Шифр «Роль»

Наукова робота на конкурс за напрямом:

Комп’ютерні науки

«Автоматизація розробки тестових сценаріїв для перевірки якості програмного забезпечення »

2019

ЗМІСТ

ВСТУП

Актуальність: Багаторівневий процес створення програмного забезпечення та ускладнення розроблюваних програмних засобів вимагає відповідних модифікацій, пов'язаних з процесами забезпечення якості. В рамках роботи компанії NetCracker виникла необхідність в реалізації програмного забезпечення для автоматизації процесу написання тестових сценаріїв.

Тема роботи: Автоматизація розробки тестових сценаріїв для перевірки якості програмного забезпечення.

Мета: Проектування технології автоматизації розробки тестових сценаріїв для перевірки якості програмного забезпечення.

Об’єкт дослідження: технологія тестування програмних систем підтримки телекомунікаційних операторів в Сумській філії компанії «NetCracker».

Предмет дослідження: технологія формування тестових сценаріїв для перевірки якості програмного забезпечення систем підтримки телекомунікаційних операторів.

Впровадження. Результати впроваджені в навчальний процес Сумського державного університету.

Гіпотеза дослідження: використання графової моделі, що тестується з подальшою генерацією тестових сценаріїв може бути засобом підвищення ефективності діяльності тестувальника програмного забезпечення.

Наукова новизна: на відміну від відомих моделей тестування програмного забезпечення, що не дозволяють генерувати інструкції для тестувальника, використовується модель переходу від графового представлення програми до тестового сценарію.

Практична значимість розробки полягає в можливості автоматизованого створення інструкції для оператора-тестувальника.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальна характеристика підприємства «NetCracker»

Одним з лідерів у галузі розробки програмного забезпечення в галузі телекомунікацій є компанія NetCracker, яка спеціалізується на створенні, впровадженні та супроводі систем експлуатаційної підтримки (OSS) для операторів зв'язку, великих підприємств і державних установ. Штаб-квартира компанії знаходиться в передмісті Бостона, місті Уолт, штат Массачусетс, США. Близько 70% співробітників в Сумському відділенні є інженерами з якості програмного забезпечення.

NetCracker Technology була заснована в 1993 році і досягла значних успіхів як самостійний гравець на ринку, ставши лідером в області створення і впровадження систем експлуатаційної підтримки. У 2008 році з метою подальшого вдосконалення продукту і зростання компанії було прийнято рішення про об'єднання з NEC Corporation. Глобальна інфраструктура NEC дозволила розширити сферу надання комплексних рішень для прискорення трансформації бізнесу операторів зв'язку. [1]. Програмний комплекс NetCracker призначений для розробки і введення в експлуатацію послуг, а також ефективного управління сервісної та ресурсної інфраструктурами.

NetCracker Framework - це ядро продукту, яке представляє собою технологічну платформу, що підтримується всіма модулями системи. Можливості платформи гарантують доступ всіх компонентів до загальної бази даних і надають засоби забезпечення безпеки. Всі модулі спроектовані на базі відкритої багаторівневої архітектури, що базується на стандартах Sun. Головні якості платформи - це відкритість, гнучкість, модульність і сумісність зі стандартами, що добре зарекомендували себе. NetCracker Framework містить уніфіковану гнучку об'єктну модель, бібліотеку шаблонів, інструменти для автоматизації процесів і управління трудовими ресурсами, систему захисту з підтримкою розмежування рівнів доступу, оболонку для створення звітів, професійні засоби управління і призначений для

користувача веб-інтерфейс з широким спектром можливостей візуалізації. Ядро продукту написано на Java EE з використанням EJB, рідний Application Server - Weblogic. За базу даних використовується Oracle. Промислові стандарти XML, CORBA, EJB, JMS, RMI, технології SOAP і галузеві стандарти забезпечення сумісності показано на рис. 1.1.

