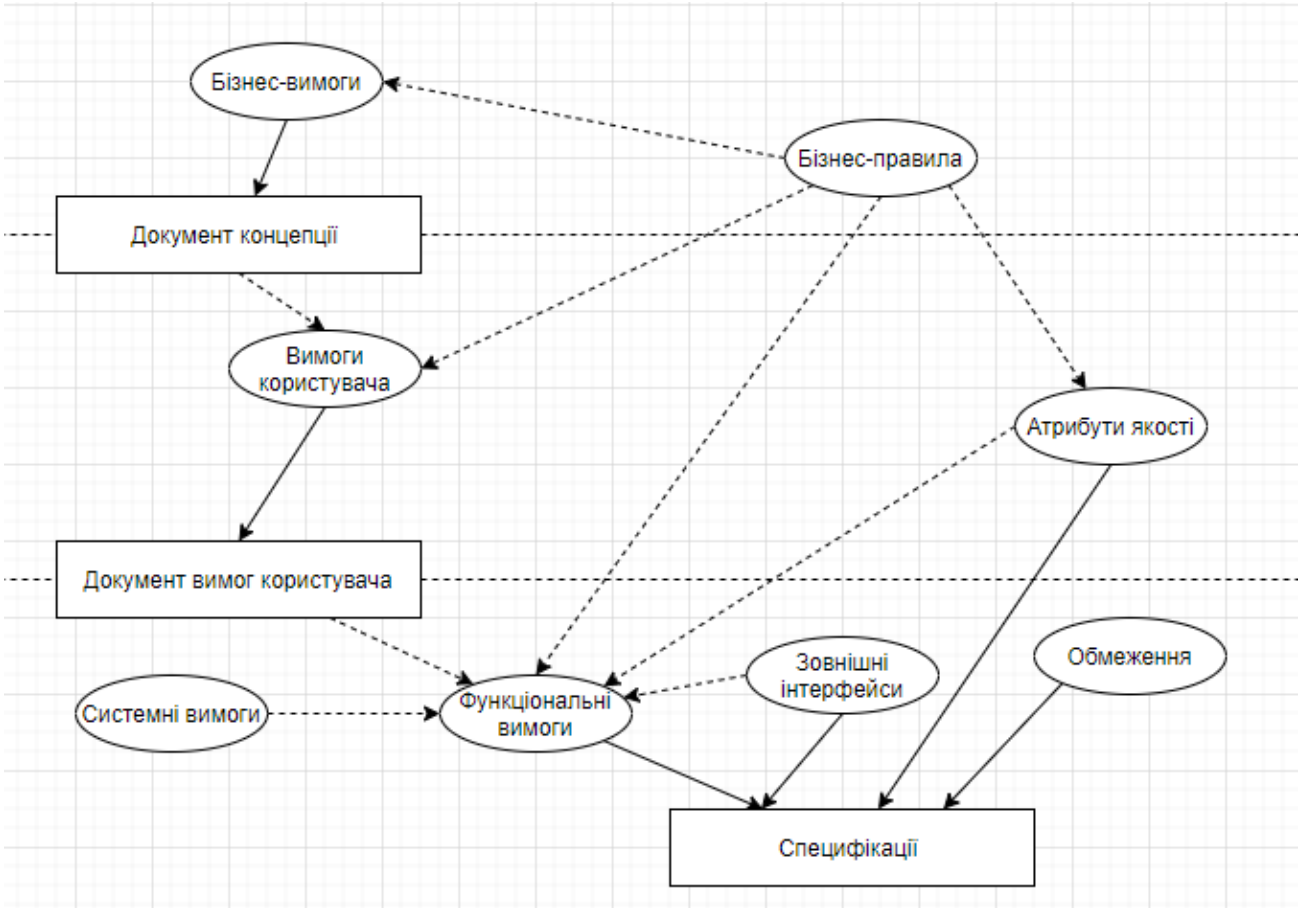

Рисунок 2.1 – Взаємозв'язки декількох типів інформації для вимог

Функціональна вимога – це опис поведінки системи за певних умов.

Атрибути якості — це клас нефункціональних вимог, які описують характеристики продуктивності послуги або продукту. Приклади атрибутів якості: зручність і простота використання, легкість переміщення, простота експлуатації, цілісність, надійність, ефективність і стійкість до поломок. Вимоги описують структуру атрибутів якості, за якими продукт демонструє бажані характеристики.

Вимоги користувача — цілі та завдання, які користувачі повинні мати можливість виконати при використанні системи, або очікування користувача щодо якості системи. Вимоги, визначені користувачем, зазвичай представлені у формі варіантів використання, визначених користувачем історій і сценаріїв.

Системні вимоги — це вимоги верхнього рівня для продукту, що складається з багатьох підсистем, які можуть бути програмним забезпеченням або комбінацією програмного забезпечення та обладнання [14].

Вимоги до інтерфейсу зовнішнього пристрою – опис інтерфейсу між системою програмного забезпечення та користувачем, іншою системою програмного забезпечення або пристроєм.

Нефункціональна вимога — це опис притаманних властивостей або характеристик, які повинна демонструвати система програмного забезпечення, або обмежень, яких необхідно дотримуватися [14].

Вимоги до системи за різними типами інформації наведено у таблиці 2.1.

Таблиця 2.1 – Вимоги до системи

Тип вимоги	Вимоги до системи
Бізнес – правило	Неможливо провести розрахунок без обраних матеріалів
Функціональні вимоги	Контроль користувацього вводу
Системні вимоги	Система повинна підтримувати інструментальні засоби, використані при розробці
Вимоги користувачів	Можливість обрати матеріали Можливість представити матеріали в двох варіанта виводу
Нефункціональні вимоги	Повинна підтримуватись операційними системами сімейства Windows.
Обмеження	Графічний інтерфейс повинен бути інтуїтивно зрозумілим
Атрибут якості	Система повинна контролювати введення даних користувачем

2.3 Вибір програмного середовища

C# — це універсальна багатопарадигмальна мова програмування, яка включає в себе сильну типізацію, лексичний обсяг, імперативне, декларативне, функціональне, загальне об’єктно-орієнтоване (на основі класів) і компонентно-орієнтоване програмування. Він був розроблений корпорацією Майкрософт приблизно в 2000 році в рамках ініціативи .NET і пізніше затверджений як міжнародний стандарт Ecma (ECMA-334) і ISO (ISO/IEC 23270:2018). Mono — це назва безкоштовного проекту з відкритим кодом для розробки компілятора та середовища виконання для мови. C# є однією з мов програмування, розроблених для спільної мовної інфраструктури (CLI).

C# розроблено Андерсом Хейлсбергом, команду розробників якого зараз очолює Мадс Торгерсен. Остання версія — 8.0, випущена в 2019 році з Visual Studio 2019 версії 16.3.

Під час розробки .NET Framework бібліотеки класів спочатку були написані за допомогою системного компілятора керованого коду під назвою Simple Managed C (SMC). У січні 1999 року Андерс Хейлсберг зібрав команду для створення нової мови під назвою Cool, що означає «C-подібна об'єктно-орієнтована мова». Корпорація Майкрософт розглядала можливість залишити назву «Cool» як остаточну назву мови, але вирішила відмовитися від неї з міркувань торгової марки. До того часу, коли проект .NET було публічно оголошено на конференції професійних розробників у липні 2000 року, мова була перейменована на C#, а бібліотеку класів ASP.NET і середовище виконання було перенесено на C#.

Гейлсберг є головним дизайнером і головним архітектором Microsoft C#, а раніше розробляв Turbo Pascal, Embarcadero Delphi (раніше відомий як CodeGear Delphi, Inprise Delphi і Borland Delphi) і Visual J++. У своїх інтерв’ю та технічних статтях він показує, що недоліки в більшості основних мов програмування, таких як C++, Java, Delphi та Smalltalk, призвели до заснування Common Language Runtime (CLR), що, у свою чергу, призвело до розробки Сама мова C#.

З моменту виходу C# 2.0 у листопаді 2005 року C# і Java дедалі більше розходяться, ставши двома абсолютно різними мовами. Одним з перших великих відхилень було додавання узагальнення для двох мов із абсолютно різними

реалізаціями. C# використовує реифікацію, щоб надати «першокласні» загальні об’єкти, які можна використовувати як будь-який інший клас, і для виконання генерації коду під час завантаження класу. Крім того, C# додав кілька основних функцій до функціонального програмування, кульмінацією чого став випуск C# 3.0 у розширеннях LINQ із його структурою підтримки лямбда-виразів, методів розширення та анонімних типів. Ці функції дозволяють програмістам C# використовувати методи функціонального програмування, наприклад замикання, коли це вигідно для їх застосування. Розширення LINQ та імпорт функцій допомагають розробникам зменшити кількість шаблонного коду, який використовується для таких звичайних завдань, як запит до баз даних, синтаксичний аналіз XML-файлів або пошук структур даних, і перемістити фокус на фактичну логіку програми для кращої читабельності та зручності обслуговування.

