Міністерство освіти і науки України

Одеський національний політехнічний університет

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Решетняк Марія Юріївна,

студентка групи АС-151

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Інформаційна технологія створення прецедентів і концептуальних класів

Спеціальність:

121 – Інженерія програмного забезпечення

Спеціалізація:

Інженерія програмного забезпечення

Керівник:

Кунгурцев Олексій Борисович,

професор

Одеса – 2020

ЗМІСТ

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ4

ЗАВДВННЯ НА РОЗРОБКУ РОЗДІЛУ «ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ»6

АНОТАЦІЯ7

ВСТУП8

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ10

1.1 Аналіз предметної області10

1.2 Критичний огляд аналогів системи14

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ СТВОРЕННЯ ПРЕЦЕДЕНТІВ І КОНЦЕПТУАЛЬНИХ КЛАСІВ16

2.1 Вдосконалення математичної моделі створення прецедентів16

2.2 Математична модель створення концептуальних класів31

3 СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ34

3.1 Опис функціональних вимог34

3.2 Опис нефункціональних вимог44

4 ПРОЕКТУВАННЯ СИСТЕМИ47

4.1 Архітектура системи47

4.2 Проектування інтерфейсу користувача50

5 РЕАЛІЗАЦІЯ ТА ВИПРОБУВАННЯ СИСТЕМИ56

6 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ64

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ 68

ВИСНОВКИ70

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ71

Додаток А. ЛІСТИНГ ПРОГРАМИ72

Додаток Б. ПИТАННЯ З ОХОРОНИ ПРАЦІ ТА БЕЗПЕКИ У НАДЗВИЧАЙНИХ СИТУАЦІЯХ 140

Б.1 Організація та управління охороною праці IT-компанії «Apostera»140

Б.2 Визначення основних параметрів умов праці згідно індивідуального завдання143

Б.3 Індивідуальне завдання. Розрахунок системи повітрообміну на робочому місці інженера-програміста 145

Б.4 Безпека у надзвичайних ситуаціях техногенного характеру. Індивідуальне завдання. Розрахунок категорії пожежонебезпечних офісних приміщень148

Б.5 Безпека у суспільстві в умовах загрози COVID-19150

Б.6 Заходи з охорони праці та безпеки у надзвичайних ситуаціях151

Міністерство освіти і науки України

Одеський національний політехнічний університет

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Спеціалізація: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Крісілов В.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Решетняк Марії Юріївни, група АС-151

1. Тема роботи: Інформаційна технологія створення прецедентів і концептуальних класів

Керівник роботи: Кунгурцев Олексій Борисович, професор

затверджені наказом ректора від «29» жовтня 2020 р. № 412-в

1. Зміст роботи: вимоги до системи, дослідження і розробка інформаційної технології, розробка і випробування системи, проведення експерименту.
2. Перелік ілюстративного матеріалу: згідно слайдам презентації.
3. Консультанти розділів роботи:

|  |  |  |
| --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата |
| завдання видав | завдання прийняв |
| 7 | Москалюк А.Ю., доц. |  | 10.11.2020 |

1. Дата видачі завдання: «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 р.

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| №з/п | Назва етапів кваліфікаційної роботи | Строк виконання | Примітка |
| 1 | Дослідження предметної області |  | виконано |
| 2 | Аналіз вимог до системи |  | виконано |
| 3 | Розробка інформаційної технології |  | виконано |
| 4 | Розробка системи |  | виконано |
| 5 | Випробування системи |  | виконано |
| 6 | Проведення експерименту |  | виконано |
| 7 | Розгляд питань охорони праці безпеки у надзвичайних ситуаціях |  | виконано |

Здобувач вищої освіти \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Ю. Решетняк

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Б. Кунгурцев

**ЗАВДАННЯ**

на розробку розділу «Охорона праці та безпека в надзвичайних ситуаціях»

Решетняк Марії Юріївни, група АС-151

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Тема роботи: Інформаційна технологія створення прецедентів і концептуальних класів

Зміст розділу:

1. Аналіз умов праці і вибір заходів і засобів захисту від небезпечних і шкідливих виробничих факторів.
2. Аналіз техногенних небезпек і вибір заходів і засобів забезпечення безпеки у надзвичайних ситуаціях.
3. Індивідуальне завдання. Розрахунок системи повітрообміну на робочому місці інженера-програміста. Розрахунок категорії пожежонебезпечних офісних приміщень.

|  |  |
| --- | --- |
| Керівник роботи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Б. Кунгурцев«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2020р. | Консультант з охорони праці та БНС\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.Ю. Москалюк«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2020р. |

**АНОТАЦІЯ**

У дипломній роботі поставлено і виконано завдання розробки програмного забезпечення для конструювання прецедентів та концептуальних класів, а саме для автоматизації цих процесів. Дана система надає можливість підвищення продуктивності створення програмного забезпечення за рахунок швидшого формалізування функціональних вимог, зменшення кількості помилок при написанні сценаріїв прецедентів та виявлення концептуальних класів. Програмна система розроблена на мові HTML, з використанням CSS та JavaScript.

Ключові слова: HTML, CSS, JavaScript, варіант використання, інформаційна технологія, сценарій прецеденту, концептуальний клас.

**ANNOTATION**

In the thesis, the task of software development for construction of precedents and conceptual classes, more precisely for automation of these processes. This system provides an opportunity to increase the productivity of software development, reduce project execution time by faster formalization of functional requirements, reduce the number of errors in writing precedent scenarios and identify conceptual classes. The software system is developed with HTML, using CSS and JavaScript.

Keywords: HTML, CSS, JavaScript, use case, information technology, precedent scenario, conceptual class.

**ВСТУП**

Актуальність роботи визначається тим, що різні варіанти використання в рамках одного проекту можуть описуватись різними людьми і в різний час. У цих умовах важко уникнути несвоєчасного опису класів, дублювання їх атрибутів і функцій, нераціонального розподілу обов'язків між ними.

Відсутність безпосереднього зв'язку процесів створення концептуальних класів і пунктів варіантів використання може привести до нечіткого відображення вимог у системі, що проектується, зробити неможливим використання, пов'язаної з ініціалізацією даних інформації, що використовується у варіантах використання.

Існуючі моделі класів не містять інформації про призначення класу, його методів і атрибутів, не містять необхідних даних про зв'язки з іншими класами, варіантами використання і їх пунктами [1, 2]. Це призводить до великих втрат часу на внесення змін до вимог, коригування структури класів, тестування модулів.

Таким чином, метою представленої роботи є підвищення продуктивності створення програмного забезпечення, зменшення часу виконання проекту за рахунок швидшого проектування моделей концептуальних класів у відповідності до пунктів сценарію варіантів використання.

Відповідно до поставленої мети визначено такі завдання:

1. створити математичну модель концептуального класу,
2. визначити спосіб впровадження математичної моделі для автоматизованого конструювання концептуальних класів,
3. створити механізм формування концептуальних класів, який визначається типом відповідного пункту сценарію варіанту використання,
4. програмно реалізувати розроблену математичну модель.

Об’єктом дослідження даної роботи є процес створення програмного забезпечення та фіксації функціональних вимог, процес створення концептуальних класів.

Предметом дослідженняданої роботи єпрограмнізасоби, що використовуються при створенні програмного забезпечення для фіксації вимог та проектування архітектури.

Для досягнення поставленої мети були використані наступні методи дослідження:

1. метод аналізу функціональної забезпеченості інформаційної системи,
2. теорія множин для опису варіантів використання та концептуальних класів.

Наукова новизна:

1. розроблено метод для автоматизованого опису концептуальних класів, пов’язаний з попередньо розробленою моделлю опису варіантів використання,
2. вдосконалено модель опису варіантів використання до рівня, що дозволяє створювати альтернативні сценарії [3, 4].

Практичне значення даної роботи полягає у тому, щона основі запропонованих моделей автоматизованого опису варіантів використання та концептуальних класів розроблено систему, що дозволяє фіксувати функціональні вимоги та на їх основі проектувати програмне забезпечення у відповідності до поставленої у роботі мети.

Публікація: Решетняк М.Ю., Черепініна Я.В., Кунгурцев О.Б. «Інформаційна технологія конструювання варіанту використання та концептуальних класів», Збірник наукових праць молодих науковців і студентів з нагоди 30-річчя кафедри кібербезпеки та комп’ютерних систем і мереж ХНУ «Інтелектуальний потенціал – 2019», с. 79, Хмельницький, ХНУ.

**1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ**

* 1. **Аналіз предметної області**

Для забезпечення конкурентоспроможності програмного забезпечення та попередження ризику невдалого завершення проекту через нестачу ресурсів, чи то часу, чи то коштів, розробка програмного забезпечення повинна починатися зі збору функціональних вимог та їх детального опрацювання. Вдало виконаний описаний етап надалі буде наріжним каменем успішного створення загальної архітектури усього проекту.

Мета розробки програмного забезпечення полягає в тому, аби не порушуючи часових обмежень та відведених ресурсів (грошових, людських і так далі), розробити саме таке програмне забезпечення, що задовольнятиме умовам ринкової конкуренції, відповідатиме потребам користувачів, та відображатиме всі задокументовані вимоги замовника, якщо він присутній у проекті.

Зазвичай, значна частина часу, відведеного на розробку програмного забезпечення, витрачається на роботу з вимогами – властивостями, які повинна мати система або її компоненти, щоб задовольняти вимоги контракту, стандарту, специфікації або іншої формальної документації [5, 6, 7].

Такий розподіл часу обумовлено тим, що усунення проблем неповноти, неузгодженості або двозначності на етапі розробки вимог вимагає на кілька порядків менше витрат ресурсів, ніж на подальших стадіях розробки.

Вимоги часто використовуються для комунікації між різними зацікавленими особами. Саме тому вони повинні бути прості і зрозумілі як для звичайних користувачів, так і для розробників. Відповідає за ці властивості, зазвичай, product-manager. Він має бути достатньо компетентним при формулюванні вимог, оскільки це значною мірою посилює ризик людського фактору у процесі розробки програмного забезпечення.