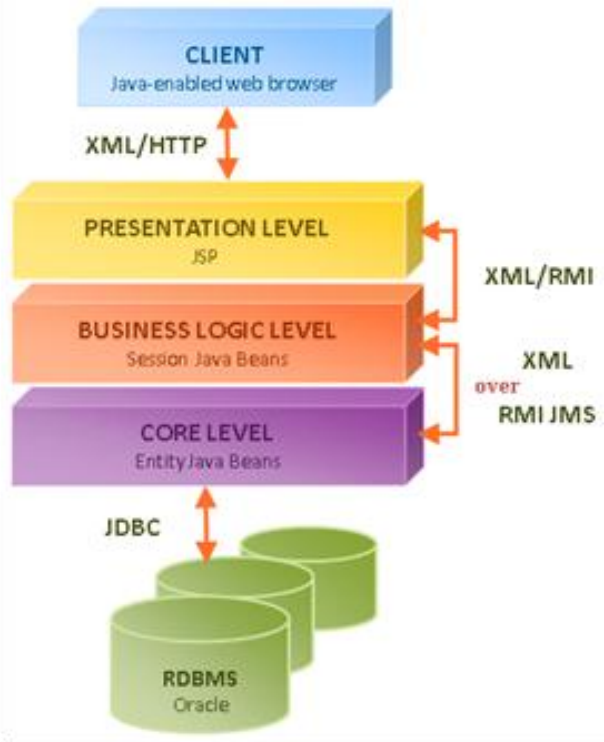

Рисунок 1.1 – Структура розроблюваних додатків

1.2. Аналіз основної проблематики тестування програмного забезпечення на підприємстві «NetCracker»

Програмні засоби великого масштабу мають величезну кількість компонент, складну структуру і взаємозв'язок між ними. Для перевірки працездатності реалізованого функціоналу створюються тестові сценарії. Тестовий сценарій - це послідовність дій на перевірку системи і відповідних результатів на виході. Дана інформація може надаватися у вигляді Excel документа, в якому кожен рядок являє собою стан системи, вплив на неї і отриманий результат.

Найбільш складним, творчим, неконтрольованим є процес інтерпретації вимог, специфікацій до відповідних тестових послідовностей. Таким чином, цей процес повністю лягає на плечі інженера з якості ПЗ. Вся зібрана і оброблена цією людиною інформація перетворюється в тестовий сценарій без створення будь-яких проміжних, допоміжних документацій, як це показано на рис. 1.2.

Рисунок 1.2 – Схема створення тестових сценаріїв

Таким чином, при цьому переході можуть виникнути такі проблеми [2, 3,4]: супровідність (при виникненні ситуації, в результаті якої інженер не може завершити написання тестового сценарію, нова людина, призначена на цю задачу, повинна максимально швидко визначити обсяг виконаної і майбутньої роботи); підтримуваність (пов´язана з доповненням, зміною, уточненням вимог та необхідністю у редагуванні вже готових сценаріїв перевірки, при чому залишається проблема визначення пріоритетності перевірки, що ускладнює процес модифікації сценаріїв); детермінація (перетворення широкого тестового сценарію відповідно до спеціальної перевірки конкретної частини функціоналу, що займає багато часу за умови роботи з текстовим документом); повнота перевірок (процес перетворення вимог, специфікацій в тестові сценарії виконується повністю по особистісним міркуванням конкретного інженера. Ця дія має бути доповнена будь-якою документацією, перевірками, контролем, що дозволяє виявити непокриті сценарії).

ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Аналіз методів вирішення задачі генерації тестових сценаріїв

Аналіз практики роботи відділів тестування підприємства і дослідження літературних джерел [5-14] дозволили виявити основні технології, підходи, які можливо використовувати для створення тестових сценаріїв.

Технологія «High Level TC» дозволяє розглянути перший варіант вирішення проблеми. В даному випадку інженер в процесі написання сценарію перевірки створює шаблонний документ, який спрощено описує основні дії. Цей документ окремо зберігається і на його основі пишеться більш детальне технічне завдання (Додаток Б, рис. Б1).

В результаті ми отримуємо такі позитивні сторони: створено проміжний документ; частково спрощений процес аналізу основних кроків і перевірок.

Негативними (неврахованими) сторонами є наступне: формат надання даних залишився той же; відсутня автоматизація; формалізацію процесу створення тестових сценаріїв не враховано.