За задумом C# — це мова програмування, яка безпосередньо відображається на базовій інфраструктурі спільної мови (CLI). Більшість його внутрішніх типів відповідають типам значень, реалізованим середовищем CLI. Однак специфікація мови не містить вимог до генерації коду компілятора, тобто не вказує, що компілятор C# повинен націлюватися на середовище виконання спільної мови або генерувати загальну проміжну мову (CIL) або генерувати будь-який інший конкретний формат. Теоретично компілятор C# може генерувати машинний код, як традиційні компілятори C++ і Fortran.

C# підтримує строго типізовані неявні оголошення змінних за допомогою ключового слова var і неявно типізовані масиви за допомогою нового ключового слова [] з наступним ініціалізатором колекції.

C# підтримує суворий логічний тип даних bool. Інструкції, які приймають умови, такі як while і if, вимагають виразу типу, який реалізує справжній оператор, наприклад логічного типу. Хоча C++ також має логічний тип, його можна вільно перетворювати в цілі числа та з них, а такі вирази, як if(a), повинні лише перетворювати a у bool, дозволяючи a бути int або вказівником. C# забороняє цей істинний або хибний цілочисельний метод на тій підставі, що примушування програмістів використовувати вирази, які повертають bool, повністю запобігає

певним типам помилок програмування, таким як if (a = b) (використання assignment = замість equals ==).

C# більш безпечний, ніж C++. Єдині неявні перетворення за замовчуванням – це ті, які вважаються безпечними, наприклад, цілочисельні розширення. Це застосовується під час компіляції, під час JIT і в деяких випадках під час виконання. Не існує неявних перетворень між логічними значеннями та цілими числами, а також між членами перерахування та цілими числами (за винятком літералу 0, який можна неявно перетворити на будь-який тип перерахування). Будь-яке кероване перетворення має бути явно позначене як явне або неявне, на відміну від конструкторів копіювання C++ і операторів перетворення, які неявні за замовчуванням.

C# явно підтримує коваріантність і контраваріантність у загальних типах, на відміну від C++, який підтримує контраваріантність лише певною мірою через семантику типу повернення віртуальних методів.

Переписувачі розміщуються у їх видимій зоні.

Мова C# не дозволяє використовувати глобальні змінні або функції. Усі методи та члени мають бути оголошені всередині класу. Статичні члени відкритих класів можуть замінювати глобальні змінні та функції.

На відміну від C і C++, локальні змінні не можуть приховати змінні всередині блоку, що містить.

C# підтримує строго типізовані покажчики на функції через ключове слово delegate. Як і фреймворк псевдо-C++ Qt, C# має семантику, включаючи середовище подій у стилі публікації-підписки, хоча C# використовує для цього делегати.

Керовану пам’ять не можна звільнити; натомість вона збирається автоматично. Збирання сміття вирішує проблему витоків пам’яті, звільняючи програміста від відповідальності за звільнення пам’яті, яка більше не потрібна.

На відміну від C++, C# не підтримує множинне успадкування, хоча клас може реалізувати будь-яку кількість інтерфейсів. Це було проектне рішення головного архітектора мови, щоб уникнути складності та спростити вимоги до архітектури загального CLI. При реалізації кількох інтерфейсів, що містять методи з однаковою сигнатурою, тобто. Тобто два методи з однаковою назвою приймають параметри

одного типу в однаковому порядку, C# дозволяє реалізувати кожен метод відповідно до інтерфейсу методу, що викликає, або, як Java, дозволяє реалізувати метод один раз і передати будь-який інтерфейс класу Кожного разу дзвонив один раз.

Однак, на відміну від Java, C# підтримує перевантаження операторів. Лише найбільш часто перевантажені оператори в C++ можуть бути перевантажені в C#.

C# має можливість використовувати LINQ через .NET Framework. Розробники можуть викликати будь-який об'єкт IEnumerable<T>, документ XML, набір даних ADO.NET і базу даних SQL. [60] Використання LINQ у C# пропонує такі переваги, як підтримка Intellisense, потужні можливості фільтрації, безпека типів із перевіркою помилок компіляції та узгодженість даних для запитів із різних джерел. Є кілька різних мовних конструкцій, які можна використовувати з LINQ у C#, це вирази запитів, лямбда-вирази, анонімні типи, неявно типізовані змінні, методи розширення та ініціалізатори об'єктів.

У серпні 2001 року Microsoft, Hewlett-Packard і Intel Corporation виступили співавторами специфікації C# та спільної мовної інфраструктури (CLI) у рамках організації стандартів Ecma International. У грудні 2001 року ECMA опублікувала специфікацію мови C# ECMA-334. C# став стандартом ISO у 2003 році (ISO/IEC 23270:2003 Інформаційні технології. Мови програмування-C#). Раніше ECMA прийняла еквівалентну специфікацію в грудні 2002 року як друге видання C#.

У червні 2005 року ECMA схвалила 3-е видання специфікації C# і оновила ECMA-334. Доповнення включають часткові класи, анонімні методи, типи з можливістю обнулення та загальні шаблони (подібні до шаблонів C++).

У липні 2005 року ECMA подала останній стандарт і відповідне ТЗ до ISO/IEC JTC 1 через прискорений процес. Зазвичай цей процес займає 6-9 місяців.

Визначення мов C# і CLI стандартизовано відповідно до стандартів ISO та Ecma, які забезпечують розумний і недискримінаційний захист від патентних претензій.

Корпорація Майкрософт погодилася не судитися з розробниками програмного забезпечення з відкритим кодом за порушення патентів у некомерційних проектах, які є структурними частинами, на які поширюється OSP. Корпорація Майкрософт також погоджується не застосовувати патенти на продукти Novell проти клієнтів

Novell, які платять, за винятком списків продуктів, у яких прямо не згадується C#.NET або реалізація .NET від Novell (проект Mono). Проте Novell стверджує, що Mono не порушує жодних патентів Microsoft. Microsoft також уклала певні угоди про звільнення від патентівПов'язано з плагіном браузера Moonlight, залежить від Mono, якщо отримано через Novell

Корпорація Майкрософт очолює розробку довідкового компілятора та набору інструментів C# з відкритим вихідним кодом, який раніше мав кодову назву «Roslyn». Компілятор, повністю написаний на керованому коді (C#), має відкритий вихідний код, що розкриває його функції як API. Це дозволяє розробникам створювати інструменти рефакторингу та діагностики.

Інші компілятори C# (деякі з яких включають загальну мовну інфраструктуру та реалізації бібліотеки класів .NET):

• Проект Mono надає компілятор C# з відкритим вихідним кодом, повну реалізацію загальномовної інфраструктури з відкритим кодом, включаючи необхідні бібліотеки інфраструктури, зазначені в специфікації ECMA, і майже повну реалізацію власних бібліотек класів Microsoft .NET до . Чистий 3.5. Починаючи з Mono 2.6, немає жодних планів впровадження WPF; WF заплановано на пізніший випуск; і існують лише часткові реалізації LINQ to SQL і WCF.

• Проект DotGNU (тепер неіснуючий) також надає компілятор C# з відкритим вихідним кодом, майже повну реалізацію загальномовної інфраструктури, включно з необхідними бібліотеками інфраструктури, зазначеними в специфікації ECMA, і деяку решту підмножини власних класів Microsoft .NET. Бібліотека .NET 2.0 (не задокументована та не включена до специфікації ECMA, але включена до стандартного дистрибутива Microsoft .NET Framework). • Microsoft Common Language Infrastructure під кодовою назвою «Rotor» забезпечує реалізацію середовища CLR і компілятора C# зі спільним кодом, ліцензований лише для освітніх і дослідницьких цілей, а також підмножину бібліотеки Common Language Infrastructure, яка вимагається специфікацією ECMA. (У C# 2.0, підтримується лише в Windows XP).

C# було обрано як основну мову для розробки цього проекту через усі вищезазначені причини, тобто функції, які підтримує мова, підтримувані операційні системи та легкість вивчення.

Загалом функціональні можливості C# найкраще підходять для такого роду розробки через просту реалізацію основних принципів ООП, легкість розробки форм і зрозумілий інтерфейс.

