ДИПЛОМНА РОБОТА
БАКАЛАВРА

ДРБ. ІП - 44.00.00.000 ПЗ

Група ІП-18-2

Хопта Назарій

2022

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерія програмного забезпечення

УДК 004.4

ДИПЛОМНА РОБОТА БАКАЛАВРА

Тема: "Імплементація інтерактивних моделей візуалізації програмного забезпечення"

(назва згідно з наказом ректора)

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДРБ.ІП - 44.00.00.000 ПЗ

(позначення)

Консультанти:

Студент

ІП-18-2 /Н.В.Хопта/
(шифр групи) (підпис) (дата) (розшифровка підпису)

Керівник роботи

асист. /Л.В.Саманів/
(посада) (підпис) (дата) (розшифровка підпису)

з питань нормоконтролю

доц., к.т.н /В.Я. Піх/
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту
Завідувач кафедри

проф., д.т.н. /В.І.Шекета/
(посада) (підпис) (дата) (розшифровка підпису)

2022
(рік)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
проф. В.І. Шекета
“ ____ ” _____2021 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Хопті Назарію Володимировичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Імплементація інтерактивних моделей візуалізації програмного забезпечення"

керівник проекту (роботи) Саманів Любов Василівна, асистент,

затверджені наказом вищого навчального закладу від “ 25 ” травня 2022 р. № _____

2. Строк подання студентом проекту (роботи) 12 червня 2022 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1. Огляд існуючих концепцій, рішень та сервісів в даній області
2. Проектування та розробка алгоритму функціювання
3. Програмне рішення власного програмного продукту
4. Представлення інтерфейсу

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Ґратчаста структура ресурсу (рис. 2.3, ст. 42)
2. Внутрішня структура веб-ресурсу (рис. 2.4, ст. 42)
3. Приклад мережі Петрі (рис. 3.1, ст. 62)
4. Приклад орієнтованого графа (рис. 3.7, ст. 66)
5. Діаграма класів (рис. 3.16, ст. 70)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **17 травня 2022 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення теми дипломної роботи	01.02.2022	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	01.03.2022	виконано
3	Побудова моделі або алгоритму власного рішення	01.04.2022	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	01.06.2022	виконано
5	Тестування програмного забезпечення	08.06.2022	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2022	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

В дипломній роботі представлено інтерактивний підхід візуалізації програмного забезпечення для налаштування програмного забезпечення. Метою цього методу є підвищення зручності використання системи та підвищення продуктивності системи за рахунок скорочення часу, витрат і зусиль, витрачених на розробку програмних систем із використанням традиційних методів розробки. Було проведено дослідження зручності використання на реальних користувачах, щоб порівняти існуючу систему та запропоновану систему, а також перевірити, чи запропонована система покращує роботу користувача можливість навчання реальним користувачам.

КЛЮЧОВІ СЛОВА: ГНУЧКА МЕТОДОЛОГІЇ, ІТЕРАКТИВНИЙ ПІДХІД, МЕНЕДЖМЕНТ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МЕТОДОЛОГІЯ, ГНУЧКІСТЬ

ANNOTATION

The thesis presents an interactive software visualization approach for software configuration. The goal of this method is to improve system usability and improve system performance by reducing the time, cost, and effort spent on developing software systems using traditional development methods. A usability study was conducted on real users to compare the existing system and the proposed system, and to test whether the proposed system improves the user experience by providing training to real users.

KEY WORDS: FLEXIBILITY OF METHODOLOGY, ITERACTIVE APPROACH, MANAGEMENT, SOFTWARE, METHODOLOGY, FLEXIBILITY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 7
ВСТУП 8

1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА СЕРВІСІВ В ПРЕДМЕТНІЙ ОБЛАСТІ 10
1.1 Традиційна методологія програмної інженерії 10
1.2. Поняття лінії програмного забезпечення 14
1.3. Сервісно-орієнтована архітектура 16
1.3 Інтеграція технологій SOA та SPL 18
1.5. Визначення вимог до програмного забезпечення 21

2 АЛГОРИТМІЧНІ ТА ФОРМАЛЬНІ РІШЕННЯ ДЛЯ ЗАДАЧ ЗДІЙСНЕННЯ ВІЗУАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 27
2.1 Візуалізація програмного забезпечення 27
2.2. Етап візуальних уявлень при розробці 29
2.3. Техніки візуалізації програмного забезпечення 30
2.4. Моделі типів візуалізацій 33
2.5. Структура запропонованого методу візуалізації 37
2.6. Модель контекстного керування 38

3 ПРОГРАМНА ТА АЛГОРИТМІЧНА РЕАЛІЗАЦІЯ МЕТОДІВ ВІЗУАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 45
3.1. Реалізація системи 45
3.2. Дослідження процесу юзабіліті 52
3.3. Пропонований метод тестування юзабіліті 56

3.4. Опис завдання 57
3.5. Завдання для верифікації 58
3.6. Розробка анкети учасника 60

ВИСНОВКИ 71
СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА 73
БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІП – ітеративний процес

БД – База даних

СУБД - Система управління базами даних

ПЗ - Програмне забезпечення

ВСТУП

На початкових етапах еволюції розробки програмного забезпечення не було особливого акценту на можливості повторного використання програмних кодів. Вартість програмного забезпечення для комп'ютерної системи була незначною порівняно з вартістю використовуваного обладнання. Але з еволюцією все більш складних програмних систем можливість повторного використання програмних рішень стає набагато важливішим.

Програмне забезпечення стало набагато важливішою частиною будь-якої комп'ютерної системи. Причиною цього є гнучкість програмного забезпечення в модифікації системи, а також сила програмного забезпечення в додаванні нової функціональності в систему, яку, можливо, було б важко виконати без неї і лише шляхом модифікації апаратного забезпечення. У зв'язку з цим збільшенням розміру та складності програмної системи виникає потреба скоротити витрати та зусилля, пов'язані з поглибленням та налаштуванням цих систем. Тому для того, щоб зробити процес виробництва системи набагато ефективнішим, використовується концепція розробки програмних продуктів .

Використовуючи лінійку програмних продуктів, можна розробляти різні програмні додатки та інші програмні продукти шляхом створення багаторазових програмних рішень і повторного використання цих програмних рішень. Використовуючи такий підхід можна отримати деякі переваги, такі як зниження витрат і часу на розробку, підвищення якості, подолання еволюції та складності тощо . Також необхідно зробити цей процес розробки менш складним і більш зручним для користувачів. Користувачі, які використовують ці системи, мають різний рівень знань і можуть бути з різних доменів. Програмні візуалізації можна використовувати, щоб покращити розуміння системи і, таким чином, покращити загальну зручність використання системи.

Останнім часом доступно дуже багато методів та інструментів візуалізації програмного забезпечення, але важливо вибрати найбільш підходящий для

відповідної діяльності в процесі розробки програмного забезпечення, щоб зробити найбільш ефективну візуалізацію для конкретної програмної системи .

За останні кілька десятиліть процес розробки програмного забезпечення радикально змінився. Процес розробки програмного забезпечення пройшов довгий шлях від програмних систем з кількома рядками коду до програмних систем з мільйонами рядків коду. Незважаючи на те, що програмні системи ставали все більшими і більшими, і зараз вони є гігантськими, але, як ми бачимо, у більшості випадків існує багато програмних систем, які не так відрізняються від інших, коли ми бачимо їх на нижчих рівнях. Ці подібності на меншому рівні можуть бути використані для розробки програмних компонентів для багаторазового використання, які можна використовувати для поєднання та розробки більш складної програмної системи. Розробка лінійки програмних продуктів — це парадигма для розробки програмних додатків із багаторазовими програмними активами, які адаптуються до індивідуальних потреб клієнтів . Такий підхід допомагає знизити витрати на розробку програмних систем, оскільки нам не доведеться щоразу винаходити велосипед.

За допомогою повторного використання послуг і застосування методів на основі SOA в інженерії ПЗ, особливо методи семантичного веб-сервісу (наприклад, автоматичне виявлення та складання сервісів) , мета автоматизації розробки програмного забезпечення може бути досягнута. Але це не допомагає всім користувачам, оскільки всі користувачі ніколи не можуть мати однаковий рівень розуміння системи, яку вони намагаються розробити.

1 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА СЕРВІСІВ В ПРЕДМЕТНІЙ ОБЛАСТІ

1.1 Традиційна методологія програмної інженерії

У індустрії програмного забезпечення через підвищену складність і великі розміри програмних систем, які сьогодні є, акцент робиться на багаторазовому використанні коду. Код, написаний для однієї програмної системи або програми, повторно використовується для подібної програми, і код не повністю пишеться з нуля.

Концепція SOA використовується в лінійці програмних продуктів; це зробить масову настройку програмного застосунку шляхом повторного використання програмних рішень, що може бути дуже корисним, особливо з точки зору часу та зусиль. У лінійці програмних продуктів концепція сервісно-орієнтованої архітектури використовується для розробки автоматизованої системи програмного забезпечення для вибору та відповідності послуг багаторазового використання для створення нових додатків. Служби слабо пов’язані, щоб дозволити їм спілкуватися один з одним. Послуги можуть бути складені на основі вибору, щоб отримати найкращі послуги.

Виходячи з цього факту, попередньо був розроблений інтерфейс на основі Petri-Net, який взаємодіє з користувачем, який хоче налаштувати програмне забезпечення, природною мовою та автоматично виконує процес виявлення вимог на основі онтології це. Система візуалізації використовує Petri-Net, щоб надати користувачеві візуалізації. Онтологія представляє знання про особливості продукту, а також їх бізнес-логіку. Він представляє спільні риси та відмінності між групою споріднених рішень і таким чином спрямовує систему діалогу на виконання вимоги .

Проблема полягає в тому, що система може використовуватися самими різними користувачами, і обсяг знань цих користувачів дуже різниться. Система візуалізації, заснована виключно на Petri-Net, не дуже корисна для користувачів

із меншими технічними знаннями або користувачів, яких більше цікавить бізнес-аспект програмного забезпечення. Рівень досвіду, знань та сфера інтересів у цих користувачів дуже різниться. Але візуалізація програмного забезпечення для всіх цих користувачів однакова і показує однаковий тип інформації. Тому слід вивчати різні типи користувачів і відповідно до потреб цих користувачів вибирати різні методи візуалізації, які найкраще підходять для їхнього рівня та предметних знань. Це дослідження визначає необхідність вивчення різних типів користувачів і найкращих методів візуалізації для цих користувачів.

Зрештою, слід провести дослідження юзабіліті системи, щоб вивчити корисність та ефективність системи та обґрунтувати використання нового методу. Систему слід дослідити, щоб перевірити, чи покращує вона здатність до навчання, ефективність та задоволеність користувачів системою та чи зменшує рівень помилок.

Впровадження інтерактивного підходу зменшить витрати на роботу з системою, що в подальшому призведе до скорочення часу та витрат. Інтерактивна візуалізація програмного забезпечення буде реалізована в графічному інтерфейсі системи, що покращить зручність використання системи.

Користувачі поділяються на чотири групи на основі їхнього рівня знань про розробку програмного забезпечення та процес налаштування програмного забезпечення. Модель контекстного керування використовується для класифікації користувачів на основі їхніх дій. Цим користувачам будуть надані різні типи інтерфейсів візуалізації відповідно до їхніх знань.

Налаштування програмного забезпечення на основі лінійки програмного забезпечення все ще знаходиться на початковій стадії, і зусилля, пов’язані з цим процесом, є величезними. Інтерактивний підхід до візуалізації програмного забезпечення на основі користувача для налаштування програмного забезпечення зменшить зусилля, час і вартість процесу налаштування програмного забезпечення. Це дозволить користувачам із меншими знаннями процесу розробки програмного забезпечення налаштувати програмне

забезпечення для них або зрозуміти хід програмного забезпечення, що розробляється.

Інтерактивний підхід візуалізації програмного забезпечення покращує взаємодію людини з комп'ютером, надаючи різні візуалізації різним користувачам для кращого розуміння. Він надає користувачам структуру, яка є більш дружньою та інтерактивною за своєю природою, що покращує загальний досвід роботи з користувачем.

Незалежно від того, як виконується розробка програмного забезпечення або який підхід використовується, основним завданням є вирішення проблеми. Спосіб розробників вирішує проблеми, як правило, однаковий, незалежно від того, яка проблема та який підхід. Розв'язання проблеми включає чотири основні дії: вимоги - збір і документування деталей проблеми; аналіз - розуміння проблеми досить детально, щоб забезпечити правильне рішення; проектування - знаходження та конкретизація оптимального рішення задачі; і реалізація (за потреби) - реалізація рішення в будь-якій формі .

Істотною проблемою розробки програмного забезпечення є те, як реалізувати за певними технологіями та з певними обмеженнями конкретну систему обробки інформації. Хоча існують пов'язані проблеми розуміння домену, вони, як правило, не пов'язані з програмним забезпеченням. Можна стверджувати, що незалежно від того, яка парадигма чи підхід прийнято до розробки програмного забезпечення, кожна діяльність із вирішення проблеми має бути здійснена певною мірою. По суті, кожен розробник проходить цикл вимог, аналізу, проектування та впровадження, будь то протягом тривалого періоду, тижня, дня, години чи хвилин, і незалежно від того, документують вони результати, обговорюють їх з іншими на дошку, або просто розглянути їх неформально в своїй голові. Від цієї діяльності не уникнути .

Життєвий цикл розробки програмного забезпечення (SDLC) дає перспективу високого рівня того, як різні види діяльності з вирішення проблем можуть бути поетапно опрацьовані окремою особою або командою, що

займається розробкою програмного забезпечення . Найпопулярнішою традиційною моделлю є модель водоспаду. Waterfall SDLC був представлений доктором Вінстоном В. Ройсом як метод розробки програмного забезпечення. Він передбачає послідовне повне завершення кожної фази, а потім перехід до наступної фази. У випадку цієї моделі водоспаду прогрес часто розглядається як постійний спад, і тому його називають методом водоспаду. Модель водоспаду є найбільш часто використовуваною традиційною моделлю для розробки програмного забезпечення та налаштування програмного забезпечення. На рисунку 1.1 показана модель водоспаду структура:

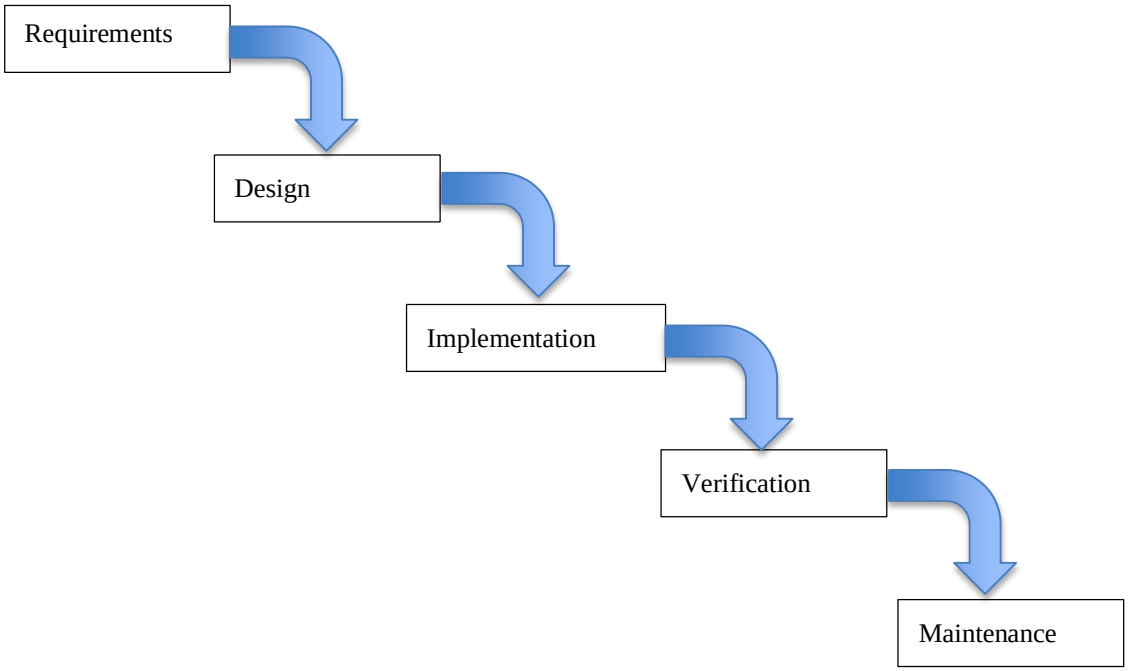


Рисунок 1.1 - Модель водоспаду

Послідовні фази в моделі водоспаду:

• Збір та аналіз вимог: усі можливі вимоги системи, яка буде розроблена, фіксуються на цьому етапі та задокументовані в документі специфікації вимог.

• Проектування системи: на цьому етапі вивчаються специфікації вимог першого етапу та готується проект системи. Проектування системи допомагає визначити вимоги до обладнання та системи, а також допомагає визначити загальну архітектуру системи.

• Впровадження: на основі даних проектування системи система спочатку розробляється в невеликих програмах, які називаються блоками, які інтегруються на наступному етапі. Кожен блок розроблено та перевірено на його функціональність, що називається модульним тестуванням.

• Інтеграція та тестування: усі блоки, розроблені на етапі впровадження, інтегруються в систему після тестування кожного блоку. Після інтеграції вся система перевіряється на наявність несправностей і збоїв.

• Технічне обслуговування: є деякі проблеми, які виникають у середовищі клієнта. Щоб усунути ці проблеми, випущені виправлення. Також для покращення продукту випущено кілька кращих версій. Технічне обслуговування здійснюється, щоб забезпечити ці зміни в середовищі клієнта.

1.2. Поняття лінії програмного забезпечення

На початкових етапах еволюції розробки програмного забезпечення не було особливого акценту на можливості повторного використання програмних артефактів. Вартість програмного забезпечення для комп'ютерної системи була незначною порівняно з вартістю використовуваного обладнання. Але з еволюцією все більш складних програмних систем можливість повторного використання програмних артефактів ставала все більш важливим фактором.

Традиційно програмне забезпечення, яке використовувалося для продуктів, було дуже невеликим і простим. Щоб модифікувати та виробляти новий продукт, раніше було набагато простіше та дешевше копіювати, транспортувати чи замінювати програмне забезпечення, ніж апаратне забезпечення. Основна увага при створенні продукту була зосереджена на апаратному і програмному забезпеченні, яке не відігравало ключової ролі у створенні продукту.

Проте зараз програмне забезпечення відіграє дуже важливу роль у будь-якій системі. Причиною цього є гнучкість програмного забезпечення в модифікації системи, а також сила програмного забезпечення в додаванні нової

функціональності в систему, яку, можливо, було б важко виконати без неї і лише шляхом модифікації апаратного забезпечення. Тому для того, щоб зробити процес виробництва системи набагато ефективнішим, буде розглянута концепція інженерії програмних продуктів.