Технологія «Handmade Graph» в даний час розглядається як альтернатива попереднього методу і досліджується можливість її впровадження на підприємствах NetCracker [1].

Розглянемо варіант з використанням орієнтованого графа після створення детального тестового сценарію. В цьому випадку інженер виконує ті ж дії, а саме: аналізує вимоги, дизайн специфікацію, на їх основі пише детальний тестовий сценарій. На основі отриманого документа формує граф, що описує всі дії, запропоновані в раніше задокументованому ТС (Додаток Б, рис. Б2).

В результаті ми отримуємо наступні рішення: надано узагальнений документ опису дій, графічне представлення дозволяє більш швидко вивчити існуючі перевірки.

Негативні (невраховані) сторони: на основі детального кейса необхідно описувати детальний граф (додатковий час); при складанні графа можуть бути знайдені повторювані або відсутні гілки (дії), в результаті чого необхідно буде правити тестовий сценарій для виключення повторень або необроблених ситуацій; відсутня автоматизація.

Технологія ATSD (Avtomation test scenariou's generation) розглядається підприємством як одна з альтернативних технологій автоматизації тестування. Розглянемо варіант з побудовою графа на основі вимог і специфікації. В цьому випадку першим документом, створеним інженером буде орієнтований граф, так як найбільш зручним способом представлення сценаріїв перевірки є орієнтований граф зі всіма варіантами його проходу. Таке уявлення дозволяє в найкоротші терміни і більш точно описати, охарактеризувати частину функціоналу, покриту даними перевірками.

На основі отриманого графа буде автоматично створюватися шаблон текстового документа. Після його доповнення та деталізації вийде фінальний документ з описаними тестовими сценаріями (Додаток Б, рис. Б3).

В результаті ми отримуємо наступні рішення: надано узагальнений документ опису дій; графічне представлення дозволяє більш швидко вивчити існуючі перевірки; при створенні шаблону документа використовується автоматичне перетворення з графічного в текстовий вигляд; створено проміжний етап між аналізом і написанням сценаріїв.

Єдиною негативною стороною при даному варіанті, можна вважати, необхідність фахівця мати знання для роботи з графічним редактором. Напрямки змін, описані раніше, в даному випадку майже повністю покриті.

2.2 Вибір базової технології розробки тестових сценаріїв

Показниками якості альтернатив на підприємстві визначено такі критерії:

- Необхідний час на створення програмного продукту для нової технології написання тестових сценаріїв. Назвемо цей критерій ступінь трудовитрат з розробки програмного забезпечення.
- Процес створення тестового сценарію повинен мати чіткі послідовності дій, переходів, умов завершення. Назвемо цей критерій ступінь формалізації процесу побудови тестового сценарію.
- Представлення даних має бути зручним з точки зору користувача і дозволяти в найкоротші терміни сприйняти інформацію. Цей критерій буде називатися ступінь наочності.
- Рутинні і найбільш трудомісткі кроки повинні бути виключені з процесу посередництвом автоматизації. Цей критерій буде називатися ступінь автоматизації створення тестового шаблону.

Таким чином завдання полягає в багатокритеріальній оцінці альтернатив. Для оцінки важливості критеріїв і «ступеня важливості альтернатив вимогам критеріїв», використовуємо метод експертного оцінювання фахівцем з тестування підприємства.

Таким чином завдання може бути зведене до задачі вибору і використання методу аналізу ієрархій [18].

Суть завдання вибору полягає в наступному: у методі аналізу ієрархій (MAI) передбачається структуризація проблеми у вигляді ієрархії. Ієрархія будується з вершини - це загальна мета або фокус проблеми. У загальному випадку цілей може бути кілька. За фокусом слідує рівень найбільш важливих критеріїв. Кожен з критеріїв може поділятися на субкритерії, за якими слідує рівень альтернатив. Етап формування множини альтернатив і критеріїв здійснюється з урахуванням рекомендацій і не формалізується.