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft. Він використовується для розробки комп'ютерних програм, а також веб-сайтів, веб-додатків, веб-служб та мобільних додатків. Visual Studio використовує платформи Microsoft для розробки програмного забезпечення, такі як API Windows, Windows Forms, Foundation Presentation Foundation, Windows Store та Microsoft Silverlight. Він може створювати як власний код, так і керований код.

Visual Studio містить редактор коду, який підтримує IntelliSense (компонент завершення коду) і рефакторинг коду. Інтегрований налагоджувач можна використовувати як налагоджувач на рівні джерела, так і на рівні машини. Інші вбудовані інструменти включають аналізатор коду, конструктор GUI, конструктор веб-сторінок, конструктор класів і конструктор схем бази даних. Він приймає плагіни, які покращують функціональність майже на всіх рівнях, включаючи додавання підтримки систем керування джерелами, таких як Subversion і Git, і додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для мов визначення домену, або набір інструментів для інших аспектів життєвого циклу розробки програмного забезпечення. (як клієнт Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування та дозволяє редакторам коду та налагоджувачам підтримувати (різною мірою) майже будь-яку мову програмування, якщо існують служби для певної мови. Вбудовані мови включають C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML і CSS. Додаткові мови, такі як Python, Ruby, Node.js і M, підтримуються через плагіни. Java (і J#) підтримувалися в минулому.

Найпростіша версія Visual Studio, Community, доступна безкоштовно. Слоган для Visual Studio Community Edition: «Безкоштовна, повнофункціональна IDE для студентів, з відкритим кодом і індивідуальних розробників».

Наразі підтримується версія Visual Studio 2019.

Visual Studio внутрішньо не підтримує жодної мови програмування, рішення чи інструменту; натомість вона дозволяє підключати функції, закодовані як VSPackage. Після встановлення функція надається як послуга. IDE надає три служби: SVsSolution, яка надає можливість створювати списки проектів і рішень; SVsUIShell, яка надає функціональність вікон і інтерфейсу користувача (включаючи вкладки, панелі інструментів і вікна інструментів); і SVsShell, яка обробляє реєстрацію VSPackages. Крім того, IDE також відповідає за координацію та забезпечення зв'язку між службами. Усі редактори, дизайнери, типи проектів та інші інструменти реалізовані як VSPackages. Visual Studio використовує COM для доступу до VSPackage. Visual Studio SDK також включає Managed Package Framework (MPF), набір керованих оболонок навколо COM-інтерфейсів, які дозволяють писати пакети на будь-якій CLI-сумісній мові. Однак MPF не надає всіх функцій, доступних для інтерфейсу Visual Studio COM. Потім ці служби можна використовувати для створення інших пакетів для додавання функціональності до Visual Studio IDE.

Підтримка мов програмування додається за допомогою спеціальних VSPackages, які називаються мовними службами. Мовні служби визначають різні інтерфейси, які реалізації VSPackage можуть реалізувати для додавання підтримки різних функцій. Функції, які можна додати таким чином, включають забарвлення синтаксису, завершення операторів, зіставлення квадратних дужок, спливаючі підказки з інформацією про параметри, списки учасників і позначки помилок для фонової компіляції. Якщо інтерфейс реалізовано, функціональність буде доступна мовою. Мовні послуги реалізуються на мовній основі. Реалізації можуть повторно використовувати код із аналізаторів мови або компіляторів. Мовні служби можуть бути реалізовані у рідному коді та керованому коді. Для власного коду ви можете використовувати рідні COM-інтерфейси або структуру Babel (частина Visual Studio

SDK). Для керованого коду MPF включає оболонки для написання керованих мовних служб.

Visual Studio не містить вбудованої підтримки керування джерелами, але визначає дві альтернативи для інтеграції систем керування джерелами з IDE. Керування джерелом VSPackage може надавати власний настроюваний інтерфейс користувача. Навпаки, плагін керування вихідним кодом Microsoft Source Code Control Interface (MSSCCI) надає набір функцій для реалізації різних функцій керування вихідним кодом через стандартний інтерфейс користувача Visual Studio. MSSCCI спочатку використовувався для інтеграції Visual SourceSafe з Visual Studio 6.0, але згодом його було відкрито через Visual Studio SDK. Visual Studio .NET 2002 використовує MSSCCI 1.1, а Visual Studio .NET 2003 використовує MSSCCI 1.2. Visual Studio 2005, 2008 і 2010 використовують MSSCCI версії 1.3, яка додає підтримку для перейменування та видалення дистрибутивів і асинхронного відкриття.

Visual Studio підтримує кілька екземплярів середовища виконання (кожен екземпляр має власний набір VSPackages). Екземпляри використовують різні кущі реєстру (див. тут визначення кущів реєстру MSDN), щоб зберігати стан конфігурації, розрізняючи їхній AppId. Екземпляри запускаються спеціальним .exe для AppId, який вибирає AppId, інсталює кореневий вулик і запускає IDE. Пакет VSP, зареєстрований для AppId, інтегрується з іншими пакетами VSP для цього AppId. Різні версії продуктів Visual Studio складаються з різними програмами. Інсталяції продукту Visual Studio Express мають власний AppId, але продукти Standard, Professional і Team Suite мають той самий AppId. Тому випуски Express можна інсталювати разом з іншими випусками, на відміну від інших випусків, які оновлюють ту саму установку. Професійна версія включає набір VSPackages у стандартній версії, а набір команд включає надмножину VSPackages у двох інших версіях. Система AppId використовується оболонкою Visual Studio у Visual Studio 2008.

Дане середовище розробки було обрано по декількох критеріях:

- Підтримка мови C#
- Інтуїтивно зрозумілий інтерфейс

- Можливості графічного редактору, підходящі до задач

Так як усі критерії задовільнено — середовищем розробки було обрано саме Visual Studio 2019.

Висновки до розділу 2

В ході написання другого розділу роботи було проаналізовано архітектуру майбутньої системи, виявлено і задокументовано основні вимоги до розроблюваної інформаційної системи, проведено вибір і огляд основних інструментів розробки.

Вся отримана в ході написання розділу інформація ляже в основу розробки і дасть загальне розуміння технологічного забезпечення процесу розробки інформаційної системи.

РОЗДІЛ 3
СТВОРЕННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Розробка методів та компонентів роботи інформаційної системи

Діаграми UML показують, як різні програмні компоненти поєднуються разом, щоб утворити більші системи, поєднуючи їх в інші компоненти. Вони використовуються для показу структури будь-якої системи, яка не є надто великою чи складною.

Діаграми компонентів допомагають показати зацікавленим сторонам, що включатиме система, а що ні. Вони також використовуються як спосіб інформування розробників і передачі ідей між ними та їхніми зацікавленими сторонами. При створенні дорожньої карти проекту формалізовані діаграми допомагають програмістам і розробникам краще зрозуміти напрямок своєї роботи і прийняти рішення щодо майбутніх завдань. Вони також корисні для системних адміністраторів, яким потрібно спланувати логічні компоненти проекту програмного забезпечення та їхні зв’язки. Це пояснюється тим, що вони можуть використовувати діаграми компонентів для концептуалізації різних частин, які складають систему.

Компоненти визначаються елементом позначення компонента, який містить інформацію про інтерфейси компонента. Один із способів представити доступні інтерфейси компонента - це приєднати прямокутну коробку до елемента компонента. Крім того, компоненти можуть бути представлені кульками у формі розетки, з’єднаними з елементом. Проводячи суцільну лінію, що з’єднує компонент та інтерфейс, чітко видно загальну залежність компонента від інтерфейсу. Це можна побачити, спостерігаючи за значком у формі льодяника або кулі, з’єднаним із суцільною лінією. Подібним чином піктограма у формі розетки або півкола, з’єднана з лінією, вказує на залежність компонента від інтерфейсу, яка потребує саме цього інтерфейсу. Щоб показати зв’язок успадкованого інтерфейсу з іншим,

використовуйте лінію, з’єднану порожнистою стрілкою. Підключення льодяника до розетки суцільною лінією надає контекст залежностей.

На рисунку 3.1 зображена діаграма компонентів системи, яка показує взаємодію між компонентами системи та утворення з них цілісної системи.