Лінія програмних продуктів – це парадигма для розробки програмних додатків і програмних продуктів шляхом створення частин, які можна використовувати повторно, і їх повторного використання. Для цього використовується масова кастомізація, що означає велике виробництво товарів з урахуванням індивідуальних вимог замовника. Для цього ми повинні зосередитися на спільних рисах і відмінностях в додатках (з точки зору вимог, архітектури, компонентів і тестових артефактів) лінійки продуктів, які будуть моделюватися загальним способом. Використовуючи SPL, можна отримати деякі переваги, такі як зниження витрат і часу на розробку, підвищення якості, подолання еволюції та складності тощо.

Парадигма розробки лінійки програмних продуктів складається з двох процесів: розробка домену та розробка додатків. Domain Engineering створює платформу та визначає спільні риси та відмінності лінійки продуктів. Наша основна увага в цій роботі зосереджена на процесі розробки предметної області. Доменна інженерія — це процес розробки лінійки програмних продуктів, у якому визначаються та реалізуються спільність та мінливість лінійки продуктів..

Application Engineering отримує програму від платформи, яка створена за допомогою доменної інженерії. Інжиніринг додатків — це процес розробки лінійки програмних продуктів, у якому додатки лінійки продуктів будуються шляхом повторного використання артефактів предметної області та використання мінливості лінійки продуктів .

Незважаючи на те, що було проведено багато досліджень щодо переваг використання ліній програмного забезпечення для розробки програмного забезпечення, а також того, як визначати та розробляти лінійки продуктів,

доступні лише деякі підходи та інструменти для створення продуктів і способів використання лінійки продуктів .

У порівнянні з зусиллями, які витрачаються на розробку та моделювання ліній програмного забезпечення, доступна незначна підтримка для підвищення їхнього використання на практиці. Без ефективних підходів до використання ліній продукції, зокрема автоматизованих, SPL не міг би бути широко прийнятим у промисловості. Іншими словами, вони матимуть більшу академічну цінність, ніж практичну.

1.3. Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура — це архітектурна модель програмного забезпечення, яка використовується для автоматизації композиції сервісу. Архітектура, орієнтована на обслуговування, розділяє єдину логіку автоматизації бізнес-програм на кілька менших блоків логіки. Ці менші одиниці простіші, відмінні та поширені за своєю природою. Нещільне з'єднання, абстракція та можливість повторного використання бізнес-функцій є основними перевагами сервіс-орієнтованої архітектури.

«Сервісно-орієнтована архітектура — це архітектурний підхід інформаційних технологій, який підтримує створення бізнес-процесів із функціональних одиниць, визначених як послуги».

У сервіс-орієнтованій архітектурі багаторазове використання коду вважається дуже важливим, і, таким чином, усі менші окремі блоки логіки можна повторно використовувати в кількох різних програмах. Один або група цих менших логічних одиниць, які можуть працювати незалежно, називаються Службами. Архітектура, орієнтована на обслуговування, надає кілька методів створення цих служб для побудови повного бізнес-процесу. Композиція послуг – це процес об'єднання послуг на основі вибору послуг.

Сервіси — це модулі функціональних можливостей бізнесу або програми. Архітектура, орієнтована на обслуговування, складається з сервісів, які спільно

використовують і повторно використовують в IT-мережі, і вони взаємодіють один з одним.

Цей зв’язок може здійснюватися шляхом передачі даних між службами або шляхом координації двох або більше служб для виконання спільної діяльності .

У сервіс-орієнтованій архітектурі багаторазове використання коду вважається дуже важливим, і, таким чином, усі менші окремі блоки логіки можна повторно використовувати в кількох різних програмах. Один або група цих менших логічних одиниць, які можуть працювати незалежно, називаються Службами. Архітектура, орієнтована на обслуговування, надає кілька методів створення цих служб для побудови повного бізнес-процесу. Композиція послуг – це процес об’єднання послуг на основі вибору послуг.

Зазвичай більшість підходів використовують SOAP.

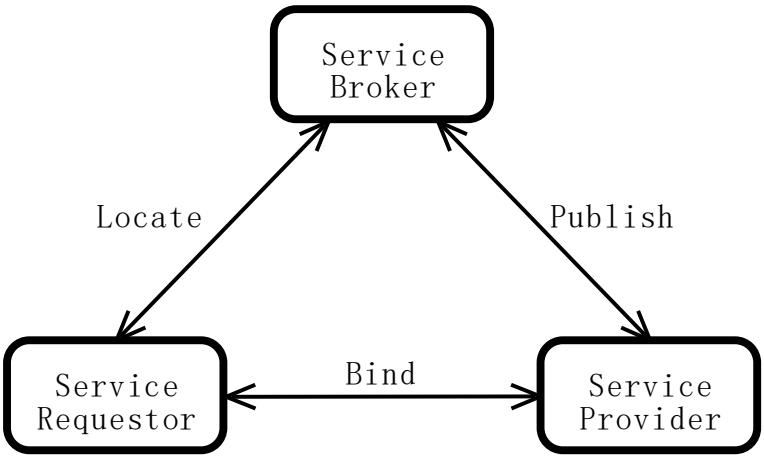


Рисунок 1.2 - Основні компоненти SOA

Архітектура, орієнтована на обслуговування, складається з трьох типів агентів. Це:

1. Постачальник послуг

Постачальник послуг є компонентом, відповідальним за створення та публікацію служби в реєстрі. Це також робить послугу доступною для інших компонентів через Інтернет.

2. Запитувач послуги

Запитувач послуг виконує виявлення служби в реєстрі служб, щоб знайти потрібну послугу, а потім отримати доступ до неї.

3. Брокер послуг

Компонент Service Broker допомагає постачальникам послуг і запитувачам послуг знаходити один одного, діючи як реєстр послуг.

Ці агенти SOA виконують операції пошуку, зв'язування та публікації. Постачальник послуг розробляє та публікує описи послуг та інформацію про доступ до них у реєстрі послуг. Запитувач послуг намагається знайти найбільш підходящу послугу в реєстрі послуг і за допомогою доступної інформації доступу в реєстрі послуг зв'язує необхідну послугу з постачальником послуг для виклику необхідних послуг. Архітектура, орієнтована на обслуговування, допомагає досягти слабкого зв'язку між сервісами, абстрагуванням і повторним використанням бізнес-функціональних можливостей.

1.3 Інтеграція технологій SOA та SPL

Однією з головних переваг SOA є гнучкість створення IT-рішень, які можуть швидко та економічно реагувати на зміни бізнес-вимог. SOA обіцяє бачення, коли постачальники послуг пропонують свої послуги, а запитувачі послуг шукають і відкривають ці послуги на основі їхніх бізнес-потреб. У SOA постачальники послуг зазвичай відокремлені від запитувачів послуг, тому запитувачі та постачальники можуть змінюватися незалежно один від одного. Крім того, розробка додатків зазвичай здійснюється шляхом складання служб, а не розробки компонентів і коду. Крім того, міжорганізаційне співробітництво може відбуватися децентралізованим і дуже розподіленим чином. Як наслідок, мінливість у SOA має інші проблеми, з якими потрібно мати справу, ніж у системах без SOA.

Хоча існує багато відмінностей між типовими лінійками програмних продуктів і сервісно-орієнтованими архітектурами, концепції SPL можна

використовувати для моделювання мінливості SOA. Оскільки послуги в SOA можуть використовуватися різними клієнтами з різною функціональністю, моделювання змінності SOA може скористатися методами моделювання змінності SPL. Системи, орієнтовані на обслуговування, можна моделювати як сімейства послуг, подібно до концепції SPL. Основною метою SPL є розробка програм-членів SPL, орієнтована на повторне використання, за допомогою використання повторно використовуваних активів на всіх фазах життєвого циклу розробки. Ця мета подібна до мети SOA, де поширеною темою є гнучка розробка додатків .

Однак у SOA не вистачає підтримки високої настройки та систематичного запланованого повторного використання. Це означає, що можна використовувати певні сервіси для розробки програмного забезпечення, але якщо відбудуться будь-які зміни в замовленні або учасниках послуг компонування сервісів, які не розроблені так, щоб їх можна було настроювати та використовувати повторно, не підтримають змінність.

Таким чином, інженерія SPL, яка в основному має принцип мінливості, налаштування та систематичного планованого повторного використання, може використовуватися для допомоги SOA для кращої функціональності та досягнення цих переваг . Крім того, інтеграція концепцій SPL і SOA дає можливість повторного використання існуючих сервісів замість постійного їх розвитку з нуля .

Поєднання сервісно-орієнтованої архітектури та лінійки програмних продуктів покращує практичні реалізації лінійки програмних продуктів. Це робить процес розробки програмного забезпечення більш ефективним і покращує якість кінцевого програмного продукту. Це зменшує витрати та зусилля на розробку, покращує час виходу на ринок, налаштовує застосування до конкретних потреб клієнтів або сегменту ринку та забезпечує конкурентні переваги. Інтеграція концепцій SPL і SOA дає можливість повторного використання існуючих сервісів замість постійного їх розвитку з нуля.

Як висновок для цієї частини, концепції SPL і SOA жодним чином не виключають один одного, а там, де вони відрізняються, вони діють як доповнення один одного.

Розробка вимог визнана найбільш важливою частиною всього процесу розробки програмного забезпечення. Як правило, понад 40% помилок у програмному проекті виникають через вимоги, тоді як на їх виправлення потрібно в 10 разів більше витрат, ніж інші помилки.

«Інженерія вимог (RE) — це систематичний процес розробки вимог через ітераційний кооперативний процес аналізу проблеми, документування результатів спостережень у різноманітних форматах представлення та перевірки точності отриманого розуміння».

Традиційні методи розробки вимог на основі процесів або сценаріїв попередньо визначають групу процесів і відповідних рекомендацій. Інженерно-технічні заходи та результати виконання вимог здійснюються відповідно до інструкцій. Однак дуже часто, коли процеси тривають, деяка важлива інформація ще недоступна. Тому інженерам доводиться повторювати процеси, що призводить до затримки проекту та додаткових витрат .

Методи розробки вимог, засновані на процесах, попередньо визначають групу процесів і відповідних рекомендацій, а діяльність з розробки вимог і результати виконуються відповідно до цих інструкцій. Але в більшості випадків вимоги не можуть бути заморожені до початку фази розробки. Більш того, дуже часто під час цих процесів якась важлива інформація недоступна. У цих випадках процес потрібно повторити, коли ця інформація буде доступна, це збільшує вартість розробки, а також затримує проект.

На відміну від традиційної розробки вимог на основі процесів, розробка вимог, керована знаннями, як нова парадигма інженерії вимог, виконується під керівництвом предметних знань. В результаті прихована інформація може бути отримана та використана для керування процесом розробки вимог. Очікується, що результат буде більш зрілим і повним, а переробка може бути значно

зменшена . Розробка вимог на основі онтології — це метод розробки вимог, керований знаннями, який має такі основні переваги:

1. Він забезпечує офіційне представництво як документів вимог, так і знань

2. Описує проблемну область з різним ступенем формалізації та виразності

3. Він добре підходить як еволюційний підхід

4. Використовується для підтримки управління вимогами та покращення вимог простежуваність артефактів .

Попередня теза з тієї ж дослідницької групи, яку проводив Чжан , має назву «Інтерактивний підхід до визначення вимог на основі онтології». У цьому проекті був запропонований підхід до визначення вимог для розробки SPL на основі SOA як моделі програмування для реалізації інтерактивної інженерії вимог.

1.5. Визначення вимог до програмного забезпечення

Одним із найважливіших завдань інженерії вимог під час розробки програмного забезпечення є визначення вимог. Дослідження показують, що основною причиною проблем у програмних проектах є неадекватна розробка вимог . Отже, основною передумовою лінійки програмних продуктів, яка є парадигмою розробки програмного забезпечення, є процес виявлення вимог, який показує спільні та відмінні риси вимог.

Існують різні методики, які можна використовувати для виявлення вимог. Ці методи бувають або розмовними, які в основному проводяться шляхом інтерв'ю з двома або більше людьми, спостережливими, які можна зробити, спостерігаючи за людьми, коли вони виконують свою рутинну роботу, аналітичними, що означає вивчення наявної документації чи знань, отриманих у результаті розмови чи спостереження та синтетичних методів. який поєднує бесіду, спостереження та аналітичні методи в єдиний метод. На практиці ці методики не застосовуються належним чином .

У зазначено, що корисні, корисні та бажані програмні продукти створюються за допомогою дизайну взаємодії. Однак розробники програмного забезпечення не виграють від дизайну взаємодії. Інструменти, які розробники програмного забезпечення використовують для розробки, є недостатніми і непривабливими для них. Хоча важливість використання концепції взаємодії людини та комп'ютера (HCI) у процесі розробки програмного забезпечення (SDP) не дуже зрозуміла для багатьох розробників програмного забезпечення, експерти HCI намагалися показати, що інтеграція між цими двома способами може призвести до кращого задоволення користувачів, отриманих від SDP, орієнтований на користувача. Проте впровадження інтерактивної парадигми програмної інженерії все ще залишається проблемою. Одна з можливих ідей — скористатися перевагами концепцій SOA і SPL. SOA можна використовувати, щоб полегшити роботу розробникам програмного забезпечення, представивши послуги як слабо пов'язані функціональні можливості програмного забезпечення, що усуває складність нижнього рівня. З іншого боку, SPL корисний для управління змінною інженерією програмного забезпечення. В інтерактивній інженерії програмного забезпечення машини можна використовувати, щоб направляти користувачів у виборі програмних активів для багаторазового використання та реалізації програми-кандидата шляхом створення замовлених послуг.

Попередня теза з тієї ж дослідницької групи, яку проводив Чжан, має назву «Інтерактивний підхід до визначення вимог на основі онтології». У цьому проекті був запропонований підхід визначення вимог для розробки SPL на основі SOA як моделі програмування для реалізації інтерактивної вимоги.

Запропонована інтерактивна модель являє собою діалогову систему, яка взаємодіє з користувачами природною мовою. Принцип роботи системи діалогів полягає в тому, що вона витягує та аналізує вирази, створені людьми-користувачами, щоб виконати завдання, і відповідно генерує вираз природною

мовою для користувача. Тому діалогова система може бути зручним способом взаємодії людини і машини.

У попередній запропонованій системі діалогу задачі заповнення слотів розглядаються для процесу виявлення вимоги, в якому користувач знає про цілі та має інформацію про виконання завдання. Ці завдання будуть виконуватися на основі бази знань діалогової системи. Онтологія представляє загальне знання в межах області. Це означає, що він надає спільний словниковий запас для побудови понять, об'єктів та їхніх властивостей і відносин домену чи завдання, що може спричинити спільне розуміння структури інформації між людьми або програмними агентами. За допомогою онтології загальні поняття домену можуть бути визначені експертами, а знання можуть бути використані людьми з будь-яким досвідом і без професійної підготовки .

Для розвитку онтології необхідно визначити поняття в предметній області та встановити між ними ієрархічний порядок. Також слід визначити слоти та дозволені положення для цих слотів. В кінці слід заповнити екземпляри та значення для слотів екземплярів .

Модель, розроблена в попередньому проекті, інтегрує знання інженерних вимог із знаннями, орієнтованими на обслуговування. Оскільки SOA інкапсулює функціональні можливості програми у слабо пов'язані послуги, програмні додатки можуть бути реалізовані шляхом виявлення, створення та виклику служб у SOA. Онтологія сервісів робить можливим автоматичне виявлення та компонування сервісів. В онтології існує клас під назвою ServiceProfile, який містить характеристики послуг і використовується для відповідності запитам клієнта. Таким чином буває, що з метою виявлення служб, ServiceProfile запитувача автоматично буде збігатися з ServiceProfile провайдера через семантичну відповідність можливостей , і якщо збіг буде успішним, потрібні послуги будуть знайдені.

У сфері виявлення вимог вимоги можна класифікувати на три категорії: функції, якість та м'які цілі. Кожна з цих категорій має різні ролі в системі, а

також для всіх визначено інший фактор, який називається рангом, який необхідний для керування процесом виявлення вимог і виражається в моделі онтології. Функції – це функціональні можливості в системі, які користувач може замовити. Якість - це нефункціональне обмеження, яке накладається на функцію. Програмні цілі – це нефункціональні обмеження, що накладаються на все системне середовище. Між кожним із цих трьох типів вимог існують певні відносини, такі як узагальнення, декомпозиція, опора, суперечність, асоціація, hasRank та недійсні. Ці відносини будуть коротко обговорені таким чином:

• Узагальнюючий зв'язок визначається, щоб показати, що екземпляр функції, якості і Softgoal також є прикладом вимоги.

• Коли вимога розкладається на у, у є менш складною вимогою того ж типу, що й х.

• Вимога х покладається на вимогу у, це означає, що реалізація х залежить від реалізації у.

• Коли вимога х і вимога у суперечать, це означає, що вони не повинні бути реалізовані один з одним у програмному забезпеченні продукту одночасно.

• Функція х асоціюється з якістю у.

• Відношення HasRank показує, що вимога х має унікальний ранг r.

• Недійсне відношення показує, що існує недійсний зв'язок між вимогами х та у.

Для створення екземпляра моделі онтології, спочатку всі ці відносини повинні бути встановлені між доступними вимогами, і наступна процедура покаже екземпляр онтології :

1. Буде визначено основні функції, які є коренями дерева декомпозиції

2. Якщо будь-які діти кореня вносять в композицію разом зі своїм батьком, їх слід розкласти за відношенням Decompose, а якщо діти дітей також є розкладними, ту саму історію слід повторювати над ними, поки не буде композиції між батьками та дітьми. .

3. Необхідно знайти всі обмеження якості та встановити асоційовані стосунки між дітьми та відповідними функціями.

4. Програмні цілі слід ідентифікувати та розкласти.

5. Повинні встановлюватися стосунки, що залежать і суперечать

6. Кожній з вимог слід присвоювати ранг залежно від їх важливості.

На основі того, що було обговорено, буде створено графік, як на рисунку 1.3.