Для графічних моделей вимог історично використовувалися діаграми або методології графічного моделювання, серед яких найбільш популярною на сьогодні є UML (The Unified Modeling Language) [8, 9]. Діаграма варіантів використання дозволяє записати діаграму прецедентів, встановити асоціацію між прецедентами і акторами, але ніяк не визначає зміст прецедентів.

Подібна проблема існує у встановленні зв’язку між концептуальними класами та пунктами сценаріїв варіантів використання, оскільки виявлення сутностей та методів пов’язує ці процеси але не фіксується для подальшого використання.

Зазвичай, графічне формулювання варіантів використання деталізують до стану сценаріїв прецедентів, тобто спеціальних послідовностей дій або взаємодій між виконавцем і системою.

Багаторічне використання варіантів використання дозволило виробити ряд рекомендацій для їх створення, а також правила оформлення преамбули, опису основних успішних та альтернативних сценаріїв.

Також розроблено вимоги до формулювань деяких дій. Але, разом з тим, розрізнені і часто суперечливі правила складання прецедентів, їх знаходження у широкому спектрі різних джерел, у протилежність до централізованого зібрання, що було б вельми корисно, призводить до того, що при описі прецедентів часто допускаються помилки, які, як зазначено вище, коштують що дорожче, в залежності від того, як пізно виявляються.

Помилки, допущені під час створення сценаріїв варіантів використання, також значною мірою ускладнюються або унеможливлюють процес подальшої розробки архітектури програмного забезпечення.

Створення концептуальних класів значною мірою залежить від опису прецедентів, оскільки виявлення сутностей та методів майбутніх класів пов’язане з пунктами сценаріїв. Таким чином можна побачити намітки майбутньої програмної реалізації ще на етапі збору вимог, і було б корисно такі архітектурні нариси зберігати для подальшого архітектурного аналізу.

На рис. 1.1 зображено описаний вище процес роботи бізнес-аналітика. Видно, наскільки вагому роль у ньому відіграє людський фактор, втома працівника чи втрата пильності, від чого не може застрахувати ні компетентність, ні досвідченість бізнес-аналітика.



Рисунок 1.1 – Стандартний процес роботи бізнес-аналітика

Для того, аби полегшити роботу бізнес-аналітика, було б корисно розроблено методику для створення сценаріїв варіантів використання з подальшим переходом до створення концептуальних класів. Така методика, формалізована у вигляді математичної моделі, могла б стати основою інформаційної технології для автоматизації цих процесів.

Дана методика створювалася шляхом збору та аналізу більшості рекомендацій, правил та вимог до формулювання деяких дій, а також усунення протиріч та розбіжностей.

На основі аналізу дій та даних з пунктів сценаріїв було вибудувано математичну модель концептуальних класів, логічно пов’язаних з кожним пунктом та між собою за необхідності.

На рис. 1.2 відображено можливий автоматизований процес роботи бізнес-аналітика з використанням інформаційної системи, що розроблюється у даній роботі.

На схемі чітко проглядається послідовна логічна залежність між сценарієм варіанту використання та створеними на його основі концептуальними класами. Також видно, що у цьому випадку людський фактор вже не має надто високого впливу на правильність формулювання вимог та їх подальшу обробку.



Рисунок 1.2 – Процес роботи бізнес аналітика з використанням інформаційної системи

Подальшим логічним кроком буде визначення очікувань користувача від системи за допомогою канви ціннісної пропозиції. Результати відображені у таблиці 1.1 нижче.

Таблиця 1.1 – Канва ціннісної пропозиції

|  |  |
| --- | --- |
| **Ціннісна пропозиція** | **Сегмент користувачів** |
| автоматизація процесу створення прецедентів та концептуальних класів | бізнес-аналітики проектів з розробки програмного забезпечення |
| **Послуга, що пропонується** | **Задача, яку вирішує користувач** |
| інформаційна технологія | фіксація функціональних вимог |

Продовження таблиці 1.1

|  |  |
| --- | --- |
| **Що утворює цінність** | **Що цінно для користувача** |
| 1. загальні рекомендації перетворено у строгу математичну модель,
2. фіксування логічної пов’язаності між пунктом сценарію та концептуальними класами,
3. автоматизоване генерування документації.
 | 1. скорочення часу на опис прецедентів та визначення концептуальних класів,
2. якість отриманих сценаріїв та збереження інформації про концептуальні класи для архітектурного аналізу.
 |
| **Що вирішує проблему** | **Проблеми користувача** |
| програмно реалізована та експериментально перевірена математична модель | 1. ризик людського фактору,
2. ризик дублювання робіт через неякісне документування.
 |

* 1. **Критичний огляд аналогів системи**

Наразі на ринку присутні наступні рішення вищеозначеної проблеми: Star UML, Confluence, Podio, NPJ. Розроблювана система далі буде означуватись як UC\_CC\_Creator.

Визначено наступні критерії оцінки існуючих рішень:

1. автоматизованість – кількість роботи, що бізнес-аналітик повинен виконувати мануально (0 – все, 1 – нічого);
2. швидкість – час, що знадобиться бізнес-аналітику на створення документації щодо функціональних вимог та концептуальних класів (0 – повільно, 1 – швидко);
3. логічні залежності – збереження додаткової інформації щодо зв’язку пункту прецеденту та концептуальних класів за замовчуванням (0 – відсутнє, 1 – присутнє);
4. зручність використання – звичність та зрозумілість графічного інтерфейсу користувача (0 – не зручно, 1 – зручно).

На рис. 1.3 наведено стратегічну канву з результатами аналізу.

Рисунок 1.3 – Стратегічна канва

Аналогічні рішення можуть лише фіксувати данні введені бізнес-аналітиком з автозбереженням, але не здатні автоматично створювати сценарії прецедентів із дотриманням рекомендацій, що забезпечують їх зрозумілість для розробників проекту та створювати концептуальні класи на їх основі.

**2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ СТВОРЕННЯ ПРЕЦЕДЕНТІВ І КОНЦЕПТУАЛЬНИХ КЛАСІВ**

**2.1 Вдосконалення математичної моделі створення прецедентів**

Основою автоматизації процесу опису варіантів використання є класифікація можливих пунктів сценаріїв. Тільки в цьому випадку можна формалізувати процес опису варіантів використання, підготувавши для кожного типу пункту деякий шаблон його створення.

Класифікація дозволить не тільки скоротити час на створення опису за рахунок подальшої ​​автоматизації процесу створення варіанту використання, але дасть єдине розуміння кожного пункту сценарію, допоможе уникнути помилок і неточностей в описі.

У практиці класифікації найбільшу популярність отримали два методи класифікації – ієрархічний і фасетний. Для класифікації пунктів сценаріїв був обраний ієрархічний метод, оскільки він має такі переваги як: простота, висока інформаційна насиченість і оглядовість.

Виходячи з того, що кожен пункт сценарію – це взаємодія користувача з системою, яка повинна бути представлена ​​певними діями, найбільш часто використовуваними кваліфікаційними ознаками першого рівня повинні бути наступними:

* + - * «Введення»,
			* «Запит»,
			* «Команда».

При цьому під ознакою «Введення» будемо розуміти остаточний результат виконання пункту, коли система отримує нову, істотну для неї інформацію.

Під ознакою «Запит» будемо розуміти пункт, метою якого є отримання з системи інформації, значимої для користувача.

Під ознакою «Команда» будемо розуміти дії, спрямовані на управління системою, і безпосередньо не пов'язані з введенням і виведенням даних.

У свою чергу ознака «Введення» має додаткові ознаки. До них потрібно віднести:

* «Введення даних»,
* «Вибір зі списку».

«Введення даних» передбачає ініціативу користувача в формуванні даних. «Вибір зі списку» передбачає попереднє надання списку системою, а користувач може вводити тільки те, що представлено в списку.

Ознака «Команда» може мати додаткові ознаки. До них віднесемо:

* «Створити»,
* «Введення послуги»,
* «Виклик прецеденту»,
* «Повторення дії»,
* «Вдале завершення варіанту використання»,
* «Невдале завершення варіанту використання».

Ознака «Створити» передбачає створення по команді користувача деякого об'єкту в системі. «Введення послуги» відрізняється від введення даних можливістю переходу до іншого сценарію без перевірки умови. «Виклик прецеденту» означає перехід до іншого (раніше описаного) прецеденту з наступним поверненням в поточний сценарій. «Повторення дій» – пункт, який вказує, які пункти сценарію і скільки разів можуть повторюватися. Ознаки «Вдале завершення варіанту використання» і «Невдале завершення варіанту використання» передбачають деяку заздалегідь встановлену послідовність дій для фіксації і документування результатів виконання варіанту використання.

Ознака «Запит» має додаткові ознаки:

* «Запит значення»,
* «Запит списку значень»,
* «Запит послуги».

«Запит списку значень» відрізняється від «Запиту значення» не стільки кількістю переданих даних, скільки подальшими діями. Дуже часто після цього пункту слідує пункт «Вибір зі списку». Ознака «Запит послуги» виділена як самостійну, оскільки подібно «Введенню послуги» може викликати перехід до іншого сценарію без перевірки умови.

Ієрархія класів представлена ​​на рис. 2.1 нижче.