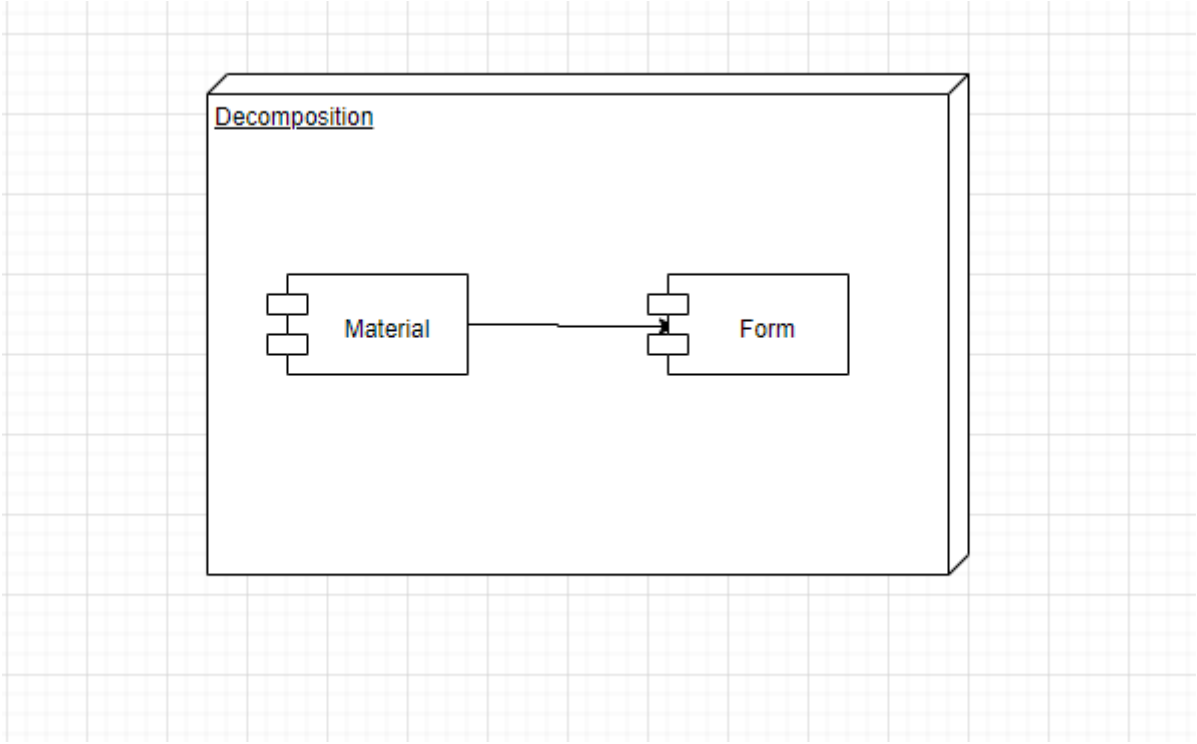

Рисунок 3.1 — Діаграма компонентів

3.2 Розробка алгоритмів

Основне функціональне призначення розроблюваної моделі полягає в системою вводу даних користувачів та виводом відповідного результату.

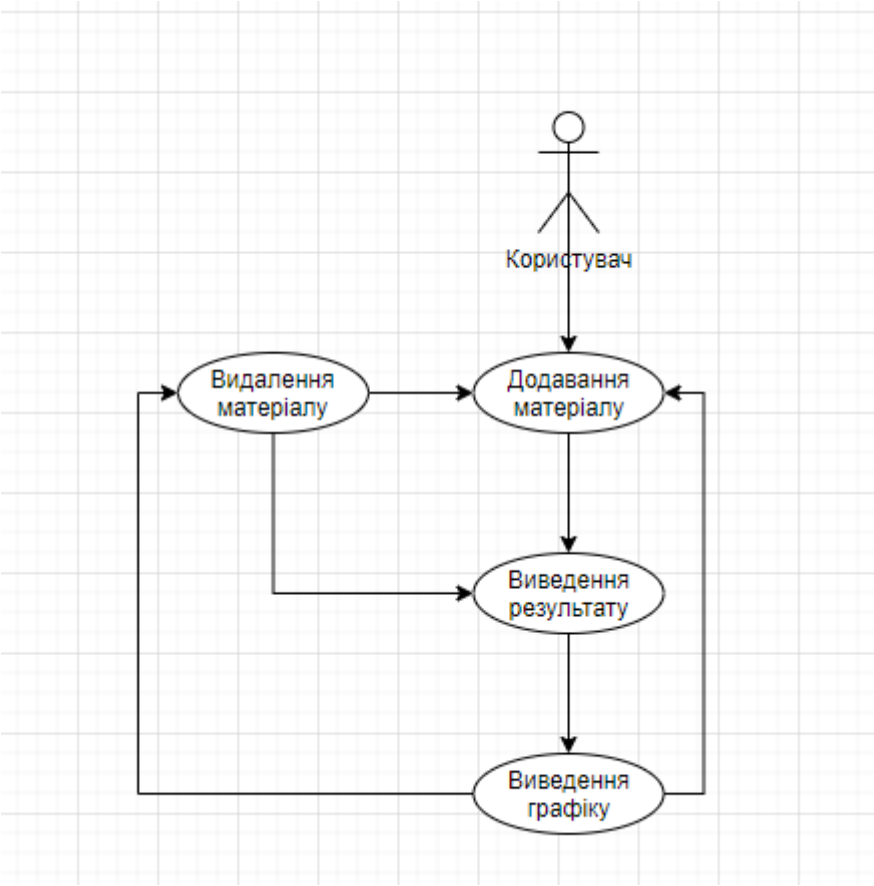

Рисунок 3.2 — Діаграма варіантів використання

3.3 Програмна реалізація інформаційної системи

У програмній інженерії діаграма класів в Уніфікованій мові моделювання (UML) - це тип статичної структурної діаграми, що описує структуру системи, показуючи класи системи, їх атрибути, операції (або методи) та взаємозв'язки між об'єктами.

Діаграма класів є основним будівельним елементом об’єктно-орієнтованого моделювання. Він використовується для загального концептуального моделювання структури програми та для детального моделювання переведення моделей у програмовий код. Діаграми класів також можуть бути використані для моделювання даних. Класи на діаграмі класів представляють як основні елементи, взаємодії в програмі, так і класи, що програмуються.

На схемі класи представлені вікнами, які містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

Щоб спроектувати систему, розгляньте можливість створення діаграми класів її класів та їхніх статичних зв’язків. Можна створювати підкласи, щоб розбивати концептуальні процеси проектування на менші частини.

Смислове відношення між залежним і незалежним елементами називається залежністю. Два елементи моделі є залежними, якщо зміни одного елемента можуть спричинити зміни іншого. Це вірно незалежно від того, змінюється клієнтський або серверний елемент. Відносини між замовником і постачальником показані пунктирною лінією з відкритою стрілкою, спрямованою від одного до іншого.

Кінцевий автомат або діаграма класів можуть бути додані для надання додаткової інформації про поведінку систем.

Асоціація представляє сім'ю відносин. Асоціація може пов'язувати будь-яку кількість класів; дві бінарні асоціації зазвичай представлені як пара рядків. Потрійна

асоціація має назву та прикрашені кінці для кожного члена. Він також може відображати кратність, показники власності та інші характеристики. Потрійні асоціації об’єднуються в групи по три.

Асоціації бувають чотирьох різних смаків. Найбільш поширеними є однонаправлені та двонаправлені асоціації. Інші типи асоціацій включають рефлексивні, які включають композиційну агрегацію, і агрегацію, що включає коасоціацію.

Льотні класи завжди йдуть поруч із класами літаків. Це пов’язано з двонаправленою природою асоціацій, які представляють спільні відносини між двома класами.

На зображенні професор Ернхардт веде урок. Цей зв'язок між ним і класом називається агрегацією. Це зв’язок, який представляє частково повний зв’язок або повний зв’язок, але він більш конкретний, ніж зв’язок. Асоціації можуть бути бінарними асоціаціями або типами асоціацій. І бінарні асоціації, і типи асоціацій можуть мати однакові назви, прикраси та більше двох класів. Однак тип асоціації не може мати більше двох класів. Будь-які асоціації або агрегації, представлені на діаграмі, представлені однаково під час впровадження. Отже, більшість діаграм не містять зв’язків агрегації. [7]

Контейнерні класи не мають сильної залежності життєвого циклу від класу-контейнера. Коли контейнер знищено, його вміст все ще існує.

Клас, який охоплює інші класи, представлений на графі UML у вигляді ромба з єдиною лінією, що з’єднує крайній клас із внутрішнім елементом ромба. Агрегати — це семантично розширені об’єкти, які можна розглядати як єдине ціле під час багатьох операцій. Однак фізично вони складаються з кількох менших об’єктів.

У цій метафорі бібліотека та студенти існують разом. Проте студент може існувати без підключення до бібліотеки через агрегацію.