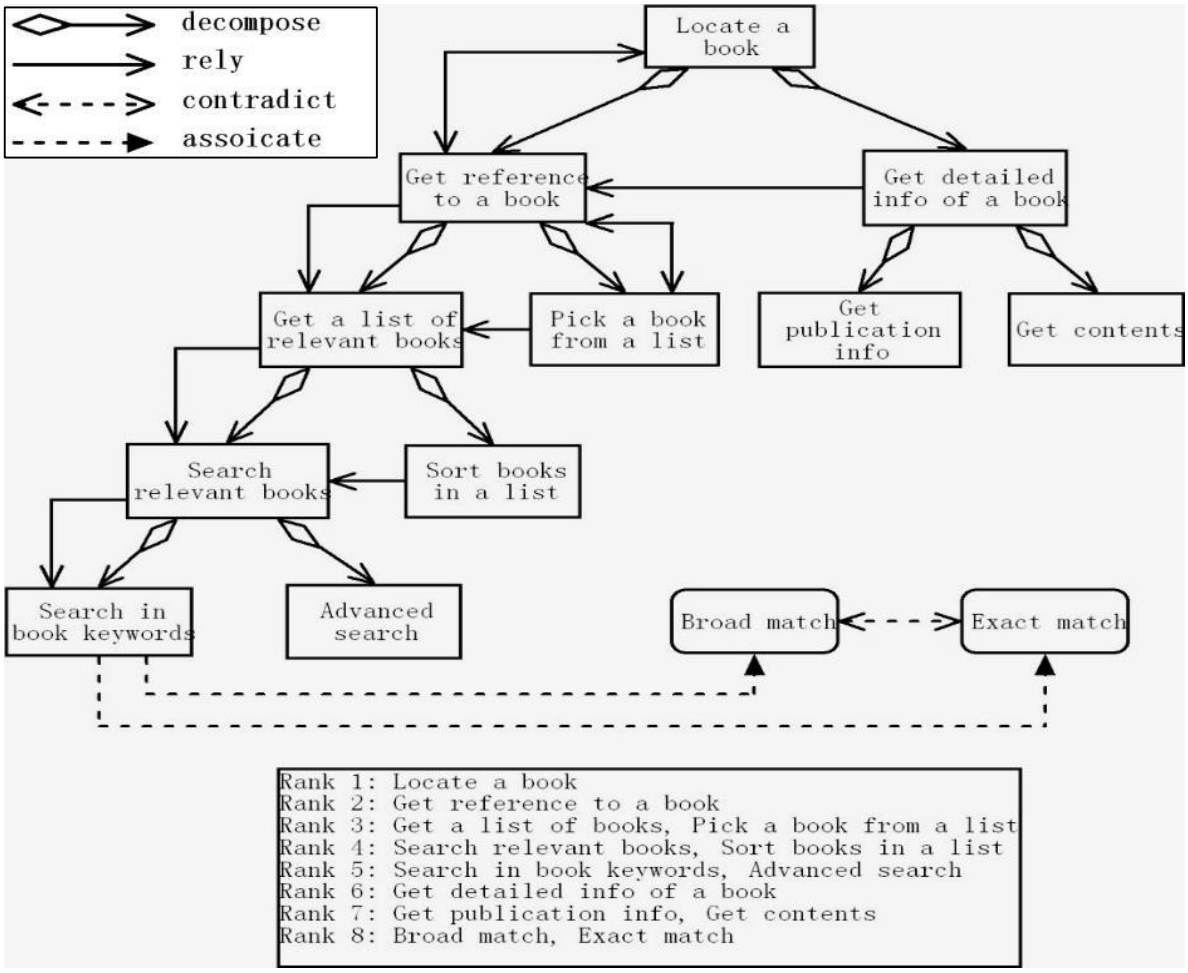


Рисунок 1.3 - Модель вимог, створена за допомогою служби пошуку книг

Вимоги будуть запропоновані користувачеві одна за одною на основі присвоєного їм рангу, і користувач повинен вибрати з них. Якщо вимога є важливою, вона буде вибрана автоматично, незалежно від думки користувача. Спочатку будуть оцінені функції, а потім – всі якості та оцінка програмних цілей.

Усім вимогам буде відповідати діалогова система. Якщо користувач вирішить відмовитися від вимоги «A», вимога «B», яка має залежний зв’язок з вимогою «A», також буде вилучена. Якщо користувач вирішив вибрати вимогу «C», а інша вимога «D» має суперечливе відношення до вимоги «C», буде вилучено, а вимога з опорним зв’язком буде вибрана [15].

2 АЛГОРИТМІЧНІ ТА ФОРМАЛЬНІ РІШЕННЯ ДЛЯ ЗАДАЧ ЗДІЙСНЕННЯ ВІЗУАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Візуалізація програмного забезпечення

Кард та ін. визначають візуалізацію як «використання підтримуваних комп'ютером інтерактивних візуальних представлень даних для посилення пізнання» [5]. Простіше кажучи, візуалізація – це графічне/візуальне відображення будь-яких знань для полегшення розуміння. Візуалізація допомагає спростити складні ідеї для кращого розуміння.

Візуалізація програмного забезпечення – це процес створення візуального образу програмної системи для кращого розуміння. Існують різні аспекти програмної системи, які можна візуалізувати, як правило, візуалізується структура програмного забезпечення, основний алгоритм, простіші компоненти програмного забезпечення або поведінка під час виконання.

Візуалізація програмного забезпечення може бути визначена як «подання комп'ютерних програм, пов'язаної документації та даних, яке покращує, спрощує та прояснює розумове уявлення інженера-програміста щодо роботи комп'ютерної системи» . Використання візуалізації програмного забезпечення допомагає легше зрозуміти програмну систему і, таким чином, зменшує зусилля та час, витрачений на різні етапи розробки програмного забезпечення та налаштування програмного забезпечення. За допомогою візуалізації розробники та зацікавлені сторони можуть отримати загальну точку зору на структуру програмного забезпечення, логіку програмного забезпечення або пояснити та спілкуватися з процесом розробки . Як правило, візуалізація програмного забезпечення в основному використовується для демонстрації поведінки програми, логічного налагодження та налагодження продуктивності, але вона в основному стосується розуміння програмного забезпечення.

Забезпечуючи гарне графічне представлення для візуалізації програмного забезпечення, краще розуміння системи користувачем може бути більш перспективним, ніж текстове представлення програмного забезпечення .

Візуалізація програмного забезпечення не повинна бути привабливою або вражаючою, але вона повинна викликати візуальне розуміння для кращого розуміння програмного забезпечення. Візуалізація програмного забезпечення полегшує людське розуміння та ефективне використання комп'ютерних програм, спираючись на ремесла типографіки, графічного дизайну, анімації, кінематографії та інтерактивної комп'ютерної графіки . Загалом графічна візуалізація може надати таку кількість інших переваг, швидшого навчання, швидшого використання та вирішення проблем, більш привабливого тощо .

Основною проблемою при реалізації програмної візуалізації широкого спектру додатків є відсутність масштабованості та гнучкості. Більшість візуалізацій, які реалізуються сьогодні, відповідають потребам конкретного проекту чи програмного забезпечення. Програмні візуалізації, реалізовані для конкретного сценарію, нелегкіможна перенести на інший сценарій.

Візуалізація програмного забезпечення має справу з представленням програмного компонента у певній візуальній формі. Для іншого застосування візуалізацію потрібно змінити відповідно до потреб нових систем. Для вирішення цієї проблеми було створено кілька узагальнених візуалізацій програмного забезпечення, щоб їх можна було використовувати в різних областях. Але у випадку цих узагальнених програмних візуалізацій, як правило, візуалізація є занадто загальною, щоб її практично реалізувати в певній області застосування. Щоб використовувати ці узагальнені візуалізації, потрібно багато змін, щоб використовувати їх у конкретній області. Зусиль, які необхідно докласти для цього, зазвичай занадто багато в порівнянні з ефективністю цих візуалізацій

Програмне забезпечення — це нематеріальний об'єкт, який неможливо побачити чи відчути фізично. Візуалізація програмного забезпечення має справу

з представленням нематеріальних програмних сутностей у вигляді чогось матеріального . У системі програмного забезпечення ми маємо великий обсяг інформації, і важко вирішити, яку інформацію має представляти візуалізація програмного забезпечення, щоб полегшити розуміння програмного забезпечення.

Візуалізація програмного забезпечення повинна бути розроблена таким чином, щоб вона могла надавати всю необхідну інформацію у такому візуальному форматі, який полегшує розуміння інформації. Рівень деталізації має бути визначений таким чином, щоб інформація, що надається за допомогою візуалізації, не була ні надто загальною, ні надто конкретною.

2.2. Етап візуальних уявлень при розробці

Програмне забезпечення — це нематеріальний об’єкт, який неможливо побачити чи відчути фізично. Візуалізація програмного забезпечення має справу з представленням нематеріальних програмних сутностей у вигляді чогось матеріального .У програмній системі міститься велика кількість інформації, і важко вирішити, яку інформацію необхідно відображати для візуалізації програмного забезпечення, щоб полегшити розуміння програмного забезпечення.

Візуалізація програмного забезпечення повинна бути розроблена таким чином, щоб вона могла надавати всю необхідну інформацію у такому візуальному форматі, який полегшує розуміння інформації. Рівень деталізації має бути визначений таким чином, щоб інформація, що надається за допомогою візуалізації, не була ні надто загальною, ні надто конкретною. Зазвичай одна візуалізація використовується для різних користувачів, але це надає інформацію, яка є занадто загальною для кількох і дуже конкретною для кількох.

Існує ряд методів візуалізації, доступних для реалізації програмної візуалізації. Різні типи візуальних елементів, таких як точки, лінії, фігури, дерева, графіки, тексти, текстури, можуть використовуватися для представлення різних

об’єктів та атрибутів. У деяких випадках для системи можна застосувати більше ніж одну техніку візуалізації .

Існують різні точки зору візуальних уявлень, на основі яких ми можемо класифікувати різні техніки візуалізації програмного забезпечення. Візуалізації можуть бути статичними або динамічними залежно від характеру використовуваної візуальної техніки. Статична візуалізація не сприймає зміни під час виконання. Прикладом статичної візуалізації є перегляд вихідного коду з кольорами . Динамічна візуалізація змінюється щоразу на основі інформації з аналізу виконання програми та даних, згенерованих під час виконання, таких як потік даних або потік керування [7].

На основі кількості вимірів найчастіше візуалізації є двовимірними або тривимірними. Двовимірні програмні інструменти візуалізації в основному включають уявлення графів або деревоподібних, які можуть містити багато вузлів і дуг. У деяких випадках може бути занадто багато інформації, яку неможливо легко представити у двох вимірах, у цих випадках потрібна додаткова просторова вимірність, що значно полегшує візуальне представлення інформації [6].

Найкраще підходящу техніку візуалізації слід вибирати, враховуючи цільових користувачів візуалізації, складність візуалізації та мету візуалізації. Під час використання певного типу візуального представлення слід враховувати рівень знань та досвіду користувачів. Використання та обмеження існуючої системи, яку планується візуалізувати, також повинні бути досліджені [6]. Необхідно впровадити техніку, яка найбільшою мірою відповідає вимогам системи.

2.3. Техніки візуалізації програмного забезпечення

Основна мета використання візуалізації в системі – зробити її більш зрозумілою та легкою у використанні для користувачів. Візуалізації допомагають користувачам мати більш чітку та точну точку зору на систему,

витрачаючи менше часу на систему. Це станеться таким чином, що замість читання коментарів і запам’ятовування структури системи користувачі динамічно бачитимуть потік системи під час роботи з текстовою системою. За допомогою візуалізацій користувачі мають огляд системи перед ними. Різні візуалізації можуть надавати різну інформацію про користувачів або вони можуть надавати ту саму інформацію різними способами. Ці різні візуалізації можна використовувати для різних типів користувачів, щоб надати їм необхідну інформацію, яка допомагає їм більш ефективно та ефективно використовувати систему, зменшуючи кількість помилок або помилок, які користувачі допускають під час використання системи.

Програмне забезпечення на основі діалогів використовується на етапі визначення вимог процесу розробки програмного забезпечення. У системі програмного забезпечення, що базується на діалогах, користувач взаємодіє з програмним забезпеченням, реагуючи на програмне забезпечення з найкращими можливими входами. Система заснована на SOA, і тому, щоб полегшити процес виявлення вимог для користувача за допомогою візуалізації, візуалізацію слід вибирати, пам’ятаючи, що вона повинна бути добре підходяща для систем на основі SOA. Сервісно-орієнтована архітектура (SOA) має багатошарову архітектуру, тому в багатьох роботах обговорюється, що відповідним підходом для візуалізації SOA є багатошаровий підхід. Однією з переваг SOA є можливість врахування різноманітних точок зору в організації оскільки в основному SOA складається як з технічних, так і з функціональних аспектів. На першому рівні можна показати потік дій, які обробляються в системі. Наступний рівень може візуалізувати послуги та зв’язки між ними. Ще більше шарів, таких як прикладний рівень, який детальніше показує реалізацію функціональних можливостей, що надаються сервісами на рівні сервісу, можна використовувати в залежності від рівня абстракції та типу користувачів.

Необхідні методи візуалізації повинні бути динамічними, щоб показати перебіг системи. Крім того, оскільки в деяких частинах системи деякі служби

мають однаковий ранг для оцінки, обрана техніка візуалізації повинна мати можливість показати паралельність і паралельність. Оскільки в цій системі використовується SOA, вона повинна забезпечувати багатошаровий дизайн для візуалізації. Для вибору кількості вимірів для системи можна вибрати як двовимірний, так і тривимірний залежно від рівня деталей, які необхідно проілюструвати. Основними об’єктами, які слід візуалізувати, є кілька завдань, таких як Оцінка, Попередня оцінка, Вибір (Так) та Відмова (Ні), які неодноразово виконуються в системі. У системі є потік, який показує порядок виконання завдань у системі. Цей потік має бути чітко представлений користувачам. Також мають бути зображені існуючі послуги, які є вимогами, які будуть висунуті.

Існує багато методів графічної візуалізації, які можуть відобразити концепцію та робочий процес інтерактивної системи виявлення вимог. Список найбільш підходящих методів, які можна використовувати для візуалізації робочого процесу програмних систем разом з їх перевагами та обмеженнями, наведено в таблиці 2.1.

Таблиця 2.1 Порівняння різних технік візуалізації

	Опис	Переваги	Недоліки
На основі мереж Петрі	Графічний інструмент для опису та аналізу одночасних процесів	Представляє функції процесу, такі як паралельність,	Модель дуже гнучка, але є
Режисерський	Показує загальний потік системи	синхронізація і	забезпечує гнучкість
Графіки	Описує модель вимог системи	конфлікти / Дозволити дуги до	втрата уваги для користувачів, які менше
На основі	Прості уявлення про вимоги	витікати з будь-якої кількості	зацікавлений у формальному аналізі

Складність є ще одним важливим фактором при виборі відповідної техніки візуалізації. Складність використовуваної візуалізації повинна бути такою, щоб візуалізація була зрозумілою, а процес розуміння візуалізації не забирав багато

часу або не вимагав від користувача занадто великих зусиль. Але складність може бути відносною у випадку взаємодії людини з комп'ютером. Різні користувачі з різним рівнем знань можуть вважати візуалізацію дуже простою або дуже складною на основі їхнього досвіду. Тому візуалізації слід вибирати таким чином, щоб вони підходили для великої групи користувачів.

2.4. Моделі типів візуалізацій

Мережа Петрі є різновидом візуального засобу комунікації, як і блок-схема або інші діаграми розробки програмного забезпечення, але головна перевага мережі Петрі полягає в тому, що її можна використовувати для аналізу та моделювання одночасної та динамічної діяльності систем» . Мережі Петрі – це дуже відомий метод формалізму для демонстрації поведінки системи робочого процесу. Мережа Петрі вперше була представлена C.A.Petri в 1962 році і з тих пір багато досліджень були зосереджені на мережах Петрі. Можливість чітко відображати в графічному вигляді проблеми, пов'язані з паралельністю, такі як паралельність, синхронізація тощо, є однією з найкращих переваг мереж Петрі .

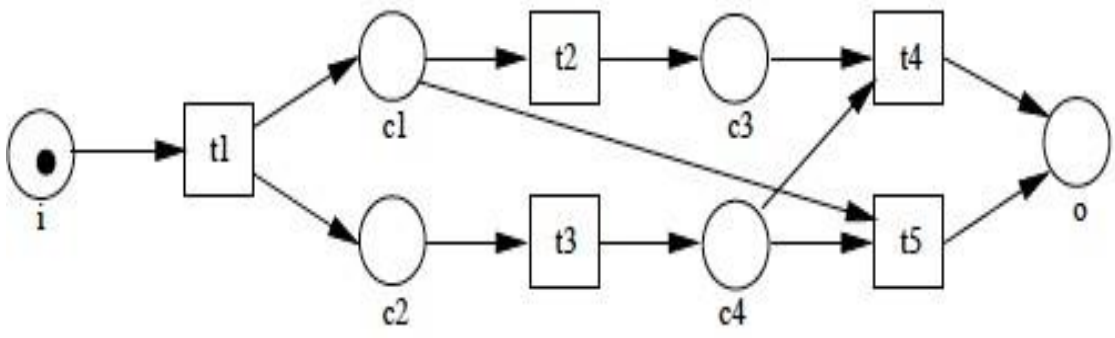


Рисунок 2.1 - Приклад мережі Петрі

Мережа Петрі має початкове маркування $M0$ і два типи вузлів, які називаються місцями та переходами, які ілюструються колами та прямокутниками відповідно. Дуга з'єднає кожне місце з переходом і кожен

перехід з місцем. Кожному місці призначається позначка, яка демонструє кількість жетонів, що існують у цьому місці. Якщо позначка місця дорівнює нулю, це означає, що місце порожнє.

Існують деякі правила, відомі як правила стрільби, які застосовуються до сітки Петрі та змінюють маркування сітки Петрі. Ці правила такі:

1. Перехід t називається увімкненим, якщо є принаймні один маркер у кожному місці введення p із t.

2. Увімкнений перехід t буде запущено, коли відбудеться пов'язана з ним подія.

3. Запуск дозволеного переходу t видаляє один маркер з кожного місця введення p з t і додає один маркер до кожного вихідного місця p з t .

Мережа Петрі - це 3-кортежний <P, T, W>, де:

• $P = \{p1, p2, p3, \dots, pm\}$ — скінченна множина місць

• $T = \{t1, t2, t3, \dots, tm\}$ — скінченна множина переходів

• $F \subseteq (P \times T) \cup (T \times P)$ — набір дуг від місця до переходу або від переходу до місця (відношення потоку) [24].

Візуалізація на основі орієнтованого графіка

Граф визначається як представлення набору об'єктів, де деякі пари цих об'єктів пов'язані один з одним. Взаємозв’язані об’єкти відомі як вершини, а зв’язки між цими вершинами називаються ребрами. Граф, у якому кожне ребро графа замінено орієнтованим ребром графа, називається орієнтованим графом. Орієнтовані графи не мають кількох ребер або петель.

Дуга e = (x,y) вважається спрямованою від x до y; y називається головою, а x — хвостом дуги; y називають прямим наступником x, а x — прямим попередником y. Якщо шлях, що складається з однієї або кількох послідовних дуг, веде від x до y, то y називають наступником x, а x — попередником y. Дугу (y,x) називають перевернутою дугою (x,y).