Рисунок 2.1 – Дерево класифікації пунктів сценаріїв

Відповідно до наведеної класифікації нижче наведено короткий опис змісту відповідних пунктів сценарію у порядку частоти їх використання:

* Створити: користувач наказує системі створити певний документ (об'єкт), який в залежності від положення відповідного пункту в сценарії може грати роль деякого шаблону для накопичення інформації, або звіту про виконану роботу.
* Ввести данні: користувач вводить в систему ряд даних, для яких система має перевірити можливість їх використання для подальшої роботи.
* Запит значення: користувач запитує у системи деяке дане. Зазвичай після цього слідує оцінка даного користувачем.
* Запит списку значень: користувач замовляє список даних для подальшого вибору з нього деяких елементів.
* Вибрати зі списку: користувач вибирає зі списку потрібне дане
* Запит послуги: в результаті виконання цього пункту можливий перехід в іншу область сценарію.
* Запит з введенням значення: користувач вводить в систему ряд даних, коректність яких може перевірятись системою. Як відповідь система виводить розраховані дані.
* Ввести послугу (документ): користувач вводить необхідну послугу або документ, який визначає подальшу послідовність дій. Наприклад, спосіб оплати по банківській картці.
* Повторення дій: користувач має можливість перейти до розташованих вище пунктам сценарію, або відмовитися від їх повторення.
* Вдале завершення варіанту використання: пункт може передбачати введення деяких даних, їх збереження і генерування певних документів.
* Невдале завершення варіанту використання: в ході виконання сценарію розширення може виникнути ситуація, коли відсутня успішне вирішення і варіант використання повинен бути завершений. При цьому має сенс зберегти проміжні результати.
* Виклик прецеденту: управління буде передано іншому прецеденту, на якому може завершитися сценарій, або управління буде передано в пункт виклику.
* Вільно конструйований пункт: є ймовірність, що деяка взаємодія користувача з системою виявилась не врахованою. У цьому випадку йому буде запропоновано створити вільно конструйований пункт.

Дієвими і зацікавленими особами, які вказуються в варіанті використання, можуть бути люди і системи:

1. користувач системи, який буде з нею безпосередньо взаємодіяти для виконання своїх професійних обов'язків,
2. клієнт – замовник деякої послуги (може виявитися в ролі користувача),
3. інша система,
4. спонсор програмного забезпечення.

Визначимо зацікавлених осіб у вигляді множини

*mA = {mAp, mAs, mAo},* (2.1)

де *mAp* – множина основних виконувачів,

 *mAs* – множина допоміжних виконувачів,

 *mAo* – множина закулісних виконувачів.

Визначимо множину варіантів використання

*mU = {<uNamei*, *ma>}i = 1,k,* (2.2)

де *uName*  – найменування варіанту використання,

 *ma* – множина зацікавлених у варіанті використання осіб (*ma ∈* *mA)*.

Опис кожного варіанту використання описаний у вигляді кортежу

*u = <uName, pr, sc, mc>,* (2.3)

де *pr* – преамбула,

 *sc* – сценарій варіанту використання,

 *mc* – множина класів, що реалізують варіант використання.

Преамбула представлена кортежем

*pr = <ap, r, ma, da, pc, gm, gs>,* (2.4)

де *ap* – основна діюча особа (*ap ∈* *Ap ∧ ap ∈* *ma*),

 *r* – область дії варіанту використання,

 *da* – інтереси учасників,

 *pc* – передумова виконання варіанту використання, що ставить умови, які повинні бути виконані перед запуском цього варіанту використання,

 *gm* – мінімальні гарантії – це найменші обіцянки системи учасникам, зокрема, якщо мета основної діючої особи не може бути досягнута,

 *gs* – гарантії успіху – встановлюють, що інтереси учасників задовольняються при успішному завершені варіанту використання у кінці основного успішного сценарію чи у кінці успішного альтернативного шляху.

Часто у преамбулу включають тригер (подію, яке приводить до початку використання варіанту використання). У технології, що розглядається, тригер міститься у першому пункті основного успішного сценарію.

При складанні моделей важливою була можливість формалізації опису пунктів сценарію. Однак, оскільки в подальшому варіант використання буде вихідним матеріалом для проектування, то вже в даній моделі показано, яким чином буде формуватися модель проекту кожного пункту.

На рис. 2.2 представлено структуру моделі пункту сценарію, що має дві складові. Модель опису пункту є поданням вимог до програмного продукту і формує опис прецеденту. Модель проектування пункту не має представлення у вимогах і є початковим етапом проектування програмного продукту.

Запропоновано в якості елементів опису використовувати:

* заздалегідь заготовлені фрагменти тексту – *tti*;
* фрагменти тексту, що формуються в процесі складання і виконання варіанту використання – *tsj*; в залежності від предметної області, де планується використовувати варіант використання, аналітик формує цей фрагмент тексту; при виконанні варіанту використання в якості *tuj* записуються конкретні дані;
* текстові змінні, що представляють виконавців і зацікавлених осіб – *Client, Actor*.



Рисунок 2.2 – Структура моделі пункту сценарію

Для елементів проектування, створюваних паралельно з описом варіанту використання, наведена інформація про місця їх формування, приблизну структуру і призначення. При описі моделей пунктів сценарію використовується єдине позначення елементів проектування – *data*.

У моделі використовується ряд метасимволів. Один або кілька елементів, укладені в квадратні дужки – «[...]», не обов'язково повинні бути присутніми в описі пункту варіанту використання. Елементи, укладені в круглі дужки «(...)», можуть повторюватися кілька разів. Один або кілька елементів укладені фігурні дужки «{...}» відносяться до етапу проектування. Вони не відображаються в тексті варіанту використання і, при необхідності, можуть бути сформовані при повторному проході по тексту варіанту використання. Символ «+» позначає конкатенацію рядків. Символ «/» означає використання одного з двох елементів, розділених цим символом.

Для пункту «Створити» користувач наказує системі створити деякий об’єкт, який може містити дані, що використовуються у рамках даного варіанту використання, так й інших варіантів використання. При цьому попередньо заданих умов немає. В результаті опис пункту має вигляд:

|  |  |
| --- | --- |
| *create = <nP, [Client, tt1, Actor, tt2, ts1],* *Actor, tt3, ts2, {data1}, [tt5, (ts3, {data2})]>,* | (2.5) |

де *nP* – номер пункту варіанту використання,

 *Client* – необов’язковий елемент; вводиться, якщо потрібно визначити, від кого йде ініціатива,

 *tt1* – «звертається до»,

 *tt2* – «з приводу»,

 *tt3* – «створює у системі»,

 *ts1* – текст, що формує користувач (наприклад, «замовлення на ремонт устаткування»),

 *Actor* – користувач, що взаємодіє із системою (повинен бути визначеним у преамбулі прецеденту),

 *ts2* – назва об’єкту (наприклад, «нове замовлення»),

 *data1* – представлення об’єкта в моделі класу (ім’я, мета, тип),

 *tt5* – «з даними»,

 *ts3* – назва атрибута, що формується користувачем (наприклад, «номер телефону»),

 *data2* – представлення атрибута в моделі класу (ім’я, мета, тип).

Для пункту «Ввести дані» опис має вигляд:

*inputData = <nP, [Client, tt1, ts1], Actor, tt2, (ts2, {data1}, [tt3])>,* (2.6)

де *tt1* – «повідомляє»,

 *ts1* – текст, що формує користувач (наприклад, «місце проживання»),

 *tt2* – «вводить у систему»,

 *ts2* – текст, що формує користувач (наприклад, «почтовий індекс»),

 *data1* – список даних, що вводяться,

 *tt3* – «Система підтверджує коректність даних».

Для пункту «Ввести і отримати дані» опис має вигляд:

|  |  |
| --- | --- |
| *inOutData = <nP, [Client, tt1, ts1], Actor, tt2,* *(ts2, {data1}, [tt3]), tt4, ts3, {data2}, [ts4]>,* | (2.7) |

де *tt1* – «повідомляє»,

 *ts1* – текст, що формує користувач (наприклад, «певну суму»),

 *tt2* – «вводить у систему»,

 *ts2* – текст, що формує користувач, *tu1* чи його складові частини,

 *data1* – список даних, що вводяться,

 *tt3* – «Система підтверджує коректність даних»,

 *tt4* – «Система повертає»,

 *ts3 –* текст,що формує користувач,

 *data2* – значення, що повертає система,

 *ts4* – текст, що формує користувач (наприклад, «Клієнт згоден»).

Для пункту «Запит значення» опис має вигляд:

|  |  |
| --- | --- |
| *requestValue = <nP, [Client, tt1, ts1], Actor, tt2, ts2, {data1},* *[tt3, ts3, {data2}], [tt4, ts4, {data3}, tt5], tt6, {data3}, [tt7]>,* | (2.8) |

де *tt1* – «бажає отримати»,

 *ts1* – текст, що формує користувач (наприклад, «свідчення про»),

 *tt2* – «запитує у системи»,

 *ts2* – текст, що формує користувач (наприклад, «звіт про»),

 *data1* – дане, запитане у системи,

 *tt3* – «на основі»,

 *ts3 –* текст,що формує користувач (наприклад, «накладних про»),

 *data2* – дані, на основі яких визначається запитуване значення,

 *tt4* – «при умові»,

 *ts4* – текст, що формує користувач (наприклад, «постачальник»),

 *data3* – дані, на основі яких перевіряється виконання умови,

 *tt5* – «Система підтверджує виконання умови»,

 *tt6* – «Система виводить»,

 *tt7* – «*Client/Actor* + згоден».

Для пункту «Запит списку» опис має вигляд:

|  |  |
| --- | --- |
| *dataList = <nP, [Client, tt1, ts1], Actor, tt2,* *ts2, {data1}, [tt3, ts3, {data2}, tt4], tt5>,* | (2.9) |

де *tt1* – «бажає отримати список»,

 *ts1* – текст, що формує користувач (наприклад, «послуг, що надаються»),

 *tt2* – «вводить у систему запит на отримання»,

 *ts2* – текст, що формує користувач (наприклад, «список послуг»),

 *data1* – список, запитуваний у системи,

 *tt3* – «при умові»,

 *ts3 –* текст,що формує користувач (наприклад, «не дорожче»),

 *data2* – дані, на основі яких визначається умова отримання списку,

 *tt4* – «Система підтверджує виконання умови»,

 *tt5* – «Система виводить + *tu2*».

Для пункту «Обрати зі списку» опис має вигляд:

*selectElem = <nP, [Client, tt1], Actor, tt2, {data1}, tp3>,* (2.10)

де *tt1* – «вказує на потрібний елемент зі списку»,

 *tt2* – «обирає елемент зі списку»,

 *data1* – елемент списку,

 *tt3* – «і вводить його у систему».