Ієрархія - це певний тип системи, заснований на припущенні, що елементи системи можуть групуватися в незв'язані множини. Елементи кожної групи перебувають під впливом елементів іншої групи і в свою чергу впливають на елементи наступної групи.

Математично ієрархія та її властивості можуть бути описані наступним чином [15]: на множині об'єктів $i = \{1,2...,N\}$ визначається ієрархічна структура шляхом задання орграфу $G = (i,W)$, $W \subset i \times i$, який:

а) розбиває вершини на непересічні рівні: $i = U_i V_j; l = \overline{1,m}; V_i \cap V_j = \varnothing; i, j = \overline{1,m}$

b) $(i,j) \in W$ означає, що вага Zi об′єкта i напряму залежить від ваги Zi об′єкта j;

c) якщо (i,j) - дуга графа G, тобто $(i, j) \in W$, то об′єкти i та j знаходяться на сміжних рівнях, тобто знайдеться таке, що $i \in V_{k+1}, j \in V_k$;

d) ваги Zi об'єкта $i \in V_{k+1}$ визначаються через ваги Zj вершин множини $L_i = \{j \mid (i,j) \in W\} \subseteq V_k$, в які ведуть дуги з вершини і за допомогою залежності, феноменологічно вводиться: $Z_i = \sum_{j \in L_i} \vartheta_{ij} Z_j$, де ϑ_{ij} - вага дуги (i, j).

На другому етапі МАІ реалізується принцип дискретизації думок. Суть його заключається в тому, що, використовуючи думку ОПР / експерта і визначенні алгоритмів їх обробки, установлюються ваги ϑ_{ij} дуг $(i,j) \in W$і і ваги Zj об'єктів першого рівня ($j \in V1$). Якщо на першому рівні один об'єкт , то вага його приймається за 1 **(**$Z_1 = 1$).

При даному дослідженні використовується метод парних порівнянь, суть якого заключається в наступному: нехай задана фіксована множина об'єктів $K = \{k_i\}, i = \overline{1,n}, K \subset I$, які порівнюються попарно з точки зору їх переваг, необхідності, важливості і т.д. Результати записуються у вигляді матриці парних порівнянь $R = \{r_{ij}\}, i,j = \overline{1,n}$, що відображає не лише факт, але і ступінь переваги. При цьому використовується шкала відносної важливості, вибір якої залежить від вимог до сприйняття ОПР переваг. Зі шкали формується властивість гомогенності (однорідності) об'єктів. Гомогенність

надзвичайно важлива, оскільки людський мозок не може адекватно оцінювати об’єкти, між якими спостерігаються великі відмінності певних характеристик.

Для реалізації вибору використовуємо ППП СППР «Выбор» [25]. Основні етапи процедури приведені у Додатку В (Рисунок В1- В7), за допомогою аналізу яких в якості базової обрано технологію «ATSG».

2.3 Постановка задачі автоматизації розробки тестових сценаріїв

Таким чином, в якості базової технології на основі якої буде розроблятися тестовий сценарій обрана технологія «ATSG». Виходячи їх основних етапів цієї технології задачу автоматизації написання сценаріїв можна сформулювати наступним чином.

Припустимо, що існує деякий модуль програмного продукту, для перевірки якого необхідно виконати n-у кількість перевірок, кожна з яких містить в собі I_i кроків, де $i = 1..n$. В цьому випадку для створення тестового сценарію необхідно виконати наступні дії:

a) На першому етапі необхідно побудувати орієнтований граф, що описує всі можливі перевірки. При цьому повинна бути надана можливість уточнювати, деталізувати необхідні вузли графа.

b) Другим кроком є можливість розбиття загального графа всіляких перевірок на спрощену, лінійну послідовність кроків. Це дозволяє виділити всі допустимі сценарії перевірок.

c) Фінальним етапом є перетворення графа в шаблон документа, який однозначно і детально описує дії, необхідні для виконання перевірки. При виконанні цих дій необхідно визначити семантичні описи об'єктів графа і їх еквівалент в текстовому документі (Word або Excel).