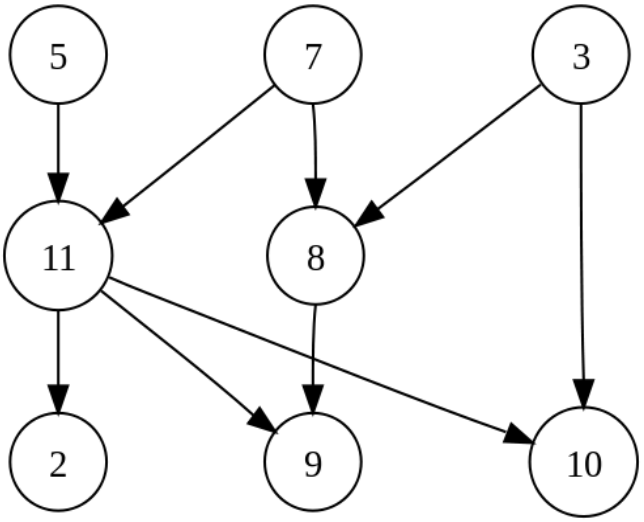


Рисунок 2.2 - Приклад орієнтованого графа

Візуалізація на основі моделі вимог

Модель вимог надає огляд і структуру процесу виявлення вимог у цій системі. Модель вимог можна використовувати для графічного пояснення процесу виявлення вимог. Таким чином, метод візуалізації на основі вимог може використовуватися для надання огляду вимог та їх взаємозв'язків між собою. Візуалізація на основі моделі вимог забезпечує ефективний метод для графічної візуалізації системи. Ця візуалізація дуже допомагає користувачам зрозуміти вимоги системи, і вони можуть легко відслідковувати всі вимоги та їх взаємозв'язки (яка вимога залежить від іншої вимоги або яка вимога є істотною або які вимоги потрібно вибирати разом) . Це дає легший огляд вимог і полегшує користувачеві знати, які вимоги були вибрані, а які вимоги відхилені.

Візуалізація на основі блоків

Прості методи візуалізації можуть надати користувачеві основну інформацію про систему, приховуючи від користувача складність системи. Візуалізацію можна вважати простою, якщо вона зрозуміла для більшості користувачів. Візуалізація в основному повинна надавати користувачеві огляд системи та поточного стану або прогресу, який був досягнутий на той момент.

Для цієї мети можна використовувати візуалізацію на основі блоків. У випадку візуалізації на основі блоків основні операції або дії представлені у вигляді простих блоків. У випадку системи виявлення вимог можна використовувати метод візуалізації на основі блоків, щоб вимоги були представлені цими простими блоками. Візуалізація на основі блоків забезпечує огляд усіх вимог і не розглядає зв'язок і зв'язки між цими вимогами. Користувачеві надається огляд того, яку вимогу було вибрано, а яку вимогу

У будь-якому життєвому циклі розробки програмного забезпечення або розробці лінійки програмного забезпечення є багато користувачів, які працюють над однією системою програмного забезпечення. Ці користувачі працюють у різних групах або командах, кожна команда має призначену роботу, а члени команди виконують подібну роботу. Ці користувачі працюють над різними аспектами програмних систем. У системі лінійки програмних продуктів на основі SOA масове налаштування програмного забезпечення полегшується за рахунок використання менших програмних артефактів як будівельних блоків для програмної системи.

У попередньому дослідженні, проведеному Відою Садрі Петрі, візуалізація на основі мережі використовується в системі, щоб допомогти користувачам взаємодіяти з системою на основі діалогу. Як зазначалося, основна група людей, які збираються скористатися перевагами цієї візуалізації, мають бути розробниками програмного забезпечення. Тим не менш, було б гарною ідеєю, щоб людям із діловим досвідом також було легко використовувати це програмне забезпечення, щоб самостійно розробляти необхідні системи. У цій роботі основна увага зосереджена на тому, щоб обмежити візуалізацію людьми з комп'ютерним досвідом, особливо розробниками програмного забезпечення. Задовольнити обидві групи з різними очікуваннями від системи – це важка робота.

Систему намагалися спроектувати так, щоб робота з нею була максимально простою навіть для людей без спеціального досвіду роботи з комп'ютерами».

Тому в системі візуалізації на основі Petri-Net основний акцент робиться на користувачах, які мають знання про програмні системи, а також певний досвід розробки програмного забезпечення. З дослідження юзабіліті, проведеного для вищезгаданого дослідження, було видно, що система не була такою корисною для людей із меншими знаннями про процес розробки програмного забезпечення.

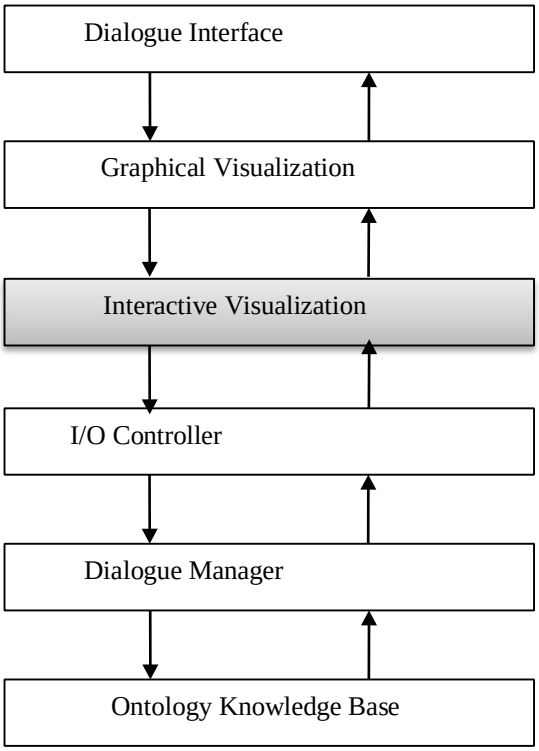
2.5. Структура запропонованого методу візуалізації

Підхід інтерактивної візуалізації має бути включено як частина раніше розробленої системи. Система на основі мережі Petri, розроблена раніше Vida

Sadri складається з п'яти компонентів: діалогового інтерфейсу, графічної візуалізації, контролера введення-виведення, менеджера діалогів і бази знань. Компонент інтерактивної візуалізації буде додано до системи між компонентами графічної візуалізації та контролера вводу-виводу.

У попередній системі відповіді передавались з компонента графічної візуалізації до контролера вводу-виводу. Але в новій системі відповідь буде передаватися через компонент інтерактивної візуалізації. Компонент інтерактивної візуалізації вирішує найкращу візуалізацію для користувача. Компонент графічної візуалізації надасть користувачеві саме цю візуалізацію. Відповідь з інтерактивної візуалізації буде передана контролеру вводу/виводу. Відповідь користувача приймається та передається до Dialogue Manager, якщо вона збігається із збереженими варіантами відповіді, якщо ні, користувача просять ввести правильну відповідь. Відповідь користувача буде перетворена у формат, який може оброблятися машиною, і буде передана менеджеру діалогів. Менеджер діалогу звернеться до бази знань онтологій і згодом згенерує відповідь. Ця відповідь буде передана контролеру вводу-виводу та компонентам візуалізації, і користувач зможе прочитати відповідь у діалоговому інтерфейсі, а також спостерігати за змінами, що відбуваються в системі, за допомогою

графічного інтерфейсу. Ці зміни відображатимуться переміщеннями маркерів і змінами кольору у візуалізаціях і фонових службах.



.

Рисунок 2.3 - Структура запропонованого методу

2.6. Модель контекстного керування

Модель контекстного контролю була розроблена Холланагелем для контролю та аналізу поведінки команди на основі когнітивних режимів. У цій моделі система вирішує, які дії слід вжити далі відповідно до контексту ситуації. Поведінка аналізується на макрорівні, а не на мікрорівні .

Модель контекстного контролю передбачає, що дії визначаються контекстом, а не будь-якими притаманними між ними відносинами. Тому модель контекстного контролю зосереджена на тому, як керується вибором наступної дії, а не на тому, чи є певні послідовності більш правильними чи ймовірними, ніж інші.

Є чотири різних режими керування. Це:

1. Скремблер

Вибір наступної дії абсолютно непередбачуваний або випадковий. Зашифрований контроль характеризує ситуацію, коли у виборі, що робити, обмірковується мало або взагалі немає.

2. Опортуністичний

Наступна дія вибирається лише з поточного змагання на основі важливих особливостей, а не більш довготривалих намірів чи цілей. Людина дуже мало планує чи передбачає, можливо, тому, що змагання не зрозуміло чітко.

3. Тактичний

Результативність базується на певному плануванні, отже, більш-менш вона відповідає відомій процедурі чи правилу. Однак планування має обмежений обсяг, і потреби, які враховуються, іноді можуть бути випадковими.

4. Стратегічний

Людина враховує глобальний контекст, таким чином використовує ширший часовий горизонт і дивиться вперед на цілі вищого рівня. Стратегічний рівень має забезпечувати більш ефективну та надійну роботу, а отже, бути ідеальним, до якого потрібно прагнути.

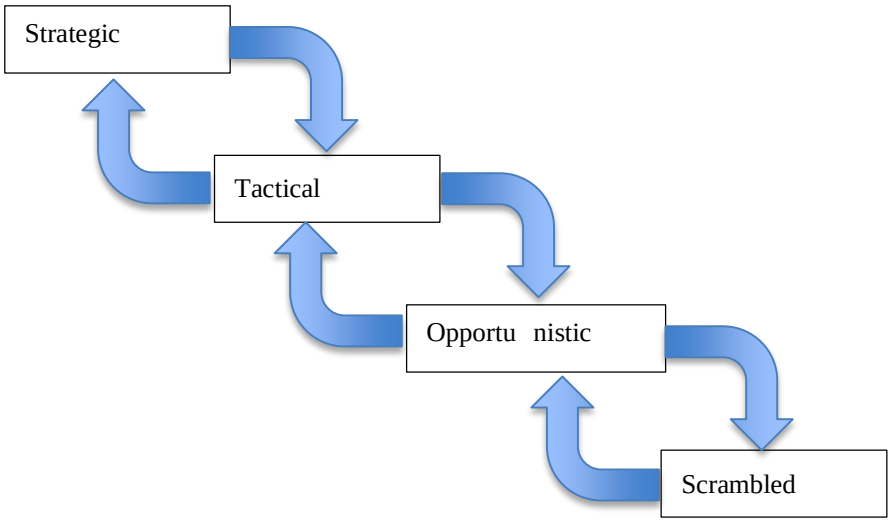


Рисунок 2.4 - Внутрішня структура моделі контекстного керування [6]

У динамічній системі індивідуальний перехід між режимами керування COCOM для підтримки контролю над динамічним станом, який, у свою чергу, залежить від поточного контексту ситуації. Він забезпечує корисну структуру для перегляду змін у когнітивній роботі у відповідь на такі контекстуальні особливості, як обмеження часу та доступність інформації. Контроль у цій моделі концептуалізується як планування того, що робити в короткостроковій перспективі та в межах часового горизонту системи, з якою взаємодіє людина.

2.7. Опис типів користувачів для візуалізації

Як обговорювалося в попередніх розділах, для програмної системи є користувачі з різним рівнем розуміння та знань. Ці користувачі мають різні сфери компетенції, і навіть у випадку користувачів з одного домену рівень або знання значно різниться. На основі чотирьох режимів контекстного контролю користувачів можна розділити на чотири категорії, враховуючи їх рівень знань та досвіду. Ці користувачі безпосередньо пов’язані з різними режимами контекстного керування, про які йшлося вище. Категоризація користувача обговорюється нижче:

1. Експерти

Користувач-експерт – це той, хто володіє найвищим рівнем знань про розробку програмного забезпечення та налаштування програмного забезпечення. Ці користувачі добре обізнані з концепцією програмного забезпечення, як-от класи, об’єкти, відносини між класами, функції та виклики функцій та інші речі, пов’язані з програмуванням. Їх більше турбують деталі речей і те, як вони працюють.

Користувачі-експерти віддають перевагу більш детальній інформації у візуалізації та можуть розуміти більш складні та технічні типи візуалізацій. Прийняття рішень у випадку цих експертних користувачів відноситься до режиму стратегічного контекстного контролю . Коли ці користувачі працюють

над налаштуванням програмної системи, вони мають довгострокові цілі, пов’язані з вибором та відмовою від необхідних послуг. Взаємодія з менеджером діалогів для цих користувачів зазвичай займає менше і більше часу. Рівень помилок у їхніх відповідях набагато менший, ніж у інших типів користувачів із меншими знаннями.

2. Професіонали

Ці користувачі – це ті люди, які мають знання про цільову систему, системні потреби та вимоги, але вони можуть мати або не мати досвіду програмування чи ні. Ці користувачі дуже чітко знають, що повинна робити система, але вони мають менше знань про те, як це потрібно робити. Планування для цих користувачів має обмежений обсяг, і потреби, які враховуються, іноді можуть бути випадковими.

Прийняття рішень у випадку цих користувачів носить тактичний характер, якщо посилатися на модель контекстного контролю . Ці користувачі будуть швидко взаємодіяти з менеджером діалогів або програмною системою, і найчастіше відповіді будуть відповідними та правильними з контекстом кінцевих цілей або кінцевих вимог.

3. Аматори

Користувачі-любителі майже не знають про налаштування програмного забезпечення чи розробку програмного забезпечення. Вони знають, чого очікувати від налаштованої програмної системи, але не мають уявлення про роботу системи. Ці користувачі знають про певні процедури чи правила, але їм бракує глибоких знань про систему.

Прийняття рішень у випадку цих користувачів відноситься до режиму опортуністичного контекстного контролю . Існує дуже мало планів для виконання необхідних завдань і дуже мало очікувань щодо майбутніх вимог. Контекст проблеми не дуже зрозумілий, і рішення приймаються після аналізу кожної вимоги в конкретний момент. Цим користувачам потрібно більше часу, щоб реагувати на певні потреби, і результати не є послідовними.

4. Новачок

Це користувачі, яким бракує технічних знань, а також бізнес-знань про програмну систему. Взаємодія людини з комп'ютером для таких користувачів є абсолютно непередбачуваною або випадковою. Про візуалізацію про технічні аспекти системи мало або зовсім немає знань. Ці типи користувачів стосуються режиму контекстного керування Scrambled, який характеризує ситуацію, коли при виборі того, що робити, потрібно мало або зовсім не думати . Цей тип користувачів робить випадкові припущення, і вони набагато більш схильні робити помилки під час процесу виявлення вимог.

2.6. Запропонований алгоритм візуалізації програмного забезпечення

Платформа візуалізації відповідає за відображення візуалізацій користувачеві. Процесор виявлення вимог обробляє відповіді та відстежує процес виявлення.

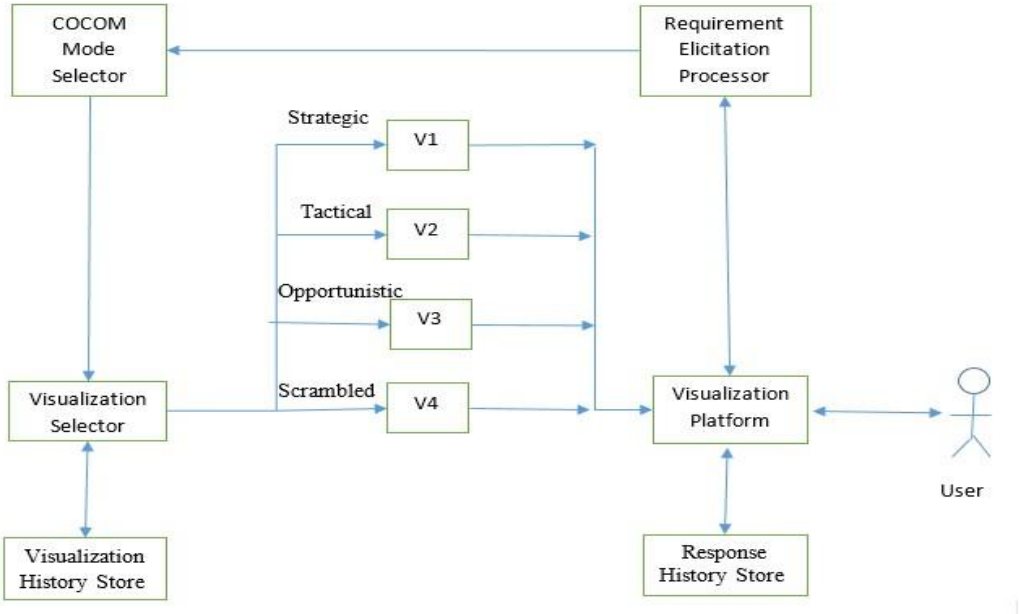


Рисунок 2.5 - Пропонована архітектура системи

Селектор режиму COCOM вибирає модель контекстного керування відповідно до історії відповідей користувача, а потім селектор візуалізації

вибирає візуалізації на основі режиму COCOM, а потім вони відображаються платформою візуалізації. Зберігання історії відповідей та історії візуалізації відстежують історію відповідей та історію візуалізації відповідно.

```
    1. ДЛЯ кожної відповіді користувача ВІДПОВІД
    2. Process_User_Response(RESP)
    3. Response_History = Get_Response_History()
    4. Visualization_History = Get_Visualization_History()
    5. COCOM_Mode = Check_COCOM_Mode(Історія_відповідей, RESP)
    6. ЯКЩО COCOM_Mode = стратегічний
    7. Updated_Pref_Visualization =
Select_Visualization(Policy_Strategic, Visualization_History)
    8. ELSE IF COCOM_Mode =тактичний
    9. Updated_Pref_Visualization =
Select_Visualization(Policy_Tactical, Visualization_History)
    10. ELSE IF COCOM_Mode = опортуністичний
    11. Updated_Pref_Visualization =
    Select_Visualization(Policy_Opportunistic,
Visualization_History)
    12. ELSE IF COCOM_Mode = зашифрований
    13. Updated_Pref_Visualization =
Select_Visualization(Policy_Scrambled, Visualization_History)
    14. IF Current_Visualization == Updated_Pref_Visualization
    15. Update_Visualization(Current_Visualization)
    16. Display_Visualization(Current_Visualization)
    17. ІНШЕ
    18. Pass_token(Current_Visualization,
Updated_Pref_Visualization)
    19. Update_Visualization(Updated_Pref_Visualization)
    20. Display_Visualiation(Updated_Pref_Visualization)
    21. Set_Current_Visualization(Updated_Pref_Visualization)
    22. Append_To_Visualization_History(Current_Visualization)
```

Рисунок 2.6 - Псевдокод системи

Process_User_Respnse викликається для обробки відповіді користувача в контексті вимоги, що розглядається. Якщо відповідь дійсна, вимогу можна оцінити, попередньо оцінити, вибрати або відкинути. Якщо відповідь недійсна, користувача просять надати дійсну відповідь. Історія відповідей користувача та історія візуалізації витягуються системою. Режим контекстного керування обчислюється на основі історії відповідей користувача та останньої відповіді. На основі цього режиму контекстного керування вибирається найбільш бажана візуалізація для користувача.

Якщо бажана візуалізація відповідає поточній візуалізації, яка відображається, візуалізація оновлюється. Якщо бажана візуалізація відрізняється від поточної візуалізації, що відображається, маркер передається з поточної візуалізації до бажаної візуалізації. Ця бажана візуалізація оновлюється відповідно до відповіді користувача, а потім відображається користувачеві. Історія візуалізації оновлюється за допомогою останньої бажаної візуалізації.