У теорії категоризації спеціалізації загального типу (супертипу) можна вважати підкласом, а суперклас є узагальненням підкласу. Це означає, що будь-який екземпляр підкласу також представляє екземпляр суперкласу. Щоб зрозуміти зв’язок між класами, подивіться, як побудовані такі фрази, як «А — це Б».

Наприклад, людина — це ссавець і тварина. Така фраза поєднує більшість класів у системі таксономії. Подібним словосполученням можна зв’язати уроки зоології — людина — вищий примат, який є ссавцем.

Дерево рядків, видовбане у формі трикутника, закінчується класом, який називається суперкласом. Середина трикутника вказує на один або декілька підкласів. Це представлено в UML лінією, що з’єднує суперклас із підкласами.

У звичайній мові відносини між поколіннями відомі як спадковість або є відносинами.

Суперкласи — також відомі як батьківські класи, суперкласи або базові класи — пов’язані через концепцію узагальнення.

Підпорядковані групи або підмножини, визначені даним відношенням спеціалізації, називаються класами та «дітьми».

Різниця між цими двома спорідненими поняттями часто втрачається для громадськості. Їх плутано називають «біологічними батьками та дітьми».

В є типом А.

Прикладом використання типу слова є вислів про тип дерева, як-от дуб, або транспортний засіб, як-от автомобіль.

Діаграми класів і діаграми використання виявляють узагальнення.

У UML-моделюванні взаємозв’язок реалізації пов’язує два елементи моделі через зв’язок між ними. Один елемент діє як клієнт, реалізуючи поведінку, визначену іншим.

Графічні угоди про дерево ліній UML і порожнистий трикутник допомагають представити кінцевий результат реалізації. Пунктирна лінія з’єднує кінцевий результат з одним або декількома реалізаторами, які на діаграмах компонентів представлені з’єднанням кулька-розетка. Стрілка позначає кінцевий результат як зв’язаний із зовнішнім світом у пунктирній лінії. Діаграми класів можуть показувати лише реалізації; зв'язок між класами, компонентами та інтерфейсами, що з'єднує клієнтів із провайдерами. Реалізація показує зв’язок між класами, компонентами та інтерфейсами, кожен з яких з’єднує клієнтів із постачальниками. Це показує, що класи реалізують операції, запропоновані інтерфейсами.

Відносини між класами можуть бути слабкими або міцними. Залежність є слабшою формою асоціації. Це вказує на те, що один клас використовує інший у певний момент часу. Один клас використовує інший, якщо незалежний клас є змінною параметра або локальною змінною методу для залежного класу. Це відрізняється від асоціації; незалежний клас має атрибут залежного класу. Функції не містять змінних, пов’язаних із реалізацією. Натомість вони можуть взяти аргументи, пов’язані з реалізацією.

Додаткові метадані додаються до асоціацій через лінії, що з’єднують їх з іншими класами. Малюючи стрілку на кінці древка, ми показуємо, що видно хвіст. Диференціація властивості шляхом малювання відрізка або кола через вістря стрілки також позначає її як належність до певного класу. Крім того, ми можемо позначити асоціацію, назвавши всі ролі та вказавши, скільки екземплярів кожної ролі бере участь.

Класи сутності моделюють довгоживучу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

На рисунку 3.3 зображено діаграму варіантів використання, яка відображає структуру програмної реалізації розробленої інформаційної системи.

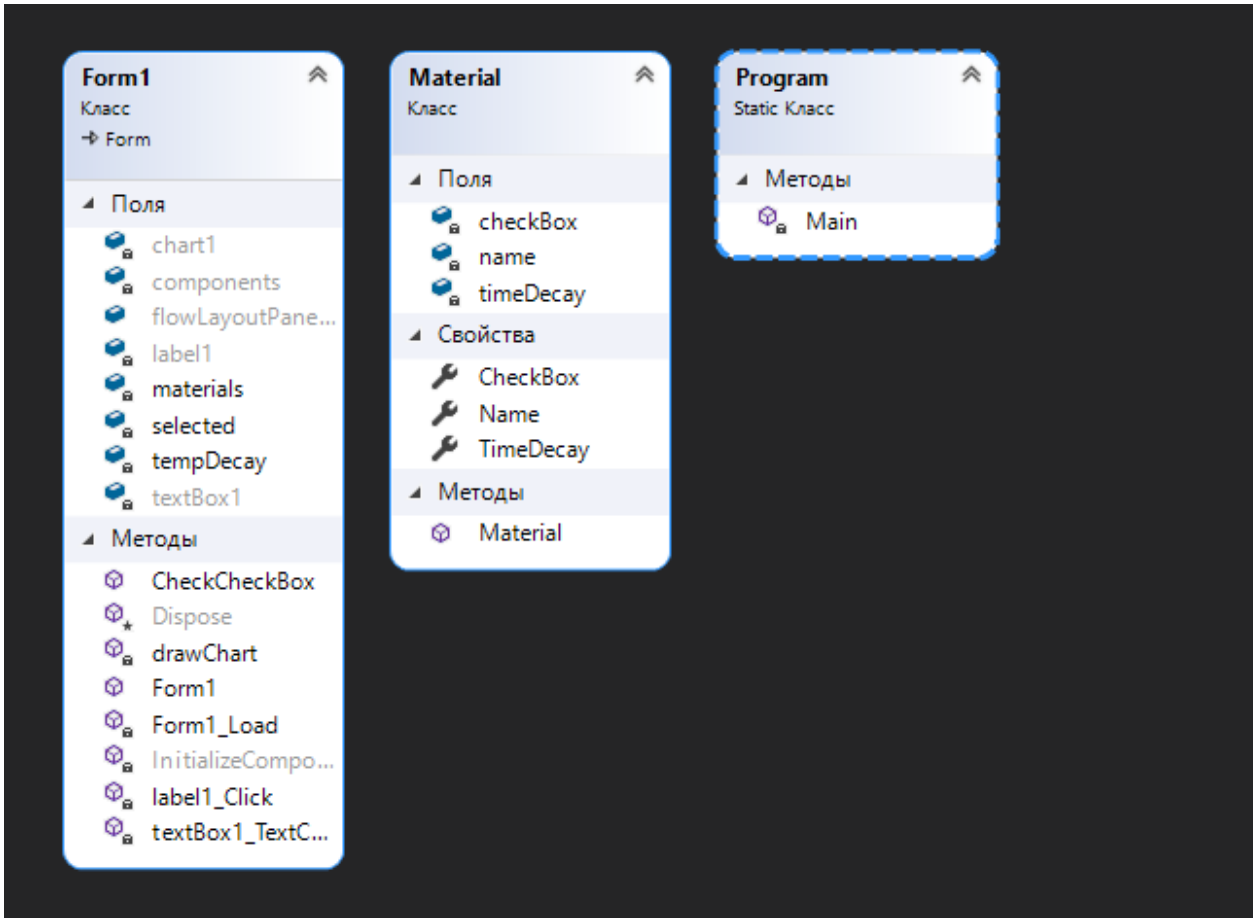

Рисунок 3.3 – Діаграма класів

3.4 Графічний інтерфейс інформаційної системи

Графічний інтерфейс користувача — це форма інтерфейсу користувача, яка дозволяє користувачеві взаємодіяти з електронним пристроєм за допомогою графічних піктограм і звукових індикаторів, таких як базові символи, а не текстового інтерфейсу користувача, міток введених команд або текстової навігації. Графічний інтерфейс користувача був представлений у відповідь на стрімке навчання інтерфейсу командного рядка (CLI), який вимагав введення команд на клавіатурі комп'ютера.

Дії в графічному інтерфейсі зазвичай виконуються шляхом безпосереднього маніпулювання графічними елементами. На додаток до комп'ютерів, GUI використовуються в багатьох портативних мобільних пристроях, таких як MP3-плеєри, портативні медіа-плеєри, ігрові пристрої, смартфони та невеликі домашні, офісні та промислові засоби керування. Термін «графічний інтерфейс», як правило, не застосовується до інших типів інтерфейсів із низькою роздільною здатністю, таких як відеоігри (бажано використовувати дисплеї в прямому ефірі (HUD)) або плоскі екрани, такі як дисплеї об'ємного звучання, оскільки цей термін обмежений ті, які здатні описувати загальну інформацію Сфера двовимірних дисплеїв, у традиціях досліджень інформатики в дослідницькому центрі Xerox Palo Alto.

Спочатку потрібно спроектувати логічні компоненти інтерфейсу сайту, а для цього потрібно зрозуміти, як сайт вирішує проблеми користувача.