Для пункту «Запит послуги (документу)» опис має вигляд:

|  |  |
| --- | --- |
| *requestService = <nP, [Client, tt1/tt2, ts1],* *Actor, tt3, ts2, {data1}, tt4,[ tt5, ts3]>,* | (2.11) |

де *tt1* – «бажає працювати з документом»,

 *tt2* – «бажає отримати»,

 *ts1* – текст, що визначає сервіс (наприклад, «огляд ходової частини») чи документ (наприклад, «заява на надання льотного тарифу»),

 *tt3* – «вводить у систему»,

 *ts2* – текст, що формує користувач, назва сервісу чи послуги,

 *data1* – представлення документу чи послуги у моделі класів,

 *tt4* – «Система підтверджує можливість виконання послуги (документу)»,

 *tt5* – «Передача управління на пункт сценарію»,

 *ts2* – номер пункту сценарію.

Для пункту «Повторення дій» опис має вигляд:

*repeatSteps = <nP, Client/ Actor, tt1, Np1>,* (2.12)

де *tt1* – «бажає повторити дії, починаючи з пункту»,

 *Np1* – номер пункту основного сценарію, що знаходиться вище пункту *nP.*

Для пункту «Вільно конструйований» опис має вигляд:

*freeConstr = <nP, [Client, ts1], Actor, ts2, action, finalPhrase>,* (2.13)

де *ts1, ts2* – тексти, що визначають зміст пункту,

 *action* – множина дій, де кожний елемент відповідає одному з можливих типів пунктів сценарію; дії можуть виконуватись у довільній послідовності,

 *finalPhrase* – один із варіантів фраз, що завершують пункт:

* «Система підтверджує коректність умови»,
* «Система перевіряє і підтверджує можливість виконання послуги»,
* «Система підтверджує коректність даних»,
* «Система підтверджує»,
* «*Client +* згодний»,
* «*Actor +* згодний»,
* «Завершення варіанту використання».

Для пункту «Вільно конструйований» опис має вигляд:

*finalStep = <nP, [elem1],[elem2], [elem3], tt1>,* (2.14)

де *tt1* – «Завершення роботи прецеденту»,

 *elem1, elem2, elem3* – необов’язкові елементи, що дозволяють прийняти дані, зареєструвати транзакцію та сформувати документ відповідно.

Операція вводу даного може повторюватись декілька разів. Для прийому даних опис має вигляд:

*elem1 = <[Client, tt2, ts1], (Actor, tt3, ts2, {data1}, tt4, ts3, {data2})>,* (2.15)

де *tt2* – «повідомляє»,

 *ts1* – список значень, що вводяться,

 *tt3* – «вводить»,

 *ts2* – назва даного, що вводиться,

 *data1* – представлення даного в моделі класів,

 *tt4* – «Система зберігає дане до»,

 *ts3* – назва об’єкта для зберігання даного (наприклад, «договір на поставку»),

 *data2* – представлення об’єкта для зберігання даного в моделі класів (об’єкт обирається зі створених раніше).

Операція реєстрації даного може повторюватись декілька разів. Для реєстрації даних опис має вигляд:

*elem2 = <tt5, (tt6, ts4, {data3}, tt7, ts5, {data4})>,* (2.16)

де *tt5* – «Система»,

 *tt6* – «реєструє: »,

 *ts4* – назва даного, що реєструється (наприклад, «номер місця у театрі»),

 *data3* – представлення даного в моделі класів,

 *tt7* – «до»,

 *ts5* – назва об’єкта для зберігання даного,

 *data4* – представлення об’єкта, що реєструє дане в моделі класів (об’єкт обирається зі створених раніше).

Документ може містити довільну кількість записів. Для створення документу опис має вигляд:

*elem3 = <tt8, ts6, {data5}, tt9,( ts7, {data6})>,* (2.17)

де *tt8* – «Система видає»,

 *ts6* – назва документу,

 *data5* – представлення документу в моделі класів,

 *tt9* – «де міститься: »,

 *ts7* – назва запису в документі,

 *data4* – представлення запису (даного), в моделі класів.

Для пункту «Виклик варіанту використання» опис має вигляд:

*requestUC = <Actor, tt1, ts1, {data1)>,* (2.18)

де *tt1* – «викликає варіант використання»,

 *ts1* – назва варіанту використання,

 *data1* – ідентифікатор варіанту використання в системі.

Для пунктів основного успішного сценарію, в яких передбачена перевірка введених користувачем даних або прийняття користувачем рішення в залежності від отриманих з системи даних, необхідно створити альтернативні сценарії.

Нумерація пунктів альтернативних сценаріїв утворюється автоматично і має наступний вигляд:

*nAP = <nP, dot, letter, [dot, number, dot]>* (2.19)

де *nP* – номер пункту основного успішного сценарію (наприклад, 4),

 *dot* – «.»,

 *letter* – множина латинських літер (наприклад, z)

 *number* – множина цілих чисел без знаку, додається для визначення підпункту.

Якщо в пункті «Ввести дані» основного сценарію відсутні елементи *tt3* («система перевіряє дане») і, як наслідок, елемент *tt4* («система підтверджує коректність даних»), то альтернативні сценарії не створюються.

Для кожного елемента *tt3* і відповідного *data1*  створюється альтернативний сценарій наступного формату:

* *nAP* «Значення даного» *ts2* «неприпустимо»;
* *nAP* «Введіть нове значення» *ts2* «Перехід до пункту *nP*».

Якщо в пункті «Запит значення» основного сценарію відсутні елементи *tt4, ts4, data3*, то альтернативні сценарії не створюються.

Якщо у пункті присутня умова отримання даного *tt4, ts4, data3*, то створюється альтернативний сценарій наступного формату:

* *nAP* «Умова» *ts4* «для отримання даного» *tu2* «неприпустима»;
* *nAP* «Введіть нову умову. Перехід до пункту *nP*».

Для кожного елемента *tt6* створюється альтернативний сценарій наступного формату:

* *nAP* «Клієнт не згодний»;
* *nAP* «Завершення варіанту використання».

Якщо в пункті «Запит списку» основного сценарію відсутні елементи *tt3, ts3, data2*, то альтернативні сценарії не створюються.

Якщо у пункті присутня умова отримання списку *tt3, ts3, data2*, то створюється альтернативний сценарій наступного формату:

* *nAP* «Умова» *ts3* «для отримання списку неприпустима»;
* *nAP* «Введіть нову умову. Перехід до пункту *nP*».

Для пункту «Обрати зі списку» створюється альтернативний сценарій наступного формату:

* *nAP* «Клієнт не згодний»;
* *nAP* «Завершення варіанту використання».

Для пункту «Запит послуги (документу)» створюється альтернативний сценарій наступного формату:

* *nAP* «Указана послуга (документ) не може бути надана»;
* *nAP* «Завершення варіанту використання».

Альтернативні сценарії для пунктів «Повторення дії» не передбачено, оскільки його виконання не є обов’язковим. Пункт «Завершити варіант використання» виконується коли всі інші пункти успішно завершились, тому альтернативний сценарій тут не передбачений.

**2.2 Математична модель створення концептуальних класів**

Усі класи, що входять у проект, представляються множиною

*mC = {c}*  (2.20)

Кожний клас (прототип) представлений кортежем

*c = <cHead, mFunc, mAttr>,* (2.21)

де*cHead* – заголовок класу,

 *mFunc –* множина функцій (методів) класу,

 *mAttr –* множина атрибутів класу.

Заголовок класу представлений кортежем

*cHead = <cName, tC, location, uName, nP, mPurp>,* (2.22)

де *cName* – назва класу,

 *tC* – вид класу, може приймати значення «*class*» або«*prototype*»,

 *location* – час життя об’єктів класу (або у ході виконання прецеденту – *u*, або у ході роботи системи – s),

 *uName, nP –* назва варіанту використання і номер пункту, де був створений клас (прототип),

 *mPurp –* масив цілей використання класу, представлений кортежем:

*mPurp = <uName, nP, purpose>* (2.23)

Множина функцій класу *mFunc* містить функції (методи) виду:

|  |  |
| --- | --- |
| *func = <fName, mPurp, mArgs, returnVal, mRsArgs,* *mNewValAttr, mCalcAttr, mRfFunc>*, | (2.24) |

де *fName –* ім’я функції,

 *mPurp –* масив цілей використання функції,

 *mArgs –* масив аргументів функції (кожний аргумент представлений ідентифікатором *id,* типом *argType*, та метою використання *argPurp*)*,*

 *returnVal –* значення, що повертається функцією (представлено типом значення *retType*, що повертається, та метою використання *purpose*),

 *mRsArgs –* множина аргументів метода, що використовуються, як результат обчислення,

 *mNewValAttr –* множина атрибутів, що приймають нові значення в результаті виконання функції,

 *mCalcAttr* – множина атрибутів класу, що використовуються у обчисленнях даної функції,

 *mRfFunc* – множина посилань на зовнішні функції (методи інших класів), що використовуються у даному методі. Кожний елемент множини *mRfFunc* представляється кортежем:

*mRfFunci* = *<cNamej, fName >*, (2.25)

де *cNamej* – клас, котрому належить зовнішня функція (у загальному випадку декілька зовнішніх функцій можуть належати одному класу),

 *fName* – ім’я зовнішньої функції.

Множина атрибутів класу містить атрибути виду:

*mAttr = <attrName, attrPurp, attrType, attrRf>,* (2.26)

де *attrName* – назва атрибута,

 *attrPurp* – мета використання атрибута,

 *attrType* – тип атрибута,

 *attrRf* – посилання на функції (методи), що використовують атрибут, представляється кортежем:

*attrRf* = *< fName, cName, uName, nP>* (2.27)

У моделі запропоновано використовувати наступні типи даних:

* *List* – список (наприклад, лінійний список, масив, множина);
* *Struct* – структура ( у загальному випадку містить поля різного типу), повинна містити нумерацію полів;
* *Text* – будь-який текст;
* *Numb* – будь-який числовий формат;
* *Bool* – логічне значення;
* *Void* – функція не повертає значення;
* *PClass* – посилання на об’єкт класу.

Для *argType* не використовується тип *Void.* Для *retType* не використовується тип *PClass.*

**3 СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ**

* 1. **Опис функціональних вимог**

Інформаційна система, повинна забезпечити перетворення даних, введених користувачем, у повноцінні сценарії варіантів використання та відповідні концептуальні класи.