3 ПРОГРАМНА ТА АЛГОРИТМІЧНА РЕАЛІЗАЦІЯ МЕТОДІВ ВІЗУАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Основна мета цього дипломного дослідження — створити систему візуалізації, яка може задовольнити потреби різних типів користувачів на основі їх взаємодії з системою налаштування програмного забезпечення. У цьому розділі буде обговорено впровадження системи інтерактивної візуалізації в систему налаштування програмного забезпечення. Ця система була впроваджена, щоб покращити взаємодію людини з комп'ютером та зробити процес налаштування програмного забезпечення простішим і зрозумілим для користувачів із меншими знаннями про програмні системи та розробку програмного забезпечення. У цій главі також буде обговорено дослідження зручності використання, щоб перевірити, чи дійсно запропонована інтерактивна система корисна для реальних користувачів.

3.1. Реалізація системи

У цій дипломній роботі реалізація інтерактивного графічного інтерфейсу виконана за допомогою Java на операційній системі Windows. Для кодування та налагодження використовується Eclipse IDE. Симулятор графічного інтерфейсу під назвою «Rakiura JFern», який є фреймворком на основі Java, який використовується для розробки візуалізацій, що використовуються в реалізації.

Інтерактивні системи візуалізації працюють разом із системою на основі тексту. Користувач повинен працювати над налаштуванням програмного забезпечення, взаємодіючи з системою на основі діалогів і перевіряючи перебіг процесу в інтерактивному графічному інтерфейсі. Усі вимоги відображені в деяких прямокутниках на задньому плані візуалізованої системи.

Інтерактивний підхід було реалізовано в системі шляхом створення деяких нових методів та шляхом модифікації деяких існуючих методів, які використовуються для виявлення вимог, текстового діалогового інтерфейсу та

діалогового інтерфейсу на основі мережі Петрі. Клас DialogueInterface.java відповідає за основний інтерфейс системи. Більшість робіт, пов'язаних із впровадженням інтерактивної системи, виконано саме на цьому занятті. На рисунку нижче показано список методів у класі DialogueInterface.java.

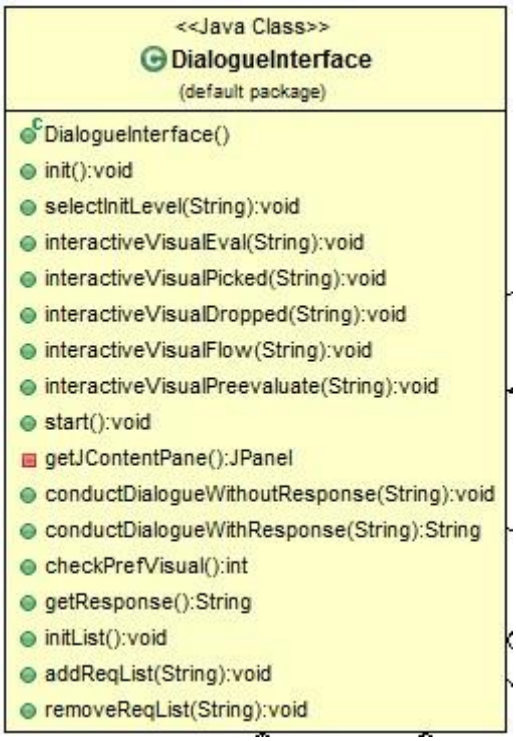


Рисунок 3.1 - Клас DialogueedInterface.java

Були створені такі нові методи:

void interactive Visual Eval (String) : Цей метод використовується для виконання візуалізації для всіх дій оцінки в процесі виявлення вимог.

void interactive Visual Picked (String) : Цей метод використовується для оновлення візуалізацій, коли користувач вибирає вимогу.

void interactive Visual Dropped (String) : цей метод оновлює всі візуалізації, коли користувач відхиляє будь-яку вимогу.

void interactive Visual Preevaluate (String) : цей метод оновлює візуалізації, коли вимога попередньо оцінюється.

void interactive Visual Flow (String) : Цей метод контролює потік керування у візуалізаціях.

void select InitLevel (String) : Цей метод використовується для ініціалізації бажаного методу візуалізації для користувача.

int check Pref Visual() : Цей метод перевіряє бажаний метод візуалізації для конкретного користувача та повертає значення на основі цього.

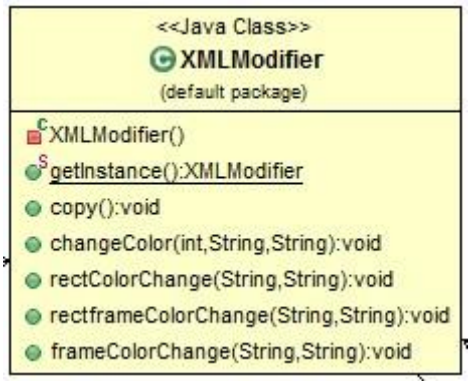


Рисунок 3.2 - Клас XMLModifier.java

У класі XMLModifier.java всі методи були оновлені, щоб включити логіку підходу до інтерактивної візуалізації. Методи були оновлені, щоб їх можна було використовувати для створення чотирьох різних візуалізацій і таким чином, що всі чотири візуалізації оновлюються одночасно, коли користувач відповідає на інтерфейс діалогу.

Існують зміни в інших класах, таких як Requirement.java,

InteractiveRE.java, BookShoppingNet.java. Усі ці класи модифіковані та оновлені, щоб пристосуватись до нового інтерактивного підходу.

В інтерактивній системі використовуються чотири різні візуалізації, але один раз лише одна з них відображається на екрані для користувача. Найкраще підходяща візуалізація для будь-якого конкретного користувача виводиться системою на основі алгоритму, розглянутого раніше в цій дипломній роботі. Інтерактивний графічний інтерфейс допомагає користувачеві знати та розуміти поточний процес налаштування програмного забезпечення. Найкраща візуалізація відображається користувачеві на основі відповідей користувача та взаємодії з системою. Графічний інтерфейс користувача надає користувачам

можливість вибрати іншу візуалізацію або переглянути іншу візуалізацію та повернутися до бажаного методу візуалізації.

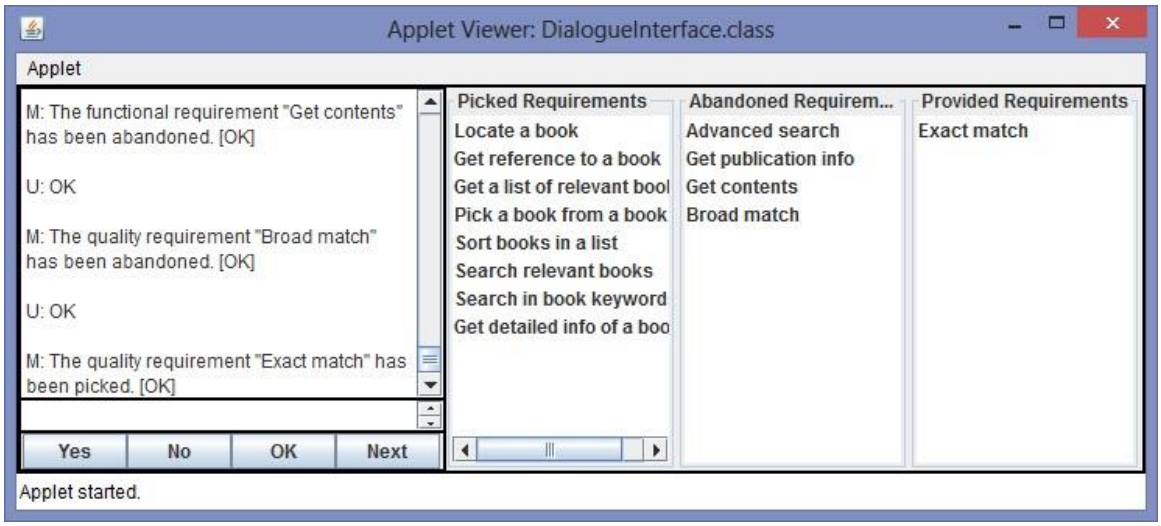


Рисунок 3.3 - Інтерфейс менеджера діалогів

Користувач взаємодіє з системою для досягнення цілей налаштування програмного забезпечення. Ця взаємодія може відбуватися кількома різними способами залежно від методів введення, які використовує користувач. Користувач може ввести свої відповіді в текстове поле, а потім може переглянути відповідь комп'ютера на самій панелі. Менеджер діалогів для процесу виявлення вимог був розроблений таким чином, що відповіді складаються з «Так», «Ні» або «ОК». З іншого боку, користувач може просто використовувати кнопки, надані в графічному інтерфейсі користувача, щоб відповісти на менеджер діалогів, відповіді системи будуть відображатися на панелі. Для цього в інтерактивній системі передбачено три різні кнопки, це кнопки «Так», «Ні» та «ОК». Є ще одна кнопка, вбудована в інтерфейс користувача

«Далі», ця кнопка використовується для зміни поточної візуалізації, яка відображається користувачеві. В інтерактивній системі ми маємо чотири різні візуалізації, іноді користувач може захотіти перевірити якусь іншу візуалізацію, і це можна зробити, натиснувши кнопку «Далі». Інтерактивна структура,

запропонована в цій дипломній роботі, розділяє користувачів на чотири групи на основі моделі контекстного контролю. Відповіді користувачів вивчаються, а потім відображається найкраща можлива візуалізація для конкретного користувача. Для цього було реалізовано чотири різні візуалізації. Це:

1. Візуалізація на основі мереж Петрі
2. Візуалізація на основі орієнтованого графіка
3. Візуалізація на основі моделі вимог
4. Візуалізація на основі блоків

На рисунку 3.4 показано візуалізацію на основі мережі Петрі. . Як показано, всі вимоги розміщені в деяких прямокутниках на тлі візуалізованої системи, а мережа Петрі знаходиться зверху. Як зазначалося раніше, оскільки ця візуалізація відноситься до категорії двовимірної візуалізації, то здається, що ці два шари накладаються.

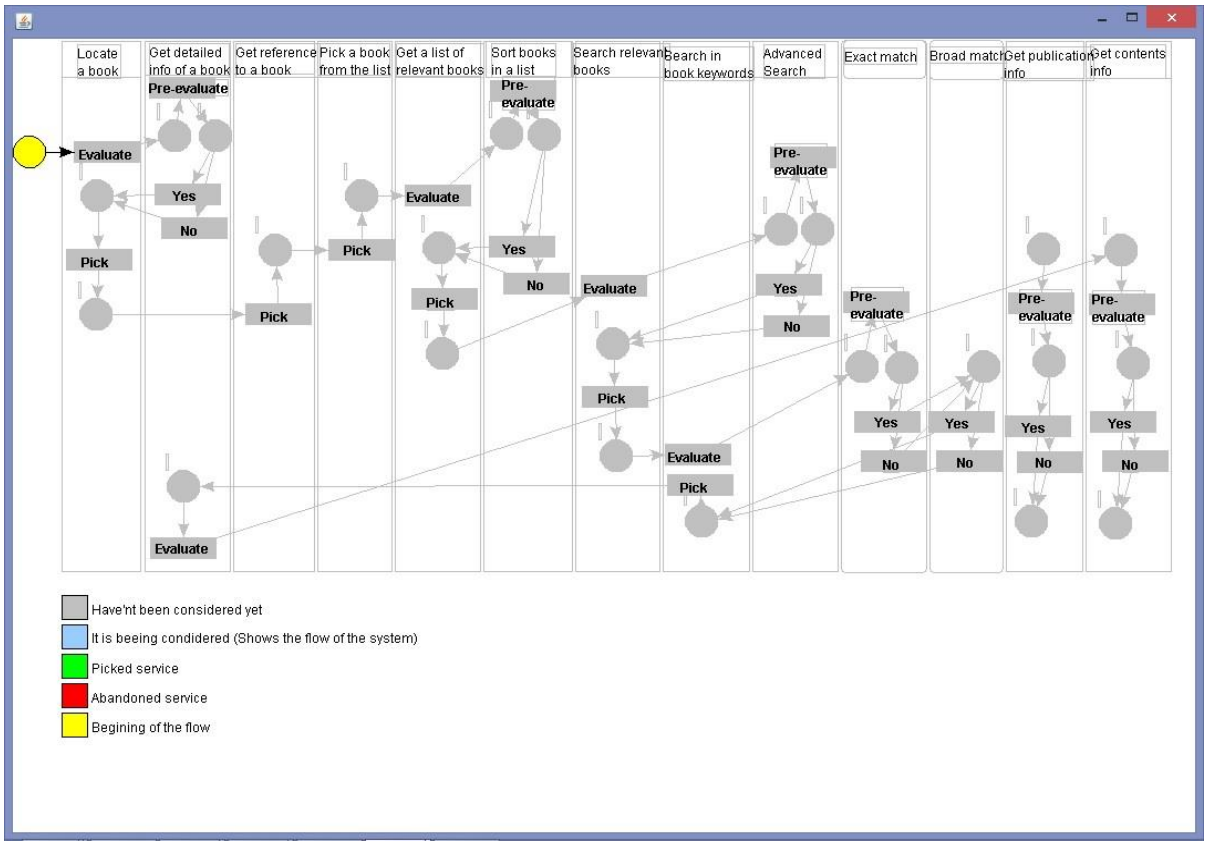


Рисунок 3.4 - Візуалізація на основі мережі Петрі

Основні дії, що виконуються в онтології текстової системи, - це оцінка, попередня оцінка, вибір і вибір або відмова від послуг. Дії представлені переходами. Кожного разу, коли виконується кожна з цих дій, перехід, пов'язаний із цим завданням, запускатиметься, і колір цього переходу, його вхідної дуги та місця та вихідної дуги стане синім. Таким чином, потік системи буде представлений зміною кольору. Кожна вибрана вимога в системі стане зеленою, а кожна залишена стане червоною, як тільки система на основі діалогу оголосить, що відповідно вона вибрала або відмовилася від цієї послуги.

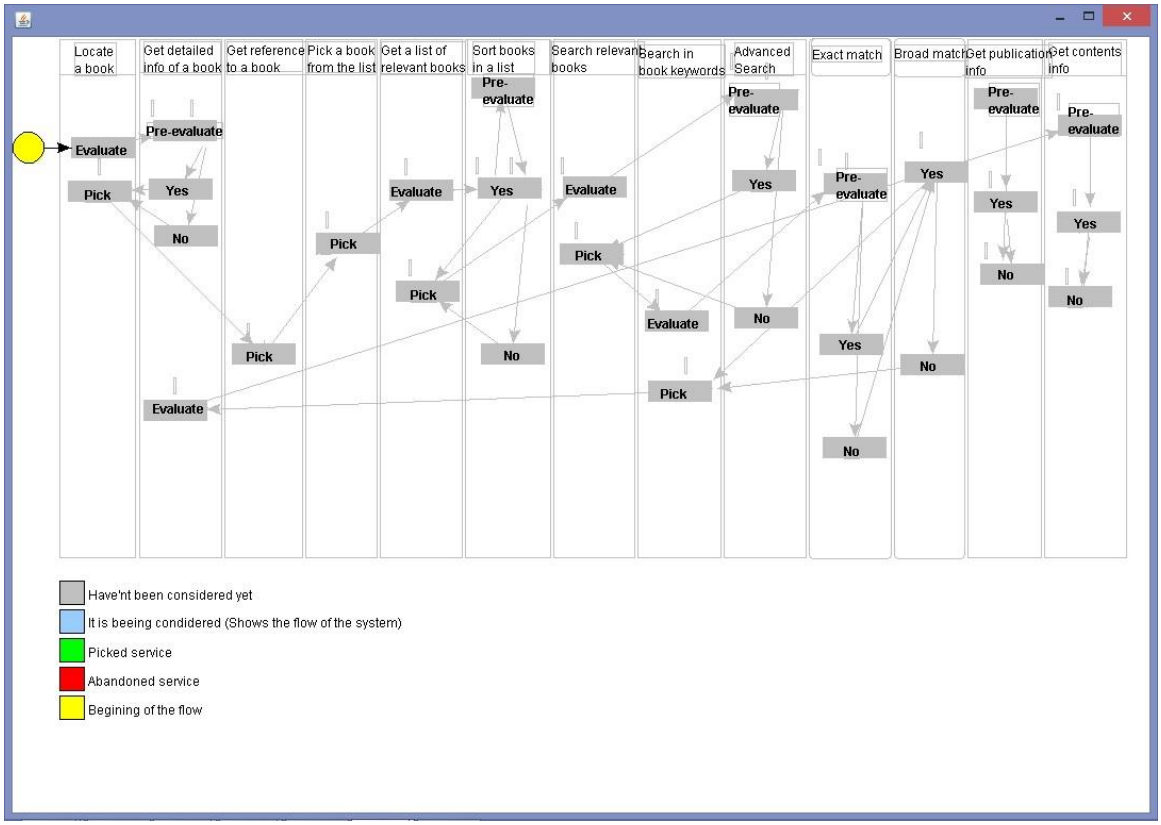


Рисунок 3.5 - Пряма візуалізація на основі графіка

На рисунку 3.5 показана візуалізація на основі орієнтованого графіка. У випадку візуалізації на основі орієнтованого графіка всі вимоги відображаються у вигляді прямокутників на задньому плані візуалізованої системи, а орієнтований графік знаходиться над ним. Через двовимірну природу візуалізації здається, що два шари накладаються.

Дії зображуються вершинами орієнтованого графа. Кожного разу, коли виконується кожна з цих дій, перехід, пов’язаний з цим завданням, буде запущено, і колір цього переходу стане синім, а потік системи буде представлений зміною кольору. Вимоги візуалізуються точно так само, як і у візуалізації на основі мережі Петрі, вибрані вимоги позначені зеленим, а випущені — червоним кольором.