Розробка візуальної композиції та тимчасової поведінки графічних інтерфейсів є важливою частиною програмування програмного забезпечення у сфері взаємодії людини з комп'ютером. Його метою є підвищення ефективності та простоти використання базової логічної конструкції збережених програм, критерій дизайну, який називається юзабіліті. Методи проектування, орієнтовані на користувача, використовуються, щоб гарантувати, що візуальна мова, введена в дизайн, добре підходить для завдання.

Особливість графічного інтерфейсу програми іноді називають chrome або GUI. Як правило, користувачі взаємодіють з інформацією, маніпулюючи візуальними віджетами, які дозволяють взаємодіяти відповідно до типу даних, які вони мають.

Виберіть добре розроблені віджети інтерфейсу користувача, щоб підтримувати дії, необхідні для досягнення цілей користувача. Model-View-Controller забезпечує гнучку структуру, де інтерфейс є незалежним і опосередковано пов'язаним із функціональністю програми, тому графічний інтерфейс можна легко налаштувати. Це дозволяє користувачам вибирати або створювати різні скіни за бажанням, а також полегшує дизайнерам зміну інтерфейсу в міру розвитку потреб користувачів. Хороший дизайн інтерфейсу більше стосується користувачів, ніж архітектури системи. Великі віджети, такі як вікна, зазвичай містять рамку або контейнер для основного вмісту презентації, наприклад веб-сторінки, повідомлення електронної пошти або зображення. Менші зазвичай є інструментами введення користувача.

Графічні інтерфейси можуть бути розроблені як графічні інтерфейси для конкретних програм для задоволення потреб вертикального ринку. Приклади включають банкомати (банкомати), сенсорні екрани торгових точок (POS) у ресторанах, автоматичні каси, які використовуються в роздрібних торгових точках, самообслуговування квитків і реєстрацію на рейс, кіоски в громадських місцях, таких як вокзали чи музеї, і використання операційних систем реального часу (RTOS) для моніторингу або керування екранами у вбудованих промислових програмах.

У 1980-х роках стільникові телефони та портативні ігрові системи також використовували сенсорний графічний інтерфейс. Новіші автомобілі використовують графічний інтерфейс користувача (GUI) у своїй навігаційній системі та мультимедійному центрі або комбінацію навігаційного мультимедійного центру.

Графічні інтерфейси використовують комбінацію технологій і пристроїв, щоб забезпечити платформу, з якою користувачі можуть взаємодіяти для збору інформації та виконання завдань виробництва.

Багато елементів, які відповідають візуальній мові, еволюціонували для представлення інформації, що зберігається в комп'ютерах. Це спрощує роботу та використання комп'ютерного програмного забезпечення для людей із невеликими навичками роботи з комп'ютером. Найпоширенішою комбінацією таких елементів

у графічному інтерфейсі є парадигма вікна, значка, меню, покажчика (WIMP), особливо в персональних комп’ютерах.

Стиль взаємодії WIMP використовує віртуальні пристрої введення для представлення положення інтерфейсу вказівного пристрою (найчастіше миші) і представляє інформацію, організовану у вікнах і представлену значками. Доступні команди зібрані в меню, а дії виконуються жестами за допомогою вказівного пристрою. Віконний менеджер полегшує взаємодію між вікнами, програмами та віконною системою. Віконна система обробляє апаратні пристрої, такі як вказівні пристрої, графічне обладнання та позиціонування покажчика.

У персональному комп’ютері всі ці елементи моделюються за допомогою метафори робочого столу для створення симуляції, що називається робочим середовищем, де дисплей представляє робочий стіл, на якому можна розміщувати документи та папки з документами. Менеджери вікон та інше програмне забезпечення поєднуються для імітації робочого середовища з різним ступенем реалістичності.

При розробці графічного інтерфейсу користувача було одне графічне вікно (рис. 3.4).

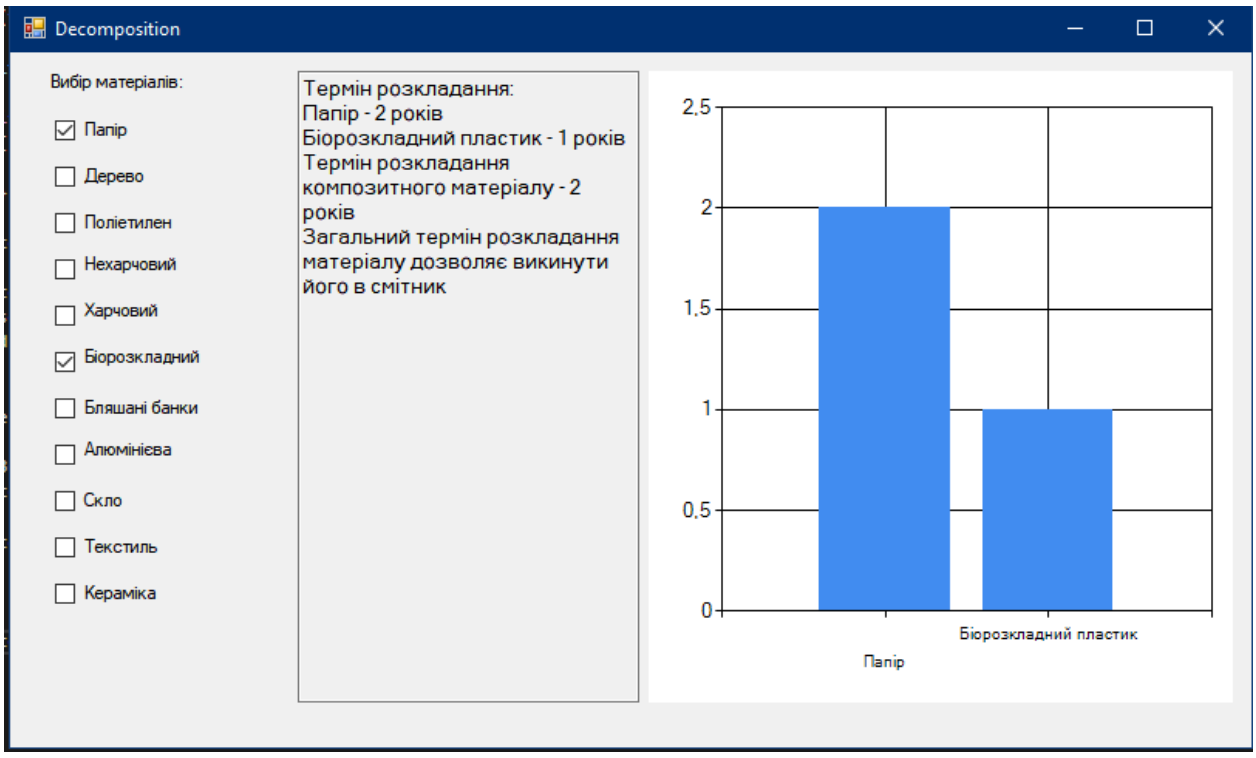

Рисунок 3.4 — Вікно програми

Графічний інтерфейс користувача складається з трьох основних елементів, а саме:

- списку матеріалів;
- поле виводу текстового результату;
- графік результатів.

3.4 Тестування системи

Тестування програмного забезпечення – це акт перевірки артефактів і поведінки програмного забезпечення, що тестується, шляхом перевірки та верифікації. Тестування програмного забезпечення також може забезпечити об’єктивне, незалежне уявлення про програмне забезпечення, щоб дозволити бізнесу оцінити та зрозуміти ризики впровадження програмного забезпечення. Методи тестування включають, але не обов’язково обмежуються:

- аналіз вимог до продукту щодо повноти та правильності в різних контекстах, таких як галузева перспектива, бізнес-перспектива, доцільність та життєздатність впровадження, зручність використання, продуктивність, безпека, міркування інфраструктури тощо.
- перегляд архітектури продукту та загального дизайну продукту
- робота з розробниками продуктів над удосконаленням техніки кодування, шаблонів проектування, тестів, які можна написати як частину коду на основі різних методів, таких як граничні умови тощо.
- виконання програми або програми з метою перевірки поведінки
- перевірка інфраструктури розгортання та пов’язаних скриптів та автоматизації
- брати участь у виробничій діяльності, використовуючи методи моніторингу та спостереження

Тестування програмного забезпечення може надати користувачам або спонсорам об’єктивну, незалежну інформацію про якість програмного забезпечення та ризик його несправності.

глюки і глюки

Помилки програмного забезпечення виникають через такий процес: Програмісти роблять помилки (помилки), які викликають помилки (дефекти, баги) у вихідному коді програмного забезпечення. Якщо виникне ця помилка, у деяких випадках система видаватиме неправильні результати, що призведе до збою.