Детальний аналіз предметної області дозволив визначити перелік варіантів використання та встановити зв’язки між ними:

* «Редагувати пункт варіанту використання»;
* «Заповнити пункт варіанту використання»;
* «Створити пункт варіанту використання»;
* «Створити альтернативний сценарій варіанту використання»;
* «Редагувати концептуальний клас»;
* «Заповнити концептуальний клас»;
* «Створити концептуальний клас»;
* «Створити документацію»;
* «Зберегти документацію»;
* «Переглянути документацію»;
* «Переглянути список раніше створених варіантів використання»;
* «Створити варіант використання»;
* «Заповнити преамбулу»;
* «Редагувати преамбулу».

На рис. 3.1 зображено діаграму варіантів використання системи, що розроблювалася в межах даної дипломної роботи. Для стислості викладу на діаграмі замінено наступні словосполучення:

* «варіант використання» на «UC»;
* «пункт варіанту використання» на «пункт»;
* «концептуальний клас» на «клас».



Рисунок 3.1 – Діаграма варіантів використання

Наступним кроком щодо визначення функціональних вимог до системи є створення сценаріїв варіантів використання, що поділяються на основні успішні та альтернативні.

Основний сценарій – це опис одного простого для розуміння й досить типового сценарію, у якому досягається мета основного діючої особи й задовольняються інтереси всіх учасників. Всі інші способи досягнення мети й невдачі описуються у альтернативних сценаріях.

Описи визначених варіантів використання наведені у табл. 3.1-3.14 нижче. При створенні сценаріїв даних варіантів використання, автор послуговувалась розробленою математичною моделлю.

Таблиця 3.1 – Опис варіанту використання «Редагувати пункт прецеденту»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Користувач виявив помилку у тексті пункту варіанту використання після успішного завершення варіанту використання «Створити пункт варіанту використання» |
| Основний сценарій | 1. Користувач вводить в систему змінене значення поля з помилкою відповідно до математичної моделі даного пункту в системі. Система підтверджує коректність даних. Система повертає текст пункту з оновленим значенням.
2. Користувач бажає повторити дії починаючи з пункту 1.
3. Система видає текст пункту де міститься значення полів, відповідно до математичної моделі в системі. Завершення роботи прецеденту.
 |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє внесені зміни. 3.а.2 Завершення роботи прецеденту. |

Таблиця 3.2 – Опис варіанту використання «Заповнити пункт прецеденту»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Розпочато виконання прецеденту «Створити пункт варіанту використання» |
| Основний сценарій | * + - 1. Користувач вводить в систему значення поля відповідно до математичної моделі даного пункту в системі. Система підтверджує коректність даних. Система повертає текст пункту.
 |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| Основний сценарій | * + - 1. Користувач бажає повторити дії починаючи з пункту 1.
			2. Система видає текст пункту де міститься значення полів, відповідно до математичної моделі в системі. Завершення роботи прецеденту.
 |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє створений текст пункту. 3.а.2 Завершення роботи прецеденту. |

Таблиця 3.3 – Опис варіанту використання «Створити пункт прецеденту»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Успішне завершення варіанту використання «Створити варіант використання» |
| Основний сценарій | Користувач вибирає елемент списку типів пунктів варіантів використання і вводить його в систему.Користувач викликає варіант використання «Заповнити пункт прецеденту». Користувач створює в системі пункт прецеденту.Завершення роботи прецеденту. |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє створення пункту. 3.а.2 Завершення роботи прецеденту. |

Таблиця 3.4 – Опис прецеденту «Створити альтернативний сценарій»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Успішне завершення варіанту використання «Створити пункт прецеденту» |
| Основний сценарій | 1. Інформаційна технологія створює в системі альтернативний варіант використання з даними, що вимагають перевірки.
2. Завершення роботи прецеденту.
 |
| Альтеративні сценарії | 1.а Дані для перевірки відсутні. 1.а.1 Інформаційна технологія відміняє створення альтернативних сценаріїв прецеденту. 2.а.2 Завершення роботи прецеденту. |

Таблиця 3.5 – Опис прецеденту «Редагувати концептуальний клас»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Користувач виявив помилку в концептуальному класі після успішного завершення варіанту використання «Створити концептуальний клас» |
| Основний сценарій | * + - 1. Користувач вводить в систему змінене значення поля з помилкою відповідно до математичної моделі концептуального класу в системі. Система підтверджує коректність даних. Система повертає концептуальний клас з оновленим значенням.
			2. Користувач бажає повторити дії починаючи з пункту 1.
 |

Продовження таблиці 3.5

|  |  |
| --- | --- |
| Основний сценарій | 1. Система видає концептуальний клас, де міститься значення полів, відповідно до математичної моделі в системі. Завершення роботи прецеденту.
 |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє внесені зміни. 3.а.2 Завершення роботи прецеденту. |

Таблиця 3.6 – Опис прецеденту «Заповнити концептуальний клас»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Розпочато прецедент «Створити концептуальний клас» |
| Основний сценарій | Користувач вводить в систему значення поля відповідно до математичної моделі концептуального класу в системі. Система підтверджує коректність даних. Система повертає концептуальний клас.Користувач бажає повторити дії починаючи з пункту 1.Система видає концептуальний клас де міститься значення полів, відповідно до математичної моделі в системі. Завершення роботи варіанту використання. |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє створений концептуальний клас. 3.а.2 Завершення роботи варіанту використання. |

Таблиця 3.7 – Опис прецеденту «Створити концептуальний клас»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Успішне завершення варіанту використання «Створити пункт варіанту використання» |
| Основний сценарій | Користувач викликає варіант використання «Заповнити концептуальний клас». Користувач створює в системі концептуальний клас.Завершення роботи прецеденту. |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє створення концептуального класу. 3.а.2 Завершення роботи прецеденту. |

Таблиця 3.8 – Опис варіанту використання «Створити документацію»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Успішне завершення варіанту використання «Створити концептуальний клас» |
| Основний сценарій | Інформаційна технологія створює в системі документацію з даними про основний успішний сценарій варіанту використання, альтернативний сценарій варіанту використання, концептуальні класи, створені на основі даного варіанту використання.Виклик варіанту використання «Переглянути документацію».Завершення роботи прецеденту. |

Продовження таблиці 3.8

|  |  |
| --- | --- |
| Альтеративні сценарії | 1.а Користувач не згодний. 1.а.1 Система відміняє створення документації. 1.а.2 Завершення роботи прецеденту. |

Таблиця 3.9 – Опис варіанту використання «Зберегти документацію»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Успішне завершення варіанту використання «Створити документацію» |
| Основний сценарій | * + - 1. Користувач відправляє у систему запит про збереження документації. Система підтверджує коректність даних. Система зберігає документацію.
			2. Завершення роботи прецеденту.
 |
| Альтеративні сценарії | 1.а Користувач не згодний. 1.а.1 Система відміняє збереження документації. 1.а.2 Завершення роботи прецеденту. |

Таблиця 3.10 – Опис варіанту використання «Переглянути документацію»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Успішне завершення варіанту використання «Створити документацію» або «Переглянути список раніше створених варіантів використання» |
| Основний сценарій | Користувач відправляє у систему запит про перегляд документації. Система підтверджує коректність даних. Система відображає документацію.Завершення роботи прецеденту. |

Продовження таблиці 3.10

|  |  |
| --- | --- |
| Альтеративні сценарії | 1.а Користувач не згодний. 1.а.1 Система відміняє перегляд документації. 1.а.2 Завершення роботи прецеденту. |

Таблиця 3.11 – Опис варіанту використання «Переглянути список раніше створених варіантів використання»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Успішне завершення варіанту використання «Створити документацію» або «Створити варіант використання» |
| Основний сценарій | * + - 1. Користувач вводить в систему запит на отримання списку раніше створених варіантів використання. Система виводить список раніше створених варіантів використання.
			2. Завершення роботи прецеденту.
 |
| Альтеративні сценарії | 1.а Раніше створені варіанти використання відсутні. 1.а.1 Система повідомляє Користувача про відсутність раніше створених варіантів використання. 1.а.2 Завершення роботи прецеденту. |

Таблиця 3.12 – Опис варіанту використання «Редагувати преамбулу»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Користувач виявив помилку у тексті преамбули після успішного завершення варіанту використання «Створити варіант використання» |

Продовження таблиці 3.12

|  |  |
| --- | --- |
| Основний сценарій | 1. Користувач вводить в систему змінене значення поля з помилкою відповідно до математичної моделі. Система підтверджує коректність даних. Система повертає оновлений текст преамбули.
2. Користувач бажає повторити дії починаючи з пункту 1.
3. Система видає текст преамбули, де міститься значення полів, відповідно до математичної моделі. Завершення роботи прецеденту.
 |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє внесені зміни. 3.а.2 Завершення роботи прецеденту. |

Таблиця 3.13 – Опис варіанту використання «Заповнити преамбулу»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Розпочато виконання варіанту використання «Створити варіант використання» |
| Основний сценарій | 1. Користувач вводить в систему значення поля згідно математичної моделі. Система підтверджує коректність даних. Система повертає текст преамбули.
2. Користувач бажає повторити дії починаючи з пункту 1.
3. Система видає текст преамбули, відповідно до математичної моделі. Завершення роботи прецеденту.
 |

Продовження таблиці 3.13

|  |  |
| --- | --- |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє створений текст преамбули. 3.а.2 Завершення роботи прецеденту. |

Таблиця 3.14 – Опис прецеденту «Створити варіант використання»

|  |  |
| --- | --- |
| Дієві особи | Користувач |
| Передумова | Користувач почав взаємодію з системою. |
| Основний сценарій | 1. Користувач викликає варіант використання «Заповнити преамбулу».
2. Користувач створює в системі варіант використання.
3. Завершення роботи прецеденту.
 |
| Альтеративні сценарії | 3.а Користувач не згодний. 3.а.1 Система відміняє створення варіанту використання. 3.а.2 Завершення роботи прецеденту. |

Таким чином функціональні вимоги до системи, що розроблюється у даній роботі, формалізовано графічно, у вигляді діаграми варіантів використання.