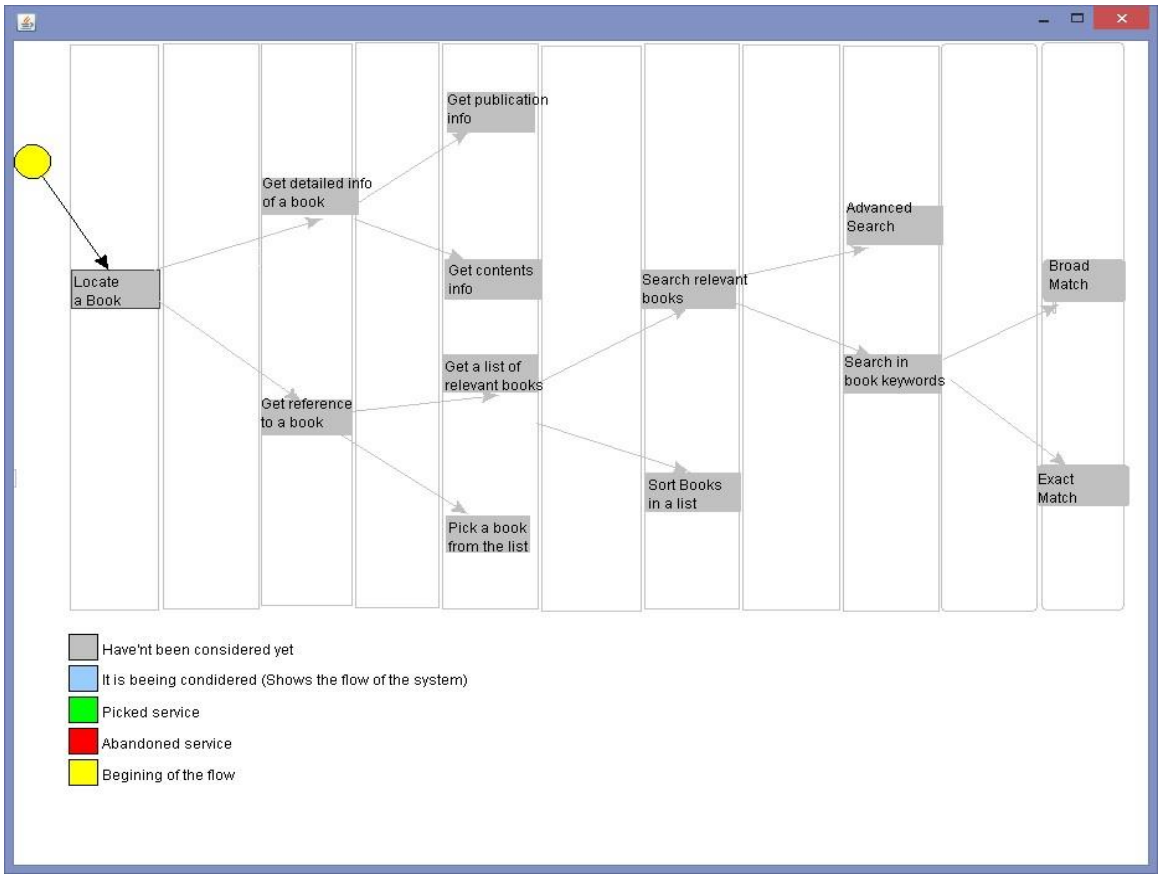


Рисунок 3.6 - Візуалізація на основі моделі вимог

Візуалізація на основі моделі вимог не візуалізує дії, які виконуються на кожному кроці. Це зменшує складність візуалізації та допомагає користувачам зосередитися на вимогах. Модель вимог зображує вимоги та їх взаємозв’язки між собою. Коли вимога вибирається, блок, що представляє цю конкретну вимогу, стає зеленим, тоді як коли вимога відкидається, блок стає червоним.

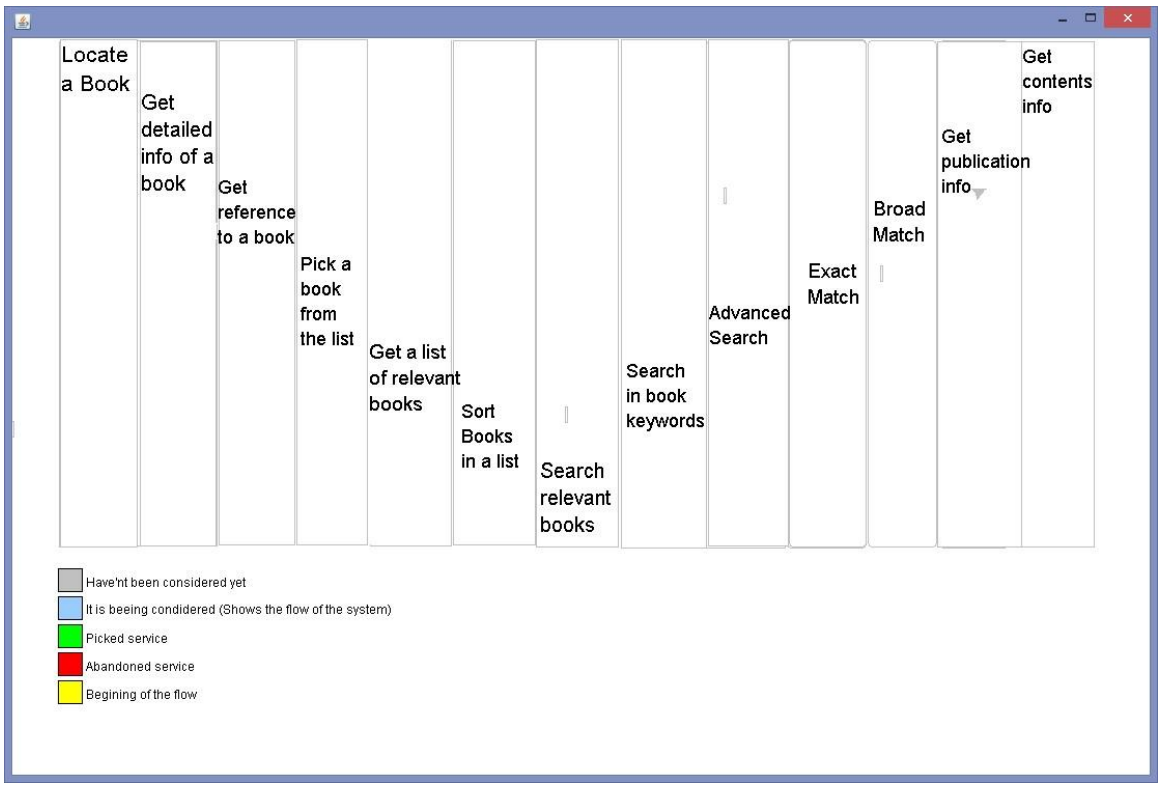


Рисунок 3.7. Візуалізація на основі блоків

Візуалізація на основі блоків є найпростішою серед усіх реалізованих візуалізацій, і вона лише візуалізує вимоги процесу виявлення вимог. У разі такої візуалізації навіть зв'язки між вимогами не враховуються. Блок, що представляє конкретну вимогу, стає зеленим, коли вимога обрана, або червоним, коли вимога відкидається.

3.2. Дослідження процесу юзабіліті

Юзабіліті можна визначити як легкість, з якою користувач може навчитися працювати, готувати вхідні дані, інтерпретувати вихідні дані системи або компонента [4]. Зручність інтерфейсу — це міра ефективності, ефективності та задоволеності, з якою певні користувачі можуть досягти визначених цілей у певному середовищі за допомогою цього інтерфейсу [7].

Зручність використання стосується того, наскільки легко використовувати комп’ютерні системи. Юзабіліті часто відрізняють від корисності, яка більше

стосується функціональності. Як зрозуміло з наведених вище визначень, зручність використання охоплює багато аспектів використання системи. Наприклад, ми повинні враховувати легкість навчання, простоту регулярного використання, запам’ятовування, обробку помилок і навіть суб’єктивне задоволення. Ми також повинні враховувати типи користувачів, які, ймовірно, будуть використовувати систему, і яких цілей вони намагатимуться досягти. Після того, як ми визначимо аспекти зручності використання та користувачів, які найбільше цікавлять, ми можемо відповідно провести дослідження юзабіліті [7].

Дослідження юзабіліті проводиться для вивчення зручності використання системи. Зручність - це якісна характеристика системи, яка стосується взаємодії системи з людьми. Це допомагає дізнатися практичну ефективність системи з точки зору зручності використання. Юзабіліті є дуже абстрактним поняттям, і дослідження юзабіліті допомагає у вивченні абстрактної концепції юзабіліті за допомогою деяких атрибутів юзабіліті, які є точними та вимірюваними за своєю природою. Нижче наведено атрибути юзабіліті, використані в дослідженні юзабіліті:

• Можливість навчання – наскільки швидко і легко користувачі можуть виконувати продуктивну роботу з новою системою і наскільки легко вони можуть запам'ятати, як система працює, після того, як деякий час не користувалися системою.

• Ефективність – кількість завдань, які користувач може виконати за певний проміжок часу.

• Надійність – рівень помилок у системі та час, необхідний для відновлення після помилок.

• Satisfaction – рівень задоволеності користувача після роботи з системою. [6]

Ці атрибути можна виміряти, спостерігаючи за користувачами, коли вони працюють у системі. З учасниками також можна провести співбесіду, щоб отримати від них зворотній зв’язок після того, як вони виконали завдання за

допомогою певної системи. Інтерв'ю також дає нам спосіб отримати зворотній зв'язок від користувача, що він відчуває про систему та їхню реакцію після використання нової системи.

У питаннях взаємодії людини з технологією парадигми аналітичного дослідження недостатньо. Тому емпіричні дослідження в галузі програмної інженерії постійно стають все більш прийнятними [6]. Юзабіліті – це те, як система взаємодіє з користувачем [7]. Інженерія юзабіліті визначає кінцевий рівень юзабіліті та гарантує, що програмне забезпечення, яке тестується, досягне цього рівня [7].

Атрибути зручності можна виміряти, спостерігаючи за користувачами, коли вони працюють із системою, або проводячи інтерв'ю та опитувальники після того, як вони скористалися системою. Інтерв'ю та анкети повинні містити запитання, пов'язані з взаємодією між користувачем і машиною. У [7] Гулд і Льюїс запропонували «Відомий

Правила» для інженерії юзабіліті. Ці відомі правила:

• Раннє зосередження на користувачах

• Участь користувача в проектуванні

• Координація різних частин інтерфейсу

• Емпіричне тестування користувачами та повторний перегляд проектів

У [6] запропоновано дев'ять евристик: простий і природний діалог, розмовляти мовою користувача, мінімізувати навантаження на пам'ять користувача, бути послідовним; забезпечити зворотній звязок; забезпечити чітко позначені виходи; забезпечити короткі шляхи; гарні повідомлення про помилки та запобігання помилкам.

Існує ще один метод когнітивних покрокових інструкцій, який використовує більш чітку, детальну процедуру та проводить аналіз юзабіліті, заснований на роботі, тестуючи реальних користувачів, коли вони стикаються з системою. Для того, щоб проаналізувати якість інтерфейсу в спрямуванні

користувача на виконання конкретного завдання, задаються три прості запитання:

• Чи будуть правильні дії достатньо очевидні для користувачів?

• Чи пов’язують користувачі опис правильної дії з тим, що вони намагаються виконати

досягти?

• Чи правильно інтерпретують користувачі відповідь системи на вибрану дію?

Відповідь на всі ці запитання має бути «так», якщо на будь-яке з цих питань є відповідь «ні», можуть виникнути проблеми [6]. Життєвий цикл інженерії юзабіліті складається з трьох етапів:

• Етап попереднього проектування

На етапі передпроектування життєвого циклу інженерії юзабіліті акцент робиться на цільовому користувачеві та завданнях, які буде виконувати кінцевий користувач. На цьому етапі слід зосередитися на користувачеві, зрозуміти природу та потреби користувачів. Існує кілька різних критеріїв, які можуть надати нам корисну інформацію про користувача. Наприклад, досвід користувача з подібними системами може бути дуже важливим фактором у більшості випадків. На цьому етапі повинні бути встановлені цілі щодо зручності використання.

• Етап проектування

На етапі проектування акцент робиться на правильній реалізації. Випущена система повинна бути зручною та корисною для користувача. Зазвичай прототип розробляється на основі принципів зручності використання та потреб користувачів. Цей прототип тестується з реальними користувачами, і відгуки від цих користувачів використовуються для аналізу, чи відповідатиме дизайн поставленим цілям.

• Етап постпроектування

Етап постдизайну – це вивчення продукту, який буде використовуватися в цій галузі. Тестування проводиться на реальній системі з реальними користувачами. На цьому етапі дизайн можна переглянути та повторно протестувати для майбутніх версій продукту.

3.3. Пропонований метод тестування юзабіліті

У цьому дослідженні з метою тестування юзабіліті системи було проведено як аналітичне, так і емпіричне тестування. На різних етапах розробки системи використовувалися методи перевірки юзабіліті та когнітивного проходження. Відомі правила використовувалися всюди, де вони були застосовні та здійсненні в системі. Основною метою було покращити зручність використання системи шляхом підвищення зручності використання атрибути.

Для покращення зручності використання системи реалізовано підхід інтерактивної візуалізації програмного забезпечення. Очікується, що покращить зручність використання системи шляхом введення різних типів візуалізацій для різних типів користувачів на основі знань і розуміння цих користувачів. Існуючу систему було вдосконалено та вдосконалено. Користувач може змінити візуалізацію, яка йому / їй. Система намагається перевірити рівень та уподобання користувача та надати йому найкращу візуалізацію на основі відповідей користувача. Щоб перевірити, чи дає запропонований підхід очікуваний рівень результатів, було проведено дослідження юзабіліті. Візуалізований інтерфейс забезпечує зворотний зв’язок для користувача, змінюючи колір вузлів, і інформує користувача про те, що відбувається в системі. Різні візуалізації були розроблені таким чином, щоб вони узгоджувалися із загальним дизайном системи. Повідомлення про помилки та інструкції для користувачів були змінені та більш зрозумілі в запропонованій системі.

Для емпіричного тестування зручності використання системи як модифіковану, так і оригінальну систему перевірять користувачі з різним рівнем знань з інформатики та інших факультетів. Для розуміння взаємодії користувача

з системою буде підготовлено анкету. Він міститиме запитання щодо порівняння між системами. Анкета міститиме запитання для надання нових ідей та пропозицій щодо покращення зручності використання системи.

Щоб перевірити зручність використання системи, як існуючу, так і запропоновану систему оцінили користувачі з різним рівнем досвіду розробки програмного забезпечення. Передбачалося, що загалом студенти інформатики мають вищий досвід розробки програмного забезпечення, ніж студенти інших факультетів. Крім того, рівень досвіду та знань щодо розробки програмного забезпечення серед інформатики різниться залежно від рівня навчання. Вважається, що аспіранти факультету інформатики мають найвищий рівень знань.

Таблиця 3.1 – Розподіл учасників за їхнім академічним рівнем

	студенти		студенти	
	Existing	Proposed	Existing	Proposed
Бакалавр (1 курс)	1	0	4	1
Бакалавр (2 курс)	1	2	2	3
Бакалавр (3 курс)	7	5	3	4
Бакалавр (4-й курс)	2	6	2	1
Аспіранти	7	5	10	12

3.4. Опис завдання

Під час роботи з системою учасники повинні були виконати декілька кроків та інструкцій. Мета дослідження пояснюється користувачеві. Також загальна функціональність комп'ютерного програмного забезпечення для онлайн-покупок книг і концепція вимог до програмного забезпечення та процес виявлення вимог. Класичний процес розробки програмного забезпечення (SDP)

і лінійку програмних продуктів (SPL) пояснювали за допомогою дуже відомої та простої концепції Lego.

Учасників попросили порівняти, як місто можна побудувати з базових кубічних деталей Lego, щоб зробити місто за допомогою готових аксесуарів Lego, таких як двері, вікна, персонажі та транспортні засоби. У другому способі, замість того, щоб створювати кожну одиницю міста шляхом з'єднання основних будівельних блоків один за одним, місто можна створити за допомогою попередньо виготовлених частин. Цю концепцію можна узагальнити до концепції класичного процесу розробки програмного забезпечення проти лінійки програмних продуктів. Користувачам пояснили, що SDP – це як будувати місто з базових деталей Lego, тому що в цьому процесі код потрібно писати з нуля. З іншого боку, SPL може бути таким же, як процес побудови міста зі складанням готових аксесуарів разом. Оскільки SDP базується на налаштуванні та повторному використанні існуючих програмних компонентів [1], цей приклад може бути ілюстративним для учасника з будь-яким рівнем знань про комп'ютери та, зокрема, розробку програмного забезпечення.

3.5. Завдання для верифікації

Після пояснення кількох основних концепцій, пов'язаних із системою, користувачам надали лист із завданнями. У цьому аркуші завдань описано завдання, яке їм було запропоновано виконати. Було два різних аркуші завдань, по одному для існуючої та пропонованої системи. Функціонування систем було пояснено в цих листках із завданнями.

Завданням для учасників було вибрати лише три вимоги із 7 необов'язкових вимог, які їм пропонує система. В обох системах вимоги «Отримати детальну інформацію про книгу», «Сортувати книги у списку», «Розширений пошук», «Точну відповідність»,

«Широка відповідність», «Отримати інформацію про публікацію» та «Отримати вміст» є необов'язковими вимогами.

Будь-який користувач може вибрати, чи не включати ці вимоги в систему. У аркуші завдань учасникам було запропоновано вибрати лише три сервіси, які надають сервісу онлайн-покупки книг наступні можливості:

1. Щоб відсортувати результати пошуку

2. Щоб шукати книгу на основі точного слова, яке введено в систему.

3. Показати користувачеві інформацію про вміст книги, яку він шукав.

Ці три описи відповідають трьом вимогам «Сортувати книги у списку», «Точна відповідність» і «Отримати вміст книги» відповідно.

Отже, завдання вимагає від учасників вибрати лише ці три вимоги та відмовитися від інших чотирьох необов'язкових вимог. Враховуючи всі вимоги, які учасник може вибрати, щоб включити в систему, було вирішено оцінити роботу кожного учасника з 7. Таким чином, за кожен правильний вибір призначався один бал, а за відмову від запитаної вимоги – один бал. Розбіжність між максимальною та мінімальною оцінкою складе 7. Якщо учасник вибере лише 3 необхідні вимоги, 3 бали будуть додані до базової оцінки 0, а ще 4 бали будуть додані за відмову від усіх вимог, які не потрібні. бути обраним. У цьому випадку учасник набере максимум 7 балів. З іншого боку, якщо учасник вибере всі непотрібні вимоги і відмовиться від усіх обов'язкових, оцінка буде 0.

Також зафіксовано час, який учасники зайняли. Час початку – це час, коли учасник фактично почав використовувати систему, і час завершення, коли учасник завершив вибір усіх вимог. Тривалість часу, який учасник витратив на виконання завдань із системою, розраховували для того, щоб дослідити, в середньому, скільки часу потрібно учасникам для виконання завдання. Цей час використовується для розрахунку ефективності системи.

Таким чином, ефективність як існуючої, так і запропонованої систем може бути розрахована за результатами часу, знайденими під час дослідження. Подібним чином бали можна використовувати для розрахунку частоти помилок системи. Учасникам було запропоновано заповнити анкету, відповіді на

запитання в анкеті були використані для розрахунку інших важливих факторів, таких як здатність до навчання, ефективність та задоволеність користувачів.

3.6. Розробка анкети учасника

Метою анкети було отримати відгуки від учасників, щоб ми могли проаналізувати зручність використання системи. Питання анкети були розроблені таким чином, щоб вони надавали інформацію про ефективність, рівень помилок, задоволеність та проблеми, пов'язані з системою. Всього в анкеті було 7 запитань. Результати, отримані від думки учасників про систему, використовуються для аналізу зручності використання.

Можливість навчання

Можливість навчання – це те, наскільки швидко та легко нові користувачі можуть вперше працювати з новою системою. Придатність для навчання системи аналізується на основі думок користувачів про простоту використання нової системи та розуміння системи користувачем. Перші два питання в

анкети використовуються для перевірки придатності системи до навчання. Ці питання були:

1. Від 1 до 10, наскільки легко вам було працювати з системою та налаштовувати сервіс онлайн-покупки книг? (1 найважчий, 10 найлегший)

2. Від 1 до 10, скільки балів ви збираєтеся поставити за своє загальне розуміння системи? (1 мінімум, 10 максимум)

Легкість

Перше запитання в анкеті пропонувало учасникам оцінити рівень зручності роботи з системою. Учасники визначають рейтинг на основі свого досвіду роботи з системою. Їх попросили оцінити його за шкалою 10, 10 — найлегший, а 1 — найважчий.