Не всі невдачі обов’язково призводять до невдач. Наприклад, помилки в мертвому коді ніколи не спричинять збій. Невиявлені збої можуть спричинити збої під час зміни середовища. Приклади таких змін середовища включають програмне забезпечення, що працює на нових комп’ютерних апаратних платформах, зміни у

вихідних даних або взаємодію з іншим програмним забезпеченням. Один збій може спричинити широкий спектр симптомів збою.

Не всі помилки програмного забезпечення викликані помилками кодування. Загальним джерелом дорогих дефектів є прогалини у вимогах, невідомі вимоги, через які розробники додатків пропускають помилки. Прогалини у вимогах часто можуть бути нефункціональними вимогами, такими як тестування, масштабованість, ремонтопридатність, продуктивність та інші. Безпека

Статичні, динамічні та пасивні випробування

Існує багато підходів до тестування програмного забезпечення. Покрокові керівництва, покрокові керівництва або перевірки відомі як статичне тестування, тоді як виконання програмного коду з використанням заданого набору тестових випадків відоме як динамічне тестування.

Статичне тестування часто неявне, наприклад коректура, як статичний аналіз програм, коли інструменти програмування/текстові редактори перевіряють структуру вихідного коду, а компілятори (прекомпілятори) перевіряють синтаксис і потік даних. Динамічне тестування відбувається під час запуску самої програми. Динамічне тестування може початися до того, як програма буде завершена на 100%, щоб перевірити різні частини коду та застосувати до окремих функцій або модулів. Типовими підходами для цього є використання заглушок/драйверів або запуск із середовища налагодження.

Статичне тестування передбачає перевірку, а динамічне – також перевірку.

Пасивне тестування означає перевірку поведінки системи без будь-якої взаємодії з програмним продуктом. На відміну від активного тестування, тестер не надає жодних тестових даних, а переглядає системні журнали та трасування. Вони шукають моделі та конкретну поведінку, щоб приймати рішення. Це пов’язано з офлайн-тестуванням і аналізом журналу.

Дослідницьке тестування — це методологія тестування програмного забезпечення, яка коротко описується як одночасне навчання, розробка тесту та виконання тесту. Джем Канер, який ввів цей термін у 1984 році, визначив дослідницьке тестування як «стиль тестування програмного забезпечення, який

підкреслює особисту свободу та відповідальність окремого тестувальника, який постійно вдосконалює своє тестування, беручи до уваги навчання, пов’язане з тестуванням, дизайн тесту та Якість роботи, виконання тестів та інтерпретація результатів тестів як взаємодопоміжні операції, що виконуються паралельно протягом усього проекту».

Методи тестування програмного забезпечення традиційно поділяються на тестування білого ящика та тестування чорного ящика. Ці два підходи використовуються для опису точки зору тестувальника під час розробки тестового прикладу. Гібридний підхід, відомий як тестування сірого ящика, також можна застосувати до методів тестування програмного забезпечення. Це «довільне розрізнення» між тестуванням у чорній та білій скриньках дещо зникло, оскільки концепція «сірого ящика» побудови тестів на основі конкретних елементів дизайну набула популярності.

Виберіть підхід чорної скриньки для тестування розробленої системи. Тестування чорної скриньки — це метод тестування програмного забезпечення, який перевіряє функціональність програми, не дивлячись на її внутрішню структуру чи роботу. Цей підхід до тестування можна застосувати майже до кожного рівня тестування програмного забезпечення: модульного, інтеграційного, системного та приймального. Його іноді називають тестуванням на основі специфікацій.

Зазвичай не потрібні спеціальні знання програмного коду, внутрішніх структур і програмування. Тестери знають, що програмне забезпечення має робити, але не знають, як воно це робить. Наприклад, тестувальник знає, що певний вхід повертає деякий незмінний вихід, але не знає, як програмне забезпечення створило вихід.

Тестові приклади базуються на специфікаціях і вимогах, тобто на тому, що має робити програма. Тестові випадки зазвичай отримують із зовнішніх описів програмного забезпечення, включаючи специфікації, вимоги та параметри дизайну. Незважаючи на те, що використовувані тести мають переважно функціональний характер, можна також використовувати нефункціональні тести. Розробник тестів вибирає дійсні та недійсні вхідні дані та визначає правильний вихід, зазвичай

використовуючи тестовий оракул або відомі хороші попередні результати, не знаючи внутрішньої структури тестового об’єкта.

Тестування починається з моделі прогнозування на рік, для даної моделі тест-кейси представлені в таблиці 3.1.

Таблиця 3.1 — Тест кейси для моделі прогнозування на рік

№ п.п.	Опис	Очікуваний результат	Отриманий результат
1	Вибір одного компонента	Система показує один результат для компонента і сукупного матеріалу	Система показує один результат для компонента і сукупного матеріалу
2	Вибір всіх компонентів	Система показує результат для кожного окремого компонента і для сукупного матеріалу	Система показує результат для кожного окремого компонента і для сукупного матеріалу
3	Відсутність вибору	Система не видає жодних результатів і очікує вибору користувача	Система не видає жодних результатів і очікує вибору користувача

Тестування показує повну функціональну вірність, відсутність неточностей і багів в написаній системі.

Висновки до розділу 3

В ході написання третього розділу роботи було спроектовано і розроблено інформаційну систему. Було спроектовано компонентний склад системи, виявлено можливі діяльності користувачів в системі, створено внутрішню структуру і графічний інтерфейсу користувача.

ВИСНОВКИ

Будь-яке підприємство роздрібної торгівлі має діло з величезною кількістю різного роду пакувального матеріалу, який треба утилізувати тип чи іншим чином. Деякі композитні матеріали придатні для звичайного викидання, інші — тільки в спеціалізованих установах. Ці відмінності залежать від строку розкладання матеріалів, з яких складається пакувальний матеріал. Виходячи з цього виникла ідея створення системи оцінки строку розкладу композитних пакувальних матеріалів.

Перший розділ роботи присвячений аналізу предметної області, з оглядом понять переробки і розкладання відходів, поняття прикладного програмного забезпечення.

Другий розділ присвячено виявленню і аналізу вимог до програмного продукту, аналізу архітектури та обраних засобів реалізації.

Третій розділ повністю присвячений проектуванню і розробці описаної вище системи і його результатом є проведений аналіз варіантів використання системи і проект внутрішньої будови системи, проведено розробку графічного інтерфейсу та тестування системи.

Результатом роботи є повноцінна, функціонуюча система розрахунку строку розкладання композитних пакувальних матеріалів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Harper, Douglas. "retail". Online Etymology Dictionary. Retrieved 16 March 2008.
2. "retail" – via The Free Dictionary.
3. Pride, W.M., Ferrell, O.C. Lukas, B.A., Schembri, S. Niininen, O. and Cassidy, R., Marketing Principles, 3rd Asia-Pacific ed., Cengage, 2018, pp. 449–50
4. Pride, W.M., Ferrell, O.C. Lukas, B.A., Schembri, S. Niininen, O. and Cassidy, R., Marketing Principles, 3rd Asia-Pacific ed., Cengage, 2018, p. 451
5. Jones, Brian D.G.; Shaw, Eric H. (2006). "A History of Marketing Thought", Handbook of Marketing. Weitz, Barton A.; Wensley, Robin (eds), Sage, p. 41, ISBN 1-4129-2120-1.
6. Thompson, D.B., An Ancient Shopping Center: The Athenian Agora, ASCSA, 1993 pp. 19–21
7. McGeough, K.M., The Romans: New Perspectives, ABC-CLIO, 2004, pp. 105–06
8. Coleman, P., Shopping Environments, Elsevier, Oxford, 2006, p. 28
9. Moore, K., and Reid., S., "The Birth of the Brand: 4000 years of Branding", Business History, Vol. 50, 2008. pp. 419–32.
10. Eckhardt, G.M. and Bengtsson. A. "A Brief History of Branding in China", Journal of Macromarketing, Vol, 30, no. 3, 2010, pp. 210–21
11. Eckhardt, G.M. and Bengtsson. A. "A Brief History of Branding in China", Journal of Macromarketing, Vol, 30, no. 3, 2010, p. 212
12. Thrupp, S.L., The Merchant Class of Medieval London, 1300–1500, pp. 7–8
13. Pevsner, N. and Hubbard, E., The Buildings of England: Cheshire Penguin, 1978, p. 170
14. Gazetteer of Markets and Fairs in England and Wales to 1516, The List and Index Society, no. 32, 2003
15. Rebecca M. Seaman, ed. (2013). Conflict in the Early Americas: An Encyclopedia of the Spanish Empire's ... p. 375. ISBN 978-1-59884-777-2.
16. Cox, N.C. and Dannehl, K., Perceptions of Retailing in Early Modern England, Aldershot, Hampshire, Ashgate, 2007, p,. 129