Також наразі маємо детальний опис процесу взаємодії користувача з системою, що зафіксований у вигляді основних успішних сценаріїв варіантів використання та альтернативних сценаріїв використання.

* 1. **Опис нефункціональних вимог**

Нефункціональні вимоги, на відміну від функціональних, регламентують внутрішні й зовнішні умови або атрибути функціонування системи, тобто вимоги визначають якою система повинна бути. Вони задають критерії якості роботи програмного забезпечення, а також накладають обмеження на проект чи реалізацію.

Стандарт ISO 9126 – це міжнародний стандарт, який визначає оціночні характеристики якості програмного забезпечення. Стандарт поділяється на чотири частини, що описують модель якості, зовнішні метрики якості, внутрішні метрики якості та метрики якості у використанні.

Модель якості встановлена в першій частині стандарту, ISO 9126-1, класифікує якість програмного забезпечення у шести характеристиках, які в свою чергу деталізовані відповідними властивостями.

Нижче, у табл. 3.15 наведені нефункціональні вимоги, які визначені згідно зі стандартом ISO 9126.

Таблиці 3.15 – Нефункціональні вимоги до системи

|  |  |  |
| --- | --- | --- |
| Характеристика | Властивість | Опис |
| Функціональність | Здатність до взаємодії | Програмна система здатна взаємодіяти з HTML документами. |
| Захищеність | Результати роботи програми зберігаються на персональному комп’ютері користувача. |
| Робота з програмою не потребує підключення до незахищених систем Wi-Fi. |
| Ефективність | Часові характеристики | Зміни до створюваних документів повинні вноситися не довше, ніж за 3 секунди. |

Продовження таблиці 3.15

|  |  |  |
| --- | --- | --- |
| Ефективність | Часові характеристики | Результати запитів повинні надходити до запитувача не довше, ніж за 5 секунд.  |
| Використання ресурсів | Для функціонування системи не є необхідним підключення до мережі Internet. |
| Надійність | Стійкість до відмов | При підвищенні кількості запитів система виконає коректно 98 запитів із 100. |
|  | Здатність до відновлення працездатності  | Система відновиться після відмови не більше, ніж за 2 хвилини. |
| Переносимість | Адаптованість | Система може використовуватися на персональному комп’ютері з будь якою операційною системою. |

Таким чином маємо вимоги до програмного забезпечення, що розроблюється у даній дипломній роботі. На даному етапі визначено набір вимог щодо властивостей, якості та функцій програмного забезпечення, що знаходиться у розробці.

Нефункціональні вимоги визначено згідно зі стандартом ISO 9126 та задокументовані у табличному вигляді.

**4 ПРОЕКТУВАННЯ СИСТЕМИ**

**4.1 Архітектура системи**

У попередньому розділах визначено функціональні та нефункціональні вимоги. На основі отриманих результатів тепер можна обрати відповідне архітектурне рішення.

Доцільним виглядає створення веб-сайту, оскільки реалізація даного рішення не потребує значних часових витрат, задовольняє усім вимогам якості та не накладає жодних обмежень щодо використання технологій для реалізації математичної моделі створення прецедентів і концептуальних класів, що забезпечує виконання функціональних вимог.

­Починати проектування веб-сайту варто з навігаційної карти. Така карта представлена на рис. 4.1. Стрілками позначені можливі переходи між сторінками, які було спроектовано за правилом «трьох кліків», суть якого полягає у тому, що користувач має здобути бажаний результат не більше, як за три переходи. Кожна сторінка веб-сайту реалізує один або декілька варіантів використання:

* «Редагувати пункт прецеденту», «Заповнити пункт прецеденту», «Створити альтернативний сценарій прецеденту» реалізується у «select page», «input page», «request page», «request service page», «request list page», «input-output page», «finish page», «call page», «create page», «free page», «repeat page».
* «Створити пункт варіанту використання» реалізується у«items page».
* «Редагувати концептуальний клас», «Заповнити концептуальний клас», «Створити концептуальний клас», «Створити документацію» реалізується у «conceptual classes page».
* «Зберегти документацію», «Переглянути документацію» реалізується у «report page».
* «Переглянути список раніше створених варіантів використання», «Створити варіант використання», «Заповнити преамбулу», «Редагувати преамбулу» реалізує сторінка «usecase page».



Рисунок 4.1 – Навігаційна карта системи

Для даного веб-сайту була обрана ієрархічна стратегія перехресних посилань, оскільки вона дозволяє послідовно відстежувати процес роботи у часі, та має можливість реалізації зворотного трасування. Також вона добре відповідає обраній пласкій архітектурі – взаємному розташуванню сторінок сайту, при якому потрібно мінімум кліків для потрапляння на внутрішню сторінку зі стартової.

Далі буде продемонстровано на рис. 4.2 діаграму взаємодії, що від початку до кінця відображає процес роботи користувача з системою.



Рисунок 4.2 – Діаграма взаємодії користувача з системою

Під назвою «some item» буде розумітися одна із сторінок: «select page», «input page», «request page», «request service page», «request list page», «input-output page», «finish page», «call page», «create page», «free page», «repeat page».

Виклики функцій між елементами «item page», «some item» та «conceptual classes page» можуть неодноразово повторюватись, поки користувач не отримає повний опис прецеденту, відображений у звіті.

Активність взаємодії між елементами «usecase page» та «report page» обмежена тільки кількістю раніше створених прецедентів. Це означає, що при першій взаємодії користувача з системою ці елементи між собою ніяк не пов’язані та взаємодіяти не можуть.

Функція «openItems()» може бути викликана також повторно елементами «some item» та «conceptual classes page». Оскільки функціональними вимогами та математичною моделлю передбачена можливість створення концептуального класу після створення пункту або уникнення цієї діяльності, повернення до вибору типу пунктів прецеденту можливо з будь-якого етапу роботи з системою, окрім перегляду звітів.

**4.2 Проектування інтерфейсу користувача**

Перед тим як безпосереднього створювати графічний інтерфейс користувача, необхідно зробити прототип майбутньої системи та проаналізувати зручність розроблюваного інтерфейсу для користувача.

Прототипування – це швидка реалізація базової функціональності системи. Цей процес дозволяє, в залежності від завдання, з різною кількістю деталей зобразити основні функції майбутньої системи.

Найпростіший та найшвидший спосіб представити бажаний графічний інтерфейс користувача – це створити прості схеми, основна мета яких буде полягати у тому, аби зрозуміти поточний хід думок щодо функціонування інтерфейсу.

Далі представлені прототипи тих вікон, що передбачають найчастіше використання та стосовно яких виникала більшість проблем к точки зору користувацького досвіду.

Такими вікнами визначено «usecase page», «items page», «conceptual classes page» та будь-яке вікно створення типу пункту. У якості прикладу вікна типу пункту обрано «create page».

Вікна «create page» та «conceptual classes page» мають подібну структуру, тому передбачається, що вирішення проблеми прототипування буде ідентичною чи схожою. Нижче на рис. 4.3 зображено прототип для «usecase page».



Рисунок 4.3 – Прототип для «usecase page»

Оскільки на даному етапі ще не створено жодного прецеденту, то у розділі «Створені раніше» відображено повідомлення «Немає раніше створених прецедентів».

Довгий список полів преамбули поміщено у внутрішнє вікно прокрутки, аби у користувача залишалася можливість на моніторі будь-якого розміру охоплювати цілісну картину створення прецеденту. Кнопка «Створити варіант використання» є статично закріпленим унизу сторінки елементом, оскільки є важливою функціональною деталлю та не повинна губитися при прокрутці сторінки.

Далі на рис. 4.4 зображено прототип для «items page».



Рисунок 4.4 – Прототип для «items page»

Оскільки на даному етапі ще не створено жодного пункту прецеденту, то у розділі «Результат» нічого не відображено. Список типів пунктів прецеденту поміщено у внутрішнє вікно прокрутки з тих же міркувань, що й у випадку полів преамбули у прототипі вище. Кнопка «Описати варіант використання» також є статично закріпленою унизу сторінки. Це стосуватиметься усіх подібних елементів у подальших прототипах.

Нижче на рис. 4.5 зображено прототип для «create page».

На рис. 4.6 зображено прототип для «conceptual classes page». Після того, як хоч один пункт прецеденту створено, його текст буде глобально відображений у системі в розділі «Результат». Якщо користувач далі обере «Обрати тип пункту», то він повернеться до прототипу для «items page». Якщо користувач забажає створити концептуальний клас для цього пункту, то він обере «Створити концептуальний клас».

Після опису концептуального класу сформований текст відобразиться у розділі «Текст пункту». Він не буде відображуватися глобально, на відміну від опису прецеденту. Дані у розділі будуть оновлені для кожного подальшого пункту прецеденту. На цьому кроці можна завершити роботу з опису прецеденту та отримати звіт, обравши «Завершити і отримати документацію».



Рисунок 4.5 – Прототип для «create page»



Рисунок 4.6 – Прототип для «conceptual classes page»

У звіті буде відображено усю створену користувачем інформацію щодо прецеденту та пов’язаних з ним концептуальних класів. Цей звіт можна локально зберегти на комп’ютер для подальшого використання, а також викликати із системи для перегляду на сторінці «usecase page», де у розділі «Раніше створені» тепер відображатиметься назва щойно створеного прецеденту. Натиснувши на цю назву можна перейти на сторінку зі звітом для перегляду.

Далі на рис. 4.7 зображено прототип для «report page».



Рисунок 4.7 – Прототип для «report page»

Нижче на рис. 4.8 зображено прототип зміненого вікна для «create page» із заповненим корисною інформацією розділом «Раніше створені».



Рисунок 4.8 – Прототип для зміненої «usecase page»

Таким чином, маємо спроектований графічний інтерфейс користувача, який у подальшому буде програмно реалізований у наступному розділі. Інтуїтивна зрозумілість інтерфейсу буде експериментально перевірена під час тестування системи та проведення експерименту.