На рисунку 3.1 середні бали легкості та середні бали були розподілені за категоріями для студентів інформатики та студентів з груп кафедр. Середнє

значення як балів, так і рангу легкості є вищими у випадку запропонованої системи.

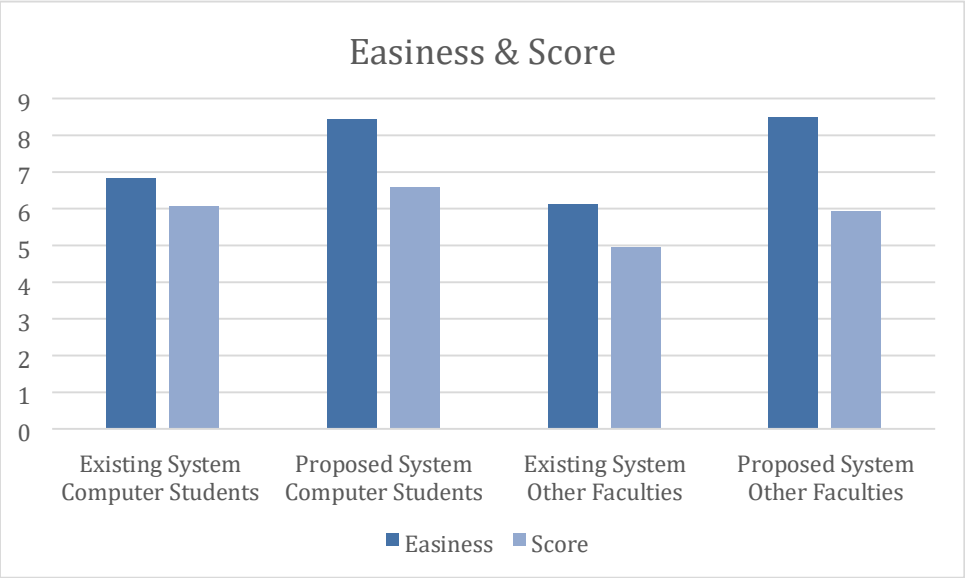


Рисунок 3.8 - Порівняння легкості та оцінки

Друге запитання в анкеті стосувалося зрозумілості системи.

Учасників запитали щодо їхнього розуміння системи. Як і в першому питанні, учасник повинен був оцінити за шкалою 10, 10 – найвища, а 1 – найнижча.

Оскільки зворотній зв’язок на запитання говорить нам лише про думку учасника про його розуміння системи, він не дає повної картини. Щоб зробити цей ранг більш значущим, нам потрібно змінити його залежно від того, наскільки успішно вони виконали завдання та як швидко вони виконали завдання. Тому пропонується наступна формула, яка задовольняє ці потреби.

Розуміння буде кращим, якщо учасник приймає правильні рішення, працюючи над системою, і досягає цілей за менший час. «Оцінка» показує, наскільки успішним був учасник у досягненні необхідних цілей. Аналогічно, «Час» — це час, затрачений учаснику на виконання завдання.

На рисунку 3.9 показано порівняння між значеннями зрозумілості та простоти для обох систем. Значення зрозумілості в інтерактивній системі більше,

ніж в існуючій системі для обох груп. Оскільки і легкість, і зрозумілість запропонованої системи вища, ніж існуюча, можна зробити логічний висновок, що засвоюваність запропонованої системи вища.

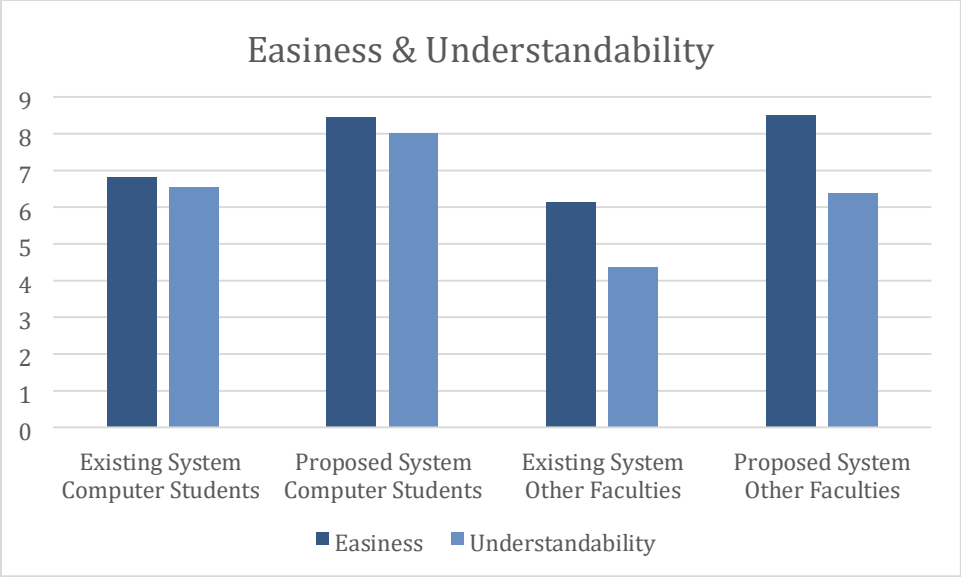


Рисунок 3.9 - Порівняння простоти та зрозумілості

Наступним фактором у дослідженні юзабіліті є ефективність. Ефективність можна визначити як кількість завдань, успішно виконаних користувачем за певний проміжок часу. У нашому дослідженні кожному учаснику було запропоновано виконати однакові завдання для досягнення тих самих цілей.

Кількість успішно виконаних завдань у цьому випадку дорівнює балу учасника. Отже, ефективність розраховується на основі балів учасника та часу, який учасник витрачений на виконання потрібних завдань.

На рисунку 3.10 наведено порівняння між оцінками ефективності для всіх користувачів існуючої системи, а також інтерактивної системи. Ефективність користувачів, які використовували інтерактивний метод незалежно від рівня їх знань, була вищою, ніж користувачів, які використовували систему візуалізації на основі Petri Net. Отже, можна зробити висновок, що запропонована система допомагає користувачам стати ефективнішими, точнішими та швидшими.

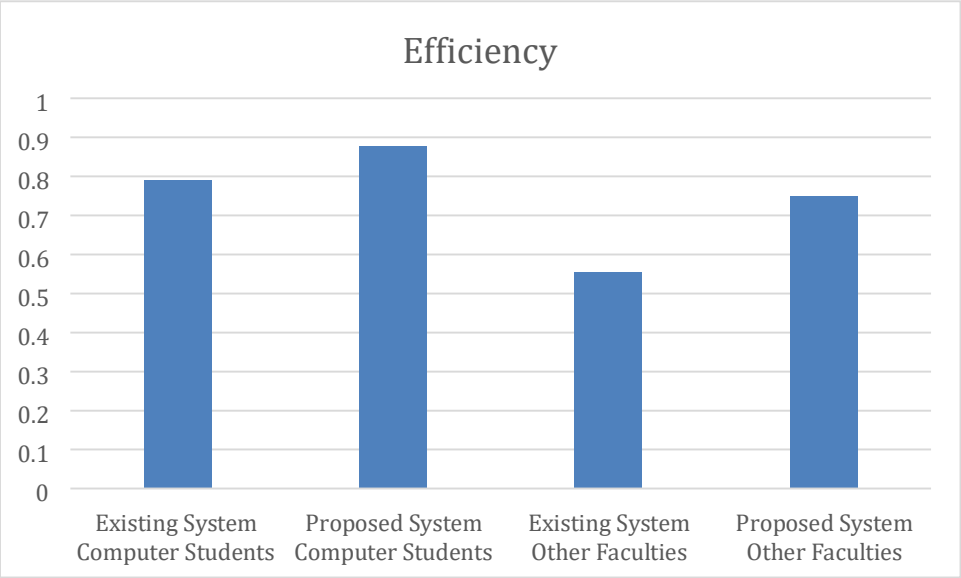


Рисунок 3.10 – Порівняння ефективності між різними групами

Наступним фактором зручності використання, який слід перевірити, є рівень помилок. Для оцінки коефіцієнта помилок кожної системи слід враховувати оцінку кожного учасника. Оскільки оцінка була розрахована з семи, помилка, яку робить кожен користувач, є доповненням до отриманого бала. Коефіцієнт помилок – це відсоток помилок, які допускає користувач під час роботи з системою.

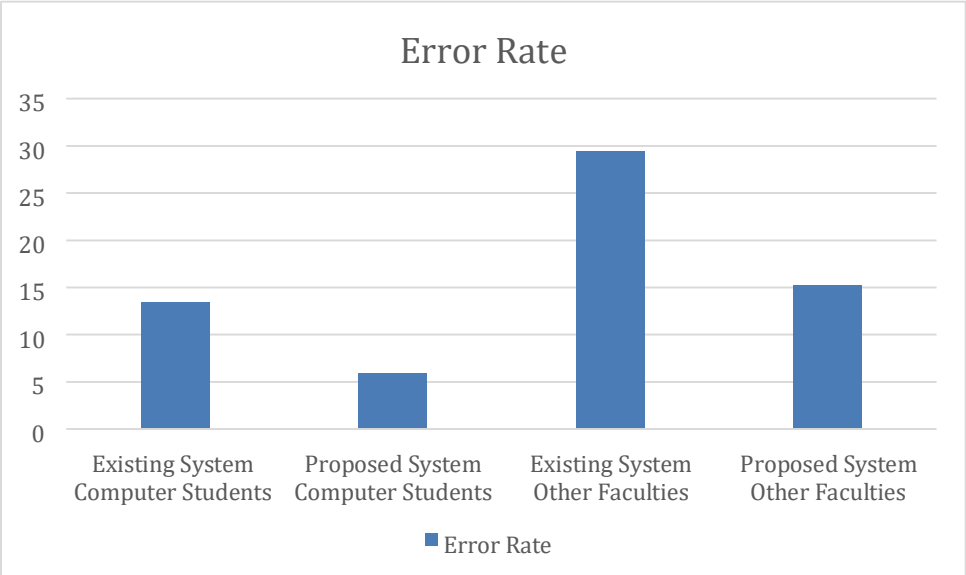


Рисунок 3.11 - Порівняння частоти помилок

Середній рівень помилок для студентів комп'ютера, які використовують існуючу систему, становить 13,44%, а для студентів-комп'ютерників, які використовують запропоновану систему, він знижується до 5,88%. Так само середній рівень помилок для студентів інших факультетів становить 29,46% при використанні існуючої системи. Тоді як при використанні інтерактивної системи середній рівень помилок для студентів інших факультетів знижується до 15,18%.

Ще одним важливим фактором у дослідженні юзабіліті є задоволення користувачів. Як випливає з назви, це говорить нам про те, наскільки учасник був задоволений після роботи з системою. Третє запитання в анкеті стосується задоволеності користувачів. Учаснику необхідно оцінити рівень задоволеності системою за 10-бальною шкалою, де 10 – повністю задоволений, а 1 – зовсім не задоволений.

Середній бал, який студенти інформатики дають існуючій системі, становить 7,22, для інтерактивної системи цей показник збільшується до 8,11. Аналогічно середній бал задоволеності наявною системою візуалізації на основі мережі Петрі для студентів інших факультетів становить 7,43, і це збільшується до 8,37 з 10 у випадку інтерактивної системи візуалізації. Коли ми порівнюємо рівні задоволеності обох систем, задоволеність вища для запропонованого інтерактивного методу.

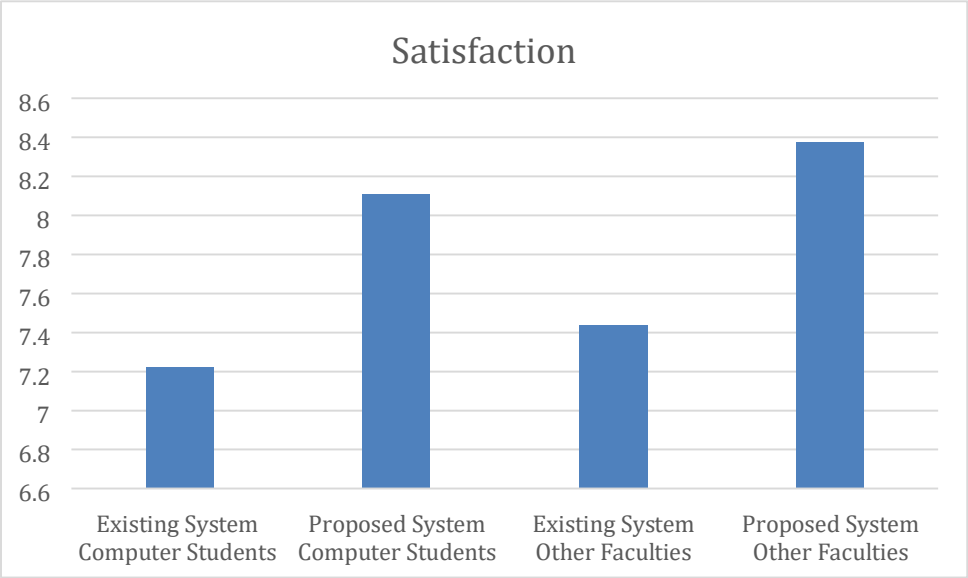


Рисунок 3.12 Порівняння задоволеності

Було одне конкретне запитання, яке просто поставили користувачам, які користувалися інтерактивною системою. Це питання стосувалося думки користувачів про необхідність інтерактивної візуалізації. Питання було:

Як ви вважаєте, чи була потрібна для системи інтерактивна візуалізація?

а. Необхідно б. Без різниці с. Не Необхідно

Варіанти відповідей на це запитання були «потрібними» для користувачів, які вважали, що це було корисно, «без різниці» для студентів, які не дивляться на графічний інтерфейс і вважають за краще читати коментарі інтерфейсу діалогу та «не потрібно» для студентів, які думають, що це може бути заплутаним і відволікаючим.

83% студентів-комп'ютерників, 81% студентів інших факультетів і в цілому 82% всіх студентів вважають, що інтерактивний підхід візуалізації необхідний для системи. 11% студентів-комп'ютерників і 14,2% студентів інших факультетів і загалом 12,8% всіх студентів виявили, що інтерактивний підхід візуалізації недостатньо корисний і для них немає ніякої різниці для його існування.

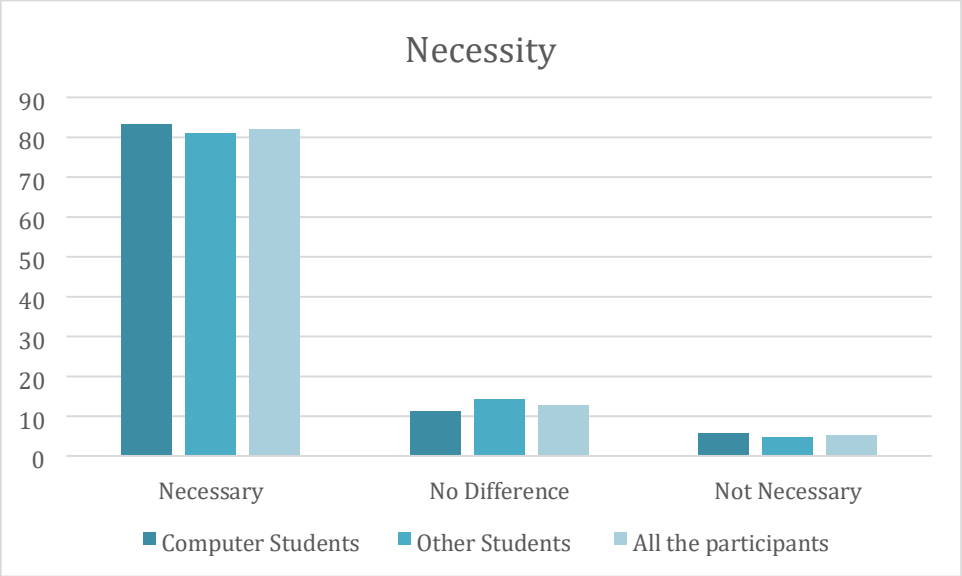


Рисунок 3.13 - Порівняння необхідності

Відсоток студентів-комп’ютерників, які вважали інтерактивний підхід візуалізації непотрібним і, ймовірно, більш заплутаним для роботи з системою, становив 5,5%, а для студентів інших факультетів – 4,7%, а загальний показник – 5,1%. Тому переважна більшість студентів віддали перевагу використанню інтерактивної системи і вважають, що вона допомагає краще зрозуміти систему.

В анкеті для існуючої системи Petri Net було питання, пов'язане з проблемами системи. Користувачів запитали про їхню думку щодо того, що вони вважають проблемою системи. Три варіанти, які були дуже помітні на основі стандартів зручності для розробки інтерфейсу користувача, були надані користувачам як варіанти, а також їх попросили додати власну думку, якщо вони знайдуть більше проблем у системі. Їм було запропоновано три варіанти:

а. Коментарі довгі і не зрозумілі.

б. Візуалізація мережі Петрі є складною

с. Відсутність альтернативних візуалізацій

Деякі учасники додали власне уявлення про систему. Деякі з доданих коментарів були такими:

Існуюча система, студенти-комп’ютерники:

1. Відсутність контролю.

2. Система складна.

Інтерактивний інтерфейс, комп'ютер:

1. Drag/Drop зробить систему ще кращою.

2. Безпосередня взаємодія з елементами візуалізації.

3. Захоплюючий

В обох анкетах є два запитання, які ставлять про всю ідею інтерактивного виявлення вимог на основі онтології. Мета цього запитання – з’ясувати, чи задоволені учасники в цілому ідеєю налаштування програмного забезпечення за допомогою лінійки програмних продуктів.

З цієї причини користувачів просять, виходячи з описів, які були надані їм до роботи з системами, а також на основі їхнього досвіду роботи з системою та

попереднього досвіду, чи мають вони переваги щодо вибору класичного процесу розробки програмного забезпечення чи вибору лінійка програмних продуктів. Також їх запитують, наскільки, на їхню думку, ця лінійка програмних продуктів може покращити процес розробки програмного забезпечення. Загалом 78,2% користувачів вважали, що SPL може бути корисним для розробки та налаштування програмного забезпечення.

Останнє запитання в обох анкетах цікавило користувачів їхню загальну думку про систему. Користувачам було запропоновано поставити оцінки від 1 до 10 за їх припущення про рівень покращення, зробленого лінійкою програмних продуктів у процесі розробки програмного забезпечення. Середній бал покращення для інтерактивної системи вищий, ніж бал покращення для існуючої системи.