17. Cox, N.C. and Dannehl, K., Perceptions of Retailing in Early Modern England, Aldershot, Hampshire, Ashgate, 2007, pp. 153–54
18. Conlin, J., Tales of Two Cities: Paris, London and the Birth of the Modern City, Atlantic Books, 2013, Chapter 2
19. Mill, J.S., Principles of a Political Economy with some of their Applications to Social Philosophy, 7th ed., London, Longman, 1909, Section IV.7.53
20. Reshaping Retail: Why Technology is Transforming the Industry and How to Win in the New Consumer Dr
21. Koot, G.M. (2011). "Shops and Shopping in Britain: from market stalls to chain stores" (PDF). University of Massachusetts, Dartmouth. Archived from the original (PDF) on 6 August 2019. Retrieved 29 May 2017.
22. Howard Moss, M., Shopping as an Entertainment Experience, Plymouth, Lexington Books, pp. 35–39
23. Goldstein. J., 101 Amazing Facts about Wales, Andrews, UK, 2013
24. Malcolm Gladwell, The Terrazzo Jungle, The New Yorker, March 15, 2004
25. Byrne-Paquet, L., The Urge to Splurge: A Social History of Shopping, ECW Press, Toronto, Canada, p. 83
26. Johanson, Simon (2 June 2015). "Bunnings Shifts Focus as it Upsizes Store Network". The Age.
27. Wahba, Phil (15 June 2017). "The Death of Retail is Greatly Exaggerated". Fortune (Print magazine). p. 34.
28. Wetherell, S., "The Shopping Mall’s Socialist Pre-History", Jacobin Magazine, 4 August 2014
29. Townsend, Matt; Surane, Jenny; Orr, Emma; Cannon, Christopher (8 November 2017). "America's 'Retail Apocalypse' Is Really Just Beginning". Bloomberg. Retrieved 15 January 2018.
30. Volpato, G. and Stocchetti, A., "Old and new approaches to marketing: The quest of their epistemological roots", MPRA Paper No. 30841, 2009, p. 34
31. Lambda, A.J., The Art Of Retailing, McGraw-Hill, (2003), 2008, pp. 315–26
32. Parker, Christopher J.; Wenyu, Lu (13 May 2019). "What influences Chinese fashion retail? Shopping motivations, demographics and spending". Journal of

Fashion Marketing and Management. 23 (2): 158–175. doi:10.1108/JFMM-09-2017-0093. ISSN 1361-2026. S2CID 170031856.

33. Fill, C., Marketing Communications: Framework, Theories and Application, London, Prentice Hall, 1995, p. 70
34. Yu-Jia, H. (2012). "The Moderating Effect of Brand Equity and the Mediating Effect of Marketing Mix Strategy On the Relationship Between Service Quality and Customer Loyalty ". International Journal of Organizational Innovation, 155–62.
35. Morschett, D., Swoboda, B. and Schramm, H., "Competitive Strategies in Retailing: An Investigation of the Applicability of Porter's Framework for Food Retailers Journal of Retailing and Consumer Services, Vol. 13, 2006, pp. 275–87
36. Constantinides, E., "The Marketing Mix Revisited: Towards the 21st Century Marketing", Journal of Marketing Management, Vo. 22, 2006, pp. 422-423
37. Berens, J.S., "The Marketing Mix, the Retailing Mix and the Use of Retail Strategy Continua", Proceedings of the 1983 Academy of Marketing Science (AMS), [Part of the series Developments in Marketing Science], pp. 323–27
38. Lamb, C.W., Hair, J.F. and McDaniel, C., MKTG 2010, Mason, OH, Cengage, pp. 193–94
39. Verhoef, P., Kannan, P.K. and Inman, J., "From Multi-channel Retailing to Omni-channel Retailing: Introduction to the Special Issue on Multi-channel Retailing", Journal of Retailing, vol. 91, pp. 174–81. doi:10.1016/j.jretai.2015.02.005
40. Dibb, S., Simkin, L., Pride, W.C. and Ferrell, O.C., Marketing: Concepts and Strategies, Cengage, 2013, Chapter 12
41. Nagle, T., Hogan, J. and Zale, J., The Strategy and Tactics of Pricing: A Guide to Growing More Profitably, Oxon, Routledge, 2016, p. 1 and 6
42. Brennan, R., Canning, L. and McDowell, R., Business-to-Business Marketing, 2nd ed., London, Sage, 2011, p. 331
43. Neumeier, M., The Brand Flip: Why customers now run companies and how to profit from it (Voices That Matter), 2008, p. 55
44. Irvin, G. (1978). Modern Cost-Benefit Methods. Macmillan. pp. 137–160. ISBN 978-0-333-23208-8.

45. Rao, V.R. and Kartono, B., "Pricing Strategies and Objectives: A Cross-cultural Survey", in Handbook of Pricing Research in Marketing, Rao, V.R. (ed), Northampton, MA, Edward Elgar, 2009, p. 15
46. Hoch, Steven J.; Drèze, Xavier; Purk, Mary E. (October 1994). "EDLP, Hi-Lo, and Margin Arithmetic" (PDF). The Journal of Marketing. 58 (4): 16–27. doi:10.1177/002224299405800402. S2CID 18134783.
47. Kaufmann, P., "Deception in retailer high-low pricing: A 'rule of reason' approach", Journal of Retailing, Volume 70, Issue 2, 1994, pp. 115–1383.
48. Guiltnan, J.P., "The Price Bundling of Services", Journal of Marketing, April 1987
49. Poundstone, W., Priceless: The Myth of Fair Value (and How to Take Advantage of It), New York: Hill and Wang, 2011, pp. 184–200
50. Barr, A., "PayPal Deepens Retail Drive in Discover Payments Deal", Technology News. 22 August 2012
51. Romis, Rafael. "Council Post: Three Ways To Crush E-Commerce: Busting Common Misconceptions". Forbes.
52. Steven Greenhouse (27 October 2012). "A Part-Time Life, as Hours Shrink and Shift". The New York Times. Retrieved 28 October 2012.
53. Hee, J.K., "Stand-alone Sale of a Free Gift: Is it effective to accentuate promotion value?" Social Behavior & Personality, Vol. 43, no. 10, 2015, pp. 1593–1606
54. Cant, M.C.; van Heerden, C.H. (2008). Personal Selling. Juta Academic. p. 176. ISBN 978-0-7021-6636-5.
55. Monash University, Dictionary, https://business.monash.edu/marketing/marketing-dictionary/r/retail-mix
56. "The Impact of Retail Servicescape on Buying Behaviour", BVIMSR's Journal of Management Research, Vol 6, No. 2, 2014, pp. 10–17
57. Wakefield, L.K. and Blodgett, G J., "The Effect of the Servicescape on Customers' Behavioral Intentions in Leisure Service Settings", The Journal of Services Marketing, Vol. 10, No. 6, pp. 45–61.

ДОДАТКИ

ДОДАТОК А «ФРАГМЕНТ КОДУ ГОЛОВНОЇ ФОРМИ»

```
// textBox1
            //
            this.textBox1.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.textBox1.Location = new System.Drawing.Point(186, 12);
            this.textBox1.Multiline = true;
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(23, 12);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(93, 13);
            this.label1.TabIndex = 3;
            this.label1.Text = "Вибір матеріалів:";
            this.label1.Click += new System.EventHandler(this.label1_Click);
            //
            // chart1
            //
            chartArea1.Name = "ChartArea1";
            this.chart1.ChartAreas.Add(chartArea1);
            this.chart1.DataSource = this.chart1.Series;
            legend1.Name = "Legend1";
            this.chart1.Legends.Add(legend1);
            this.chart1.Location = new System.Drawing.Point(413, 12);
            this.chart1.Name = "chart1";
            series1.ChartArea = "ChartArea1";
            series1.Legend = "Legend1";
            series1.Name = "Series1";
            this.chart1.Series.Add(series1);
            this.chart1.Size = new System.Drawing.Size(378, 409);
            this.chart1.TabIndex = 4;
            this.chart1.Text = "chart1";
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(803, 450);
            this.Controls.Add(this.chart1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.flowLayoutPanel1);
            this.Controls.Add(this.textBox1);
            this.Name = "Form1";
            this.Text = "Decomposition";
            this.Load += new System.EventHandler(this.Form1_Load);
            ((System.ComponentModel.ISupportInitialize)(this.chart1)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();
```