Інтернет – найконкурентніший майданчик, і саме професійний UI/UX дизайн може вигідно виділити систему серед інших гравців ринку. Якісний web-дизайн привертає увагу широкої аудиторії та дозволяє істотно зміцнити позиції в веб-просторі, залучити нових користувачів.

При невідповідному дизайні цінна інформація залишиться непоміченою. Отже, веб-дизайн – це логіка, турбота про зручність та художнє оформлення в одному.

Інтерфейс користувача сайту повинен бути не лише красивим: весь його вміст має бути логічно пов'язаний, інформація для користувачів представлена правильно і зрозуміло, а всі елементи на сторінках мають підводити користувача до вчинення цільових дій.

User Experience – це емоції, почуття і досвід користувача, його враження, сприйняття системи. Якісний UX дизайн робить веб-сайт максимально зручним та інтуїтивно зрозумілим, а також забезпечує головне: щоб відвідувачі могли легко і у найпростіший спосіб описати функціональні вимоги та створити концептуальні класи. Структура сайту набуває логіки та зрозумілості, дуже доступно передаючи весь вміст сторінок.

User Interface – це те, що бачить користувач, зовнішній вигляд системи, його привабливість та фізичні характеристики. Система повинна мати відповідність між візуальним дизайном і своєю логічною структурою.

**5 РЕАЛІЗАЦІЯ ТА ВИПРОБУВАННЯ СИСТЕМИ**

Рішення використовувати веб-технології для розробки системи було обумовлене декількома факторами:

Користувач не потребує встановлення на свій комп’ютер додаткового програмного забезпечення. Для повноцінної роботи потрібен лише браузер, зазвичай поставляється разом з операційною системою, і доступ в Інтернет.

У корпоративному секторі економічно набагато вигідніше й ефективніше використовувати веб-сайт, ніж утримувати команду адміністраторів, що займаються установкою і налаштуванням десктопних додатків.

Веб-сайти не вимогливі до ресурсів і не пред'являють ніяких вимог до апаратної платформи. Це означає, що немає ніякої різниці, скільки мегабайт оперативної пам'яті встановлено на комп'ютері користувача і з під якої операційної системи він працює.

Немає ніяких проблем з підтримкою старих версій програм і зворотною сумісністю. Коли з'являється нова версія десктопного додатка, користувачам нерідко доводиться вирішувати проблеми, пов'язані з оновленням вже встановленої на їх машині копії.

Вищеозначені фактори забезпечують реалізацію таких нефункціональних вимог як кросплатформенність та апаратна переносимість системи.

Програмування графічного інтерфейсу у даній роботі виконується мовою HTML з використанням CSS, математична модель реалізовується на мові JavaScript:

1. HTML (англ. Hypertext Markup Language) – це мова тегів, якою пишуться гіпертекстові документи для мережі Інтернет.
2. CSS (англ. Cascading Style Sheets) — це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних.
	* + 1. JavaScript – динамічна, об’єктно-орієнтована прототипна мова програмування. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість змінювати вигляд веб-сторінок та керувати браузером.

Далі на рис. 5.1 показано схему алгоритму роботи варіанту використання «Створити концептуальний клас». У цьому алгоритмі найповніше розкривається функціональна поведінка системи і можна побачити як вона реагує на дії користувача.



Рисунок 5.1 – Схема алгоритму створення концептуального класу

Розроблену систему необхідно протестувати для визначення і усунення можливих помилок.

Тестування програмного забезпечення – процес дослідження, випробування програмного продукту, який має на меті перевірку відповідності між реальною поведінкою програми і її очікуваною поведінкою на кінцевому наборі тестів, обраних певним чином. Це перевірка відповідності заявлених до продукту вимог і реально реалізованої функціональності, що здійснюється шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Може оцінюватись:

1. відповідність вимогам, якими керувалися проектувальники та розробники,
2. правильна відповідь для усіх можливих вхідних даних,
3. виконання функцій за прийнятний час,
4. практичність,
5. сумісність з програмним забезпеченням та операційними системами,
6. відповідність задачам замовника.

Процес тестування повинен зароджуватися, коли починається процес розробки. Час, витрачений на усунення помилок, набагато коротший, коли тестування заплановано на початку фази проектування, ніж у її кінці. Системні вимоги повинні бути записані і узгоджені всіма сторонами, і ці специфікації вимог повинні використовуватися в якості основного плану тестування.

Модульне тестування – процес в програмуванні, що дозволяє перевірити на коректність одиниці вихідного коду, набори з одного або більше програмних модулів разом з відповідними керуючими даними, процедурами використання та обробки. Відноситься до тестів, які перевіряють функціональність певного розділу коду, зазвичай на функціональному рівні.

Такі типи тестів зазвичай пишуться розробниками під час роботи над кодом (стиль «білої скриньки»), щоб впевнитись, що дана функція працює так, як очікувалося. Одна функція може мати кілька тестів, щоб переглянути всі випадки використання коду. Модульне тестування саме по собі не може перевірити функціонування частини програмного забезпечення, а використовується щоб гарантувати, що основні блоки системи працюють незалежно один від одного.

Модульне тестування  виконується розробником, під час будівельної фази життєвого циклу розробки ПЗ. Модульне тестування спрямоване на усунення помилок проектування.

Залежно від очікуваної організації розробки, модульне тестування може включати статичний аналіз коду, аналіз потоку даних аналізу метрик, експертні оцінки коду, аналізу покриття коду та інші методи перевірки ПЗ.

Розглянемо повні і скорочені таблиці рішень, створені для розробки модульного тестування системи. За методом мінімізації функції за картами Карно, надалі будуть отримані скорочені таблиці рішень для функцій, що тестуються.

Таблиця 5.1 – Повна таблиця рішень для прецеденту «Створити прецедент»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Умова | Обрано «Створити концептуальний клас» | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Відправлено запит про існуючі концептуальні класи | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| Отримано відповідь на запит | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| Дія | Активація створення концептуального класу |  |  |  | ˅ | ˅ | ˅ | ˅ |
| Відображення відповіді |  |  | ˅ |  |  |  | ˅ |

Таблиця 5.2 – Скорочена таблиця рішень для прецеденту «Створити концептуальний клас»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Умова | Обрано «Створити концептуальний клас» | × | 1 | 1 |

Продовження таблиці 5.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Умова | Відправлено запит про існуючі концептуальні класи | 1 | × | 1 |
| Отримано відповідь на запит | 1 | × | × |
| Дія | Активація створення концептуального класу |  | ˅ | ˅ |
| Відображення відповіді | ˅ |  |  |

Таблиця 5.3 – Повна таблиця рішень для прецеденту «Заповнити концептуальний клас»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Умова | Обрано «Ввести данні у концептуальний клас» | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Виведено поля для заповнення | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| Поля заповнені | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| Дія | Дані введено |  |  |  | ˅ | ˅ | ˅ | ˅ |
| Дані зареєстровано |  |  | ˅ |  |  |  | ˅ |

Таблиця 5.4 – Скорочена таблиця рішень для прецеденту «Заповнити концептуальний клас»

|  |  |  |  |
| --- | --- | --- | --- |
| Умова | Обрано «Ввести данні концептуальний клас» | × | × |
| Виведено поля для заповнення | 1 | 1 |
| Поля заповнені | 1 | × |
| Дія | Дані введено |  | ˅ |

Продовження таблиці 5.4

|  |  |  |  |
| --- | --- | --- | --- |
| Дія | Дані зареєстровано | ˅ |  |

Таблиця 5.5 – Повна таблиця рішень для прецеденту «Заповнити пункт»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Умова | Обрано «Заповнити пункт» | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Запит відправлено | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|  Данні існують | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| Дія | Відповідь отримано |  |  | ˅ |  |  | ˅ | ˅ |
| Дані відображено |  |  | ˅ |  |  |  | ˅ |

Таблиця 5.6 – Скорочена таблиця рішень для прецеденту «Заповнити пункт»

|  |  |  |  |
| --- | --- | --- | --- |
| Умова | Обрано «Заповнити пункт» | × | × |
| Запит відправлено | 1 | 1 |
|  Данні існують | 1 | × |
| Дія | Відповідь отримано |  | ˅ |
| Дані відображено | ˅ |  |

Таблиця 5.7 – Повна таблиця рішень для варіанту використання «Обрати пункт»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Умова | Обрано «Обрати пункт» | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Обрано пункт | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|  Дане існує | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

Продовження таблиці 5.7

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Дія | Вибір зафіксовано |  |  | ˅ |  |  | ˅ | ˅ |
| Дане відображено |  |  | ˅ |  |  |  | ˅ |

Таблиця 5.8 – Скорочена таблиця рішень для варіанту використання «Обрати пункт»

|  |  |  |  |
| --- | --- | --- | --- |
| Умова | Обрано «Обрати пункт» | × | × |
| Обрано пункт | 1 | 1 |
|  Дане існують | 1 | × |
| Дія | Вибір зафіксовано |  | ˅ |
| Дане відображено | ˅ |  |

Таким чином було показано функціональне тестування функцій системи, які були визначені у вимогах до розробки системи. У якості методу тестування були обрані таблиці рішень як один з найбільш наочних і зручних методів.

Далі наведено таблиці тест-кейсів, що дозволяє перевірити коректність поведінки системи у відповідь на дії користувача.

Таблиця 5.9 – Тест-кейс перевірки коректності створення концептуального класу

|  |  |
| --- | --- |
| Кроки тест-кейса | Очікуваний результат |
| Вибрати «Створити концептуальний клас» | З’явиться форма для заповнення |
| Ввести дані концептуального класу | Дані відображені на дисплеї |

Таблиця 5.10 – Тест-кейс перевірки коректності введення даних

|  |  |
| --- | --- |
| Кроки тест-кейса | Очікуваний результат |
| Обрати «Створити пункт» | На дисплеї з’явиться форма для заповнення |
| Ввести данні | Дані відображаються на дисплеї |
| Натиснути на кнопку «Зберегти» | Дані реєструються у системі |

Таблиця 5.11 – Тест-кейс перевірки коректності вибору пункту

|  |  |
| --- | --- |
| Кроки тест-кейса | Очікуваний результат |
| Обрати пункт «Обрати пункт» | Відображається список пунктів |
| Обрати пункт | Відображається обраний пункт та реєструється у системі |
| Натиснути на кнопку «Описати варіант використання» | Система переводить користувача на вікно створення пунктів |

Таким чином було проведено тестування основного функціоналу системи за допомогою таблиць тест-кейсів. Тепер можна побачити та перевірити, що поведінка системи є коректною та адекватною відповіддю на дії користувача.