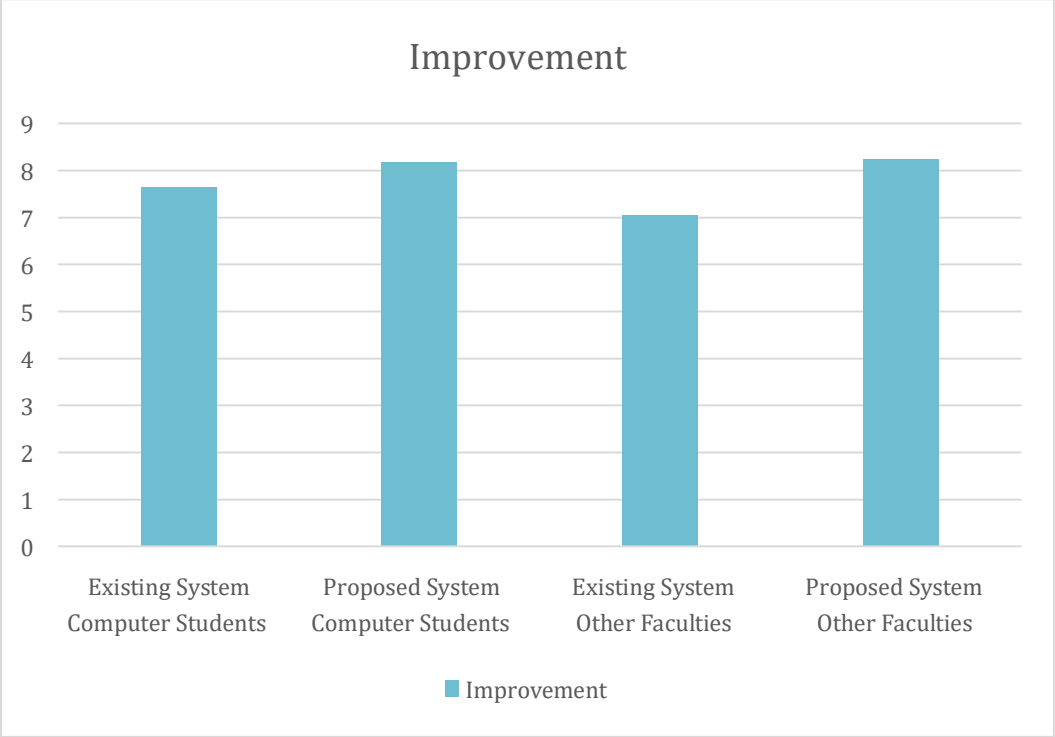


Рисунок 3.14 - Порівняння покращень

У таблиці 3.2 середні результати для користувачів були розділені та показані на основі їхньої академічної освіти, яка є або інформатикою, або бізнесом.

Таблиця 3.2 - Середні результати для всіх студентів

	Комп'ютерні студенти				Others			
	Existing		Proposed		Existing		Proposed	
	AVG	STDV	AVG	STDV	AVG	STDV	AVG	STDV
Легкість	6.83	1.12	8.44	0.83	6.13	1.29	8.50	1.05
Зрозуміло	6.54	2.20	8.01	1.58	4.36	1.23	6.40	1.90
Задоволення	7.22	1.27	8.11	1.33	7.44	1.18	8.38	1.05
Поліпшення	7.65	1.06	8.18	1.23	7.06	1.04	8.25	1.05
Час	8.33	1.94	7.89	1.76	9.19	1.91	8.13	1.13
Оцінка	6.06	0.97	6.59	0.59	4.94	1.26	5.94	0.99
Частота помилок	0.94	0.97	0.41	0.59	2.06	1.26	1.06	0.99

Підводимо підсумки всіх результатів, зібраних від учасників як для інтерактивної системи, так і для системи Petri Net:

Час, витрачений на інтерактивну систему, менше в порівнянні з системою на основі мережі Petri для обох груп. У обох групах підвищилися показники легкості та зрозумілості. Оцінка зрозумілості для користувачів, які не є комп'ютерними науками, становить 4,36 для системи на основі мережі Petri, тоді як для інтерактивної системи вона становить 6,40. Оцінка задоволеності студентів, які не вивчають інформатику, вища, ніж користувачів інформатики для обох систем. Це може бути пов'язано з різними рівнями очікувань у двох різних групах, або можна сказати, що студенти, які мають досвід роботи з комп'ютером, можуть мати більш реалістичні винятки інтерфейсу користувача, ніж люди, які не мають досвіду роботи з комп'ютером. Оцінка задоволеності для обох груп користувачів була вищою для інтерактивної системи. Обидві групи вважали, що SPL покращив процес розробки та налаштування програмного

забезпечення. Відсоток помилок як для студентів інформатики, так і для студентів, які не вивчають інформатику, зменшився майже вдвічі. Таким чином, можна зробити висновок, що інтерактивна система досягла успіху в покращенні користувацького досвіду за допомогою різних візуалізацій та вибору найкращої візуалізації для користувача.

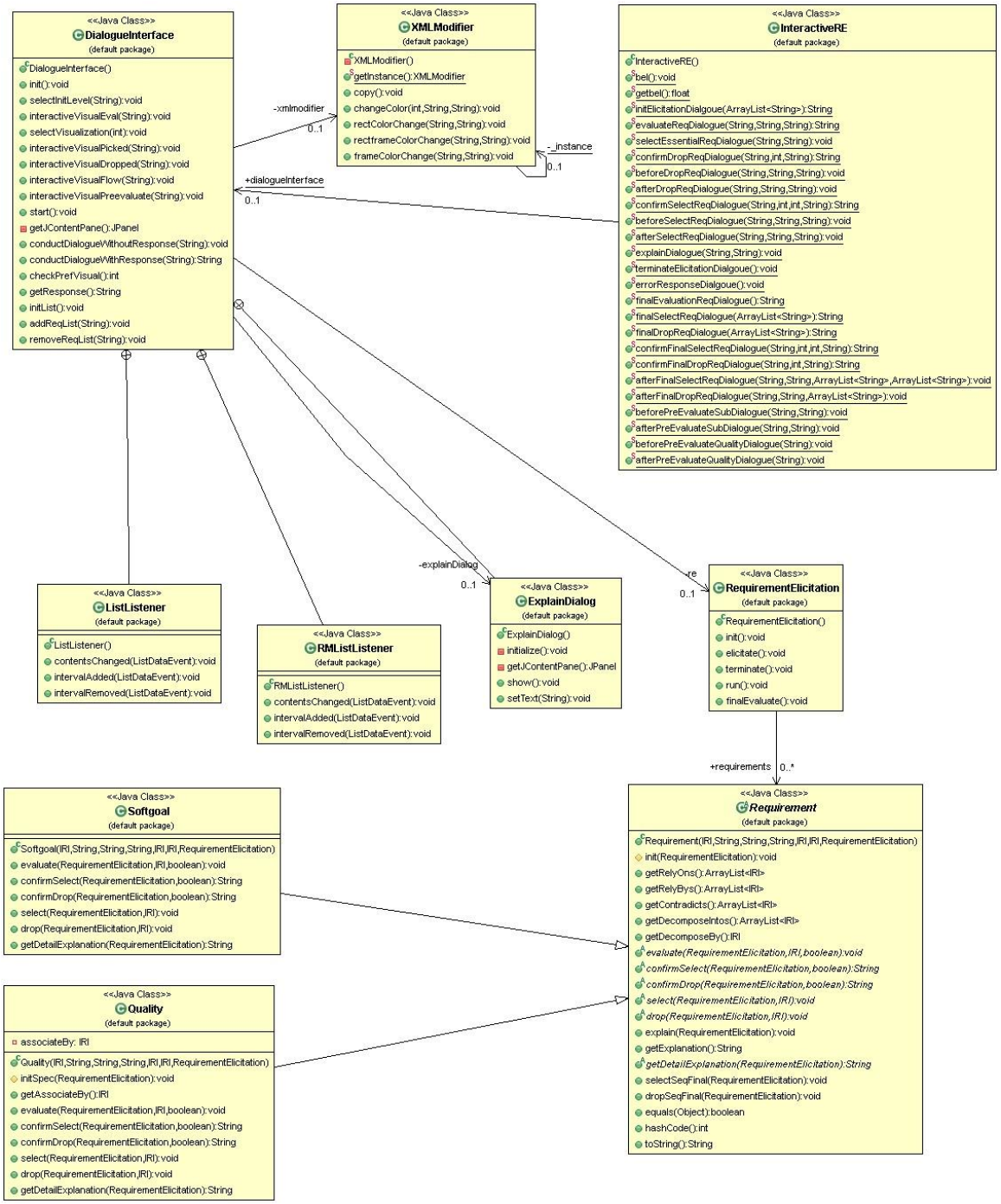


Рисунок 3.15 – Діаграма класів

Загалом, на основі зібраних результатів як від студентів інформатики, так і від студентів, які не є інформатиками, можна зробити висновок, що інтерактивна система підвищує ефективність процесу налаштування програмного забезпечення. Також це робить систему набагато привабливішою для користувачів із меншими знаннями про процес розробки програмного забезпечення.

ВИСНОВКИ

В дипломній роботі було введено інтерактивний підхід до візуалізації програмного забезпечення, що застосовується до попередньо розробленої системи виявлення вимог, це дає краще розуміння користувачам системи та зменшує час і зусилля, які їм потрібно витрачати на виявлення вимог, бажані вимоги.

Це дослідження проводилося в кілька етапів. Спочатку було вивчено існуючу текстову систему визначення вимог та систему візуалізації на основі мережі Петрі. Було вивчене дослідження юзабіліті, проведене для цих систем, що спонукало розробити інтерактивний підхід візуалізації для системи. Дослідження зручності використання було проведено на групі студентів з різних факультетів і різного академічного досвіду, щоб обґрунтувати, що запропонований дизайн може покращити зручність використання системи.

Результати дослідження показують, що в цілому користувачі мали позитивну думку щодо використання як інтерактивної системи, так і системи візуалізації на основі Petri Net. Однак на основі думок користувачів в середньому досягнуті всі параметри зручності, а саме: можливість навчання, ефективність, помилки та задоволеність користувачів, були покращені порівняно з системою інтерфейсу візуалізації на основі мережі Petri. Крім того, в середньому користувачі інтерактивного інтерфейсу виконували ті ж завдання швидше і точніше, ніж користувачі текстової системи. Загалом більшість користувачів обох систем віддають перевагу концепції програмної автоматизації, яка є основою обох систем порівняно з класичним процесом розробки програмного забезпечення.

Незважаючи на те, що інтерактивний підхід покращив зручність використання та якість системи налаштування програмного забезпечення, результати дослідження юзабіліті показують, що є ще багато можливостей для вдосконалення. Фактично пропонований фреймворк можна розглядати як ранню спробу інтерактивних інтерфейсів виключно у сфері налаштування програмного

забезпечення, у подальшій роботі знадобиться реалізувати цей підхід у різних напрямках дослідження. На даний момент існує чотири типи візуалізацій, які використовуються для різних користувачів, тому слід попрацювати над пошуком додаткових візуалізацій, які можуть ще більше підвищити зручність використання системи. Під час дослідження юзабіліті було виявлено, що користувачі зацікавлені у використанні інтерфейсу, в якому вони можуть мати пряму взаємодію з графічним інтерфейсом замість непрямого спілкування на основі діалогу.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

[1] M. N. Armstrong and C. Trudeau. Evaluating architectural extractors. 5th IEEE Working Conference on Reverse Engineering (WCRE'98), pages 30-39, Oct. 1998.

[2] F. Balmas. Displaying dependence graphs: a hierarchical approach. 8th IEEE Working Conference on Reverse Engineering (WCRE'01), pages 261-270, Oct. 2001.

[3] F. Balmas. Displaying dependence graphs: a hierarchical approach. Journal of Software Maintenance and Evolution: Research and Practice, 16(3):151-185, 2004.

[4] R. Balzer, J.-H. Jahnke, M. Litoiu, H. A. Muller, D. B. Smith, M. Storey, S. R. Tilley, K. Wong, and A. Weber. 3rd international workshop on adoption-centric software engineering. 3rd International Workshop on Adoption-Centric Software Engineering (ACSE'03), pages 1-2, May 2003.

[5] L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice. SEI Series in Software Engineering. Addison-Wesley, 1998.

[6] S. Bassil and R. K. Keller. Software visualization tools: Survey and analysis. 9th IEEE International Workshop on Program Comprehension (IWPC'01), pages 7-17, May 2001.

[7] B. Bellay and H. Gall. An evaluation of reverse engineering tool capabilities. Journal ofSoftware Maintenance: Research and Practice, 10(5):305-331, Sept./Oct. 1998.

[8] C. Best. Designing a component-based framework for a domain independnet visualization tool. Master's thesis, Department of Computer Science, University of Victoria, 2002.

[9] I. T. Bowman, M. W. Godfrey, and R. C. Holt. Connecting software architecture recovery frameworks. 1st International Symposium on Constructing Software Engineering Tools (CoSET'99), May 1999.

[10] P. Brown. Integrated hypertext and program understanding tools. IBM Systems Journal, 30(3):363-392, Mar. 1991.

[11] R. I. Bull, A. Trevors, A. J. Malton, and M. W. Godfrey. Semantic grep: Regular expressions + relational abstraction. 9th IEEE Working Conference on Reverse Engineering (WCRE'02), pages 267- 276, Oct. 2002.

[12] D. J. Carney and K. C. Wallnau. A basis for evaluation of commercial software. Information and Software Technology, 40(14):851- 860, Dec. 1998.

[13] K. Cremer, A. Marburger, and B. Westfechtel. Graph-based tools for re-engineering. Journal of Software Maintenance and Evolution: Research and Practice, 14(4):257-292, July/Aug. 2002.

[14] P. T. Devanbu. Re-targetability in software tools. ACM SIGAPP Applied Computing Review, 7(3): 19-26, Fall 1999.

[15] S. Ducasse, M. Lanza, and S. Tichelaar. The Moose reengineering environment. Smalltalk Chronicles, 3(2), Aug. 2001.

[16] S. Ducasse and S. Tichelaar. Dimensions of reengineering environment infrastructures. Journal of Software Maintenance and Evolution: Research and Practice, 15(5):345-373, Sept./Oct. 2003.

[17] M. Eichberg, M. Haupt, M. Mezini, and T. Schafer. Comprehensive software understanding with Sextant. 21st IEEE International Conference on Software Maintenance (ICSM'05), pages 315-324, Sept. 2005.

[18] J. Favre. GSEE: a generic software exploration environment. 9th IEEE International Workshop on Program Comprehension (IWPC'01), pages 233-244, May 2001.

[19] X. Franch and J. P. Carvallo. Using quality models in software package selection. IEEE Software, 20(1):34-41, Jan./Feb. 2003.

[20] J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development. 26th ACM/IEEE International Conference on Software Engineering (ICSE'04), pages 387-396, May 2004.

[21] M. Frohlich and M. Werner. The graph visualization system daVinci a user interface for applications. Technical Report 5/94, Department of Computer Science,

University of Bremen, 1994. ftp://ftp.tzi.de/tzi/biss/daVinci/ papers/techrepO594.ps.gz.

[22] J. Hainaut, V. Englebert, J. Henrard, J. -M. Hick, and D. Roland. Requirements for information system reverse engineering support. 2nd IEEE Working Conference on Reverse Engineering (WCRE'95), pages 136-145, July 1995.

[23] A. Hamou-Lhadj and T. C. Lethbridge. A survey of trace exploration tools. Conference of The Centre for Advanced Studies On Collaborative Research (CASCON'04), pages 42-55, Oct. 2004.

[24] I. Herman, G. Melancon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. IEEE Transactions on Visualization and Computer Graphics, 6(1):24-43, Jan.-Mar. 2000.

[25] J. H. Jahnke, H. A. Muller, A. Walenstein, N. Mansurov, and K. Wong. Fused data-centric visualizations for software evolution environments. 10th IEEE International Workshop on Program Comprehension (IWPC'02), pages 187-196, June 2002.

[26] R. Kazman. Tool support for architecture analysis and design. 2nd International Software Architecture Workshop (ISAW-2), pages 94- 97, Oct. 1996.

[27] R. Kazman, S. G. Woods, and S. J. Carriere. Requirements for integrating software architecture and reengineering models: CORUM II. 5th IEEE Working Conference on Reverse Engineering (WCRE'98), pages 154-163, Oct. 1998.

[28] H. M. Kienle. Exchange format bibliography. ACM SIGSOFT Software Engineering Notes, 26(1):56-60, Jan. 2001.

[29] H. M. Kienle. Building Reverse Engineering Tools with Software Components. PhD thesis, Department of Computer Science, University of Victoria, Nov. 2006. https: / /dspace. library. uvic.ca:8443/handle/1828/115.

[30] B. A. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. 26th ACM/IEEE International Conference on Software Engineering (ICSE'04), pages 273-281, May 2004.

[31] R. Koschke. Software visualization for reverse engineering. In S. Diehl, editor, Software Visualization, volume 2269 of Lecture Notes in Computer Science, pages 138-150. Springer-Verlag, 2002.

[32] R. Koschke. Software visualization in software maintenance. Journal ofSoftware Maintenance and Evolution: Research and Practice, 15(2):87-109, Mar. 2003.

[33] M. Lanza and S. Ducasse. Polymetric views a lightweight visual approach to reverse engineering. IEEE Transactions on Software Engineering, 29(9):782-795, Sept. 2003.

[34] T. C. Lethbridge and F. Herrera. Assessing the usefulness of the tksee software exploration tool. In H. Erdogmus and 0. Tanir, editors, Advances in Software Engineering: Topics in Comprehension, Evolution, and Evaluation, chapter 11, pages 73-93. Springer-Verlag, Dec. 2001.

[35] T. C. Lethbridge and J. Singer. Strategies for studying maintenance. 2nd Workshop on Empirical Studies of Software Maintenance (WESS'96), pages 79-83, Nov. 1996.

[36] J. Ma, H. M. Kienle, P. Kaminski, A. Weber, and M. Litoiu. Customizing Lotus Notes to build software engineering tools. Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'03), pages 276-287, Oct. 2003.

[37] J. I. Maletic, A. Marcus, and M. L. Collard. A task oriented view of software visualization. 1st IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'02), pages 32-42, June 2002. 8 Authorized licensed use limited to: Holger Kienle. Downloaded on August 13, 2009 at 08:27 from IEEE Xplore. Restrictions apply.

[38] M. S. Marshall, I. Herman, and G. Melancon. An object-oriented design for graph visualization. Software-Practice and Experience, 31(8):739-756, July 2001.

[39] J. Martin. Tool adoption: A software developer's perspective. 3rd International Workshop on Adoption-Centric Software Engineering (ACSE'03), pages 7-9, May 2003.

[40] S. Meyers. Difficulties in integrating multiple development systems. IEEE Software, 8(1):49-57, Jan. 1991.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи бакалавра: “Імплементація інтерактивних моделей візуалізації програмного забезпечення”

Обєм пояснювальної записки: 77 аркушів

Дата закінчення дипломної роботи 12 червня 2022р.

Підпис студента _____