**6 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ**

Для доведення ефективності використання системи було проведено експериментальне дослідження розроблюваної системи, результати якого наведені у табл. 6.1.

До експерименту було залучено дві групи студенів. Кожна група складалася із десяти чоловік. Рівні знань кожного учасника відрізнялися, що визначалося за їх балами у рейтингових списках.

Кожен учасник використовував для експерименту різні предметні області, з якими працював під час переддипломної практики. До того ж, у правилах експерименту було дотримання умови створення не менше десяти варіантів використання, серед яких хоча б чотири мали б мати зв’язки «extend» або «include».

Було обрано десять предметних областей та розподілено так, аби учасники з різних груп з приблизно однаковими рейтинговими балами працювали над однаковими темами. Перша група учасників мала виконати завдання, поставлені у ході експерименту вручну, сподіваючись лише на власні сили та знання, отримані у ході навчання у вищому навчальному закладі та під час проходження переддипломної практики. У даному випадку присутній людський фактор. Учасники другої групи складали сценарії варіантів використання для формалізації функціональних вимог за допомогою методології створення прецедентів та концептуальних класів. У цьому випадку очікується зниження впливу людського фактору, зменшення кількості помилок та часових витрат на дану роботу.

Таблиця 6.1 – Результати експерименту

|  |  |  |
| --- | --- | --- |
| № | Без використання системи | З використанням системи |
| Час (хв.) | Помилки (шт.) | Час (хв.) | Помилки (шт.) |
| 1 | 83 | 14 | 47 | 6  |

Продовження таблиці 6.1

|  |  |  |
| --- | --- | --- |
| № | Без використання системи | З використанням системи |
| Час (хв.) | Помилки (шт.) | Час (хв.) | Помилки (шт.) |
| 2 | 92 | 5 | 43 | 2 |
| 3 | 53 | 8 | 31 | 5 |
| 4 | 78 | 13 | 46 | 4 |
| 5 | 46 | 7 | 35 | 3 |
| 6 | 79 | 9 | 49 | 6 |
| 7 | 52 | 11 | 34 | 5 |
| 8 | 47 | 5 | 31 | 1 |
| 9 | 86 | 12 | 44 | 4 |
| 10 | 99 | 6 | 50 | 2 |
| Середнє значення | ≈72 | 9 | 41 | 4 |

Далі наведено порівняльні гістограми за витраченим часом на рис. 6.1 та за кількістю помилок на рис. 6.2 по осі OY, та за кількістю учасників по осі OX для учасників, що використовували систему, та для тих, хто працював власноруч.

Можна побачити, що у більшості випадків, показники учасників експерименту що працювали власноруч та з використанням системи відрізняються майже удвічі на користь системи. За формулою 6.1 розраховано як у відсотках зменшилися показники витрат часу *Rч* та кількості помилок *Rп*.

|  |  |
| --- | --- |
| *R =* $100- \frac{N\_{2}}{N\_{1}} ×100$,*Rч* = 100 – $\frac{41}{72} ×100$ = 43%,*Rп* = 100 – $\frac{4}{9} ×100$ = 55.5% | (6.1) |

де *N1* – значення, отримане у випробуванні без системи,

 *N2* – значення, отримане у випробуванні з використанням системи.

Враховуючи, що об’єм вибірки малий, варто знайти похибку, аби збільшити достовірність результату. Було вирішено скористатися методом Стьюдента. Обрано високу вірогідність 0.95, та скориставшись таблицями критерію Стьюдента було отримано похибку у 2%. Отже отримані результати приймають наступні значення: *Rч* = 43% ± 2%, *Rп* = 55.5% ± 2%.

Рисунок 6.1 – Порівняльна гістограма за витраченим часом

Рисунок 6.2 – Порівняльна гістограма за кількістю помилок

Таким чином наочно продемонстровано ефективність системи та доцільність її використання при розробці функціональних вимог та прецедентів. Розрахунки результатів експерименту вказують на повноту математичної моделі інформаційної технології.

**7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ**

У будь-якій професії є свої певні вимоги стосовно безпеки експлуатації тих чи інших пристроїв, які використовуються в роботі. У зв'язку з цим була створена і розвивається наука про охорону праці і життєдіяльності людини.

Стан діяльності, за якого з визначеною ймовірністю виключено прояв небезпек або ж відсутня надзвичайна небезпека, визначають як «безпеку». ДСТУ 2293-99 визначає термін «безпека» як стан захищеності особи та суспільства від ризику зазнати шкоди.

Дипломна робота розглядає питання розробки інформаційної технології для створення прецедентів і концептуальних класів, виконання яких здійснюється працівниками галузі інженерії програмного забезпечення. Працівники цієї галузі при роботі з комп'ютером наражаються ряду небезпечних і шкідливих виробничих факторів: електромагнітних полів (діапазон радіочастот: ВЧ, УВЧ і СВЧ), інфрачервоного і іонізуючого випромінювань, шуму і вібрації, статичної електрики і ін.

Охорона праці – це регульована предметна область, для якої потрібно визначити види заходів та відповідні умови їх проведення, необхідні для успішної реалізації і завершення кваліфікаційної роботи магістра зі спеціальності інженерія програмного забезпечення.

Організація заходів із забезпечення безпечних умов праці та поведінки у надзвичайних ситуаціях засновується на наступних нормативних документах:

* ДСТУ 2293-99. Охорона праці. Терміни та визначення;
* НАПБ А.01.001-2015. Правила пожежної безпеки в Україні;
* ДСН 3.3.6.037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку;
* НПАОП 0.00-1.28-10. Правила охорони праці при експлуатації ЕОМ;
* ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень;
* НАПБ Б.03.001-2204. Типові норми належності вогнегасників;
* ДБН В.2.5-28-2006. Природне і штучне освітлення.

Вивчення найважливіших заходів з охорони праці та безпеки у надзвичайних ситуаціях для працівників галузі інженерія програмного забезпечення наведено у Додатку Б та висвітлює наступні питання:

1. організація та управління охороною праці на у компанії «Apostera»,
2. обґрунтування заходів з покращення умов охорони праці,
3. розрахунок системи повітрообміну,
4. надзвичайні ситуації та шляхи їх запобігання.

Розглянуто організацію, управління охороною праці у компанії «Apostera», навчання охороні праці, проведення інструктажів та розслідування нещасних випадків; проаналізовано умови праці на робочому місці інженера-програміста.

Розраховано систему повітрообміну на робочому місці інженера-програміста та категорію пожежонебезпечного приміщення офісної будівлі.

Запропоновано заходи з поліпшення повітрообміну на робочому місці інженера-програміста та заходи щодо пожежної безпеки у компанії «Apostera».

# ВИСНОВКИ

У результаті виконання роботи було зменшено час виконання проекту за рахунок швидкого формалізування функціональних вимог та створення концептуальних класів завдяки використанню системи, що була розроблена в межах даної роботи. Для досягнення цієї мети було розроблено алгоритми функціонування елементів системи та взаємодії між ними.

Експериментально визначено, що у порівнянні з часовими витратами на роботу без системи, часові витрати з її використанням зменшено на 43% ± 2%.

Було зменшено кількість помилок при написанні сценаріїв варіантів використання за рахунок автоматизації цієї діяльності. Для досягнення цієї мети було вдало інтегровано у систему методологію щодо створення прецедентів та концептуальних класів.

Експериментально визначено, що у порівнянні з кількістю помилок, що допускалися при роботі без системи, кількість помилок з її використанням зменшено на 55.5% ± 2%.

З огляду на вдалі результати експерименту, можна стверджувати, що використання розробленої в даній дипломній роботі системи підвищує продуктивність створення програмного забезпечення.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

Орлов С., Технологія розробки програмного забезпечення. – СПб.: Питер, 2002. – 464 с.

Вендров А.М., Проектування програмного забезпечення економічних інформаційних систем. – М.: Финансы и статистика, 2000. – 352 с.

Кунгурцев О.Б. Уточнение классификации пунктов сценариев вариантов использования / О. Б.Кунгурцев, Н. А. Новікова, М. Ю. Решетняк, Я. В. Черепініна. // Технічні науки та технології. – Харків, 2018. – С. 79.

Решетняк М. Ю. Автоматизация описания прецедентов / М. Ю. Решетняк, Я. В. Черепініна. // Матеріали сьомої міжнародної конференції студентів і молодих науковців «Сучасні інформаційні технології 2017». – Одеса, 2017. – С. 154.

Віргес К., Розробка вимог до програмного забезпечення. 2-е вид. / пер. з англ./ – М.: Русская редакция, 2004. – 575 с.

Леффінгуелл Д., Уідриг Д., Принципи роботи з вимогами. Уніфікований підхід. / пер. з англ./ – М.: Издательский дом «Вильямс», 2002.

Мацяшек Л.А., Аналіз вимог і проектування систем. Розробка інформаційних систем з використанням UML. – М.: Издательский дом «Вильямс», 2002. – 432 с.

Ларман К. Застосування UML та шаблонів проектування. 2-е вид. / пер. з англ./ – М. : Издательский дом «Вильямс», 2004. – 624 с.

Коберн А., Сучасні методи опису функціональних вимог до систем. / пер. з англ./ – М.: Лори, 2002. – 288 с.

Бек К., Екстремальне програмування. – СПб.: Питер, 2002. – 224 с.

Леноненков А.В. Об’єктно-орієнтований аналіз і проектування з використанням UML IBM и Rational Rose. –ИНТУТЕ.ру, Бином. 2006. – 320 с.

Дж. Рамбо, М. Блаха - UML 2.O. Об’єктно-орієнтоване  моделювання і разробка. 2-е вид. — СПб.: Питер, 2007. — 544 с.