3 АПАРАТ МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ДЛЯ ПРОЦЕСІВ З АВТОМАТИЗАЦІЇ ТЕСТОВИХ ПЕРЕВІРОК

3.1 Формальна модель тестових перевірок

Для моделювання тестових перевірок будемо використовувати формальну модель тестових перевірок, запропоновану автором в роботі. У цій моделі процес тестування може бути представлений у вигляді двох кінцевих множин та зв'язків елементів цих множин між собою [16]. Математична нотація представлена у вигляді $Д_T = \{\{Д\},\{Ф\}\}$, де: Дт - формальна модель тестування; {Д}- множина дій; {Ф}- множина станів системи.

Нотація означає наступне: «Тестування - це множина дій, що проводяться над множиною станів системи». Множина {Д} визначається як кінцева множина дій, виконання яких допустимо в межах даної системи тестування. Множина форм {Ф} - кінцева множина станів, які може приймати система після проведення над ними дій з множини {Д}. Таким чином, сукупність всіх станів тестування представляє кінцеву множину, яка повністю описує всі можливі сценарії вхідних даних і переходів.

3.2 Модель зв’язаності поведінкових одиниць на графах

Для побудови графа моделі тестування пропонується використовувати спосіб відображення тестування графами. Для завдання множини вершин графа будемо використовувати множину можливих станів системи. Ребра графа задамо за допомогою множини дій Д. Встановимо цю відповідність таким чином, щоб виконувалися наступні правила:

- одній вершині графа відповідає тільки один елемент множини Ф;
- одному ребру графа відповідає тільки один елемент множини Д;
- одному елементу множини Ф відповідає тільки одна вершина графа;
- одному елементу множини Д відповідає тільки одне ребро графа.

Таке тотожне відображення множини станів Ф в множину вершин *v* і множину станів Д в множину ребер е можна математично визначити наступним чином: для будь-якого *i* справедливе твердження $v(i) = Ф(i)$ і $e(i) = Д(i)$, де $i \in I, I = 1,2,3...n$. Тобто визначаються дві парних граматики - перша граматика для встановлення перекладу Ф в *v*, друга граматика - для встановлення перекладу Д в *e*.

Таким чином, зв'язки між вершинами тотожно відповідають зв'язкам станів модельованого тестового сценарію. У графі тестування вершини графа з'єднують ребра в тому і тільки в тому випадку, якщо відповідні вершин стану пов'язані дією, відповідною ребру.

Спрямованість ребер встановлюється таким чином, щоб відображати логіку послідовності зміни станів тестування. Вершина *i* є вхідною вершиною для вершини *j* через ребро *k* в тому і тільки в тому випадку, якщо стан *i* змінюється на стан *j* після вчинення дії *k*. Таким чином, станам y_1, $y_2,...y_n$ зіставляються вершини графа $v_1, v_2,...v_n$, і кожна пара вершин v_i і v_j з'єднана дугою e_{ij},, що йде від v_i до v_j в тому і тільки в тому випадку, коли стан v_i є вхідним станом для v_j [21].

3.3 Матричний метод знаходження шляхів у графах

Матриця суміжності повністю визначає структуру графа [19]. Зведемо матрицю суміжності в квадрат за правилами математики. Кожен елемент матриці А2 визначається за формулою:

$$a_{ik}^{(2)} = \sum_{j=1}^{n} a_{ij} a_{jk}$$

Доданок у формулі дорівнює 1 тоді і тільки тоді, коли обидва числа aij і ajk рівні 1, в іншому випадку він дорівнює 0. Оскільки з рівності aij = ajk = 1 слідує існування шляху довжини 2 з вершини x_i в вершину x_k, що проходить через вершину x_j , то (i -й, k -й) елемент матриці А2 дорівнює числу шляхів довжини 2, що йдуть з x_i в x_k.

Так "1", що стоїть на перетині другого рядка і четвертого стовпця, говорить про існування одного шляху довжиною 2 з вершини x_2 до вершини x_4. Дійсно, як бачимо в графі на рис.3.1, існує такий шлях: a6, a5. "2" в матриці A2 говорить про існування двох шляхів довжиною 2 від вершини x3 до вершини x6: a8, a4 і a1, a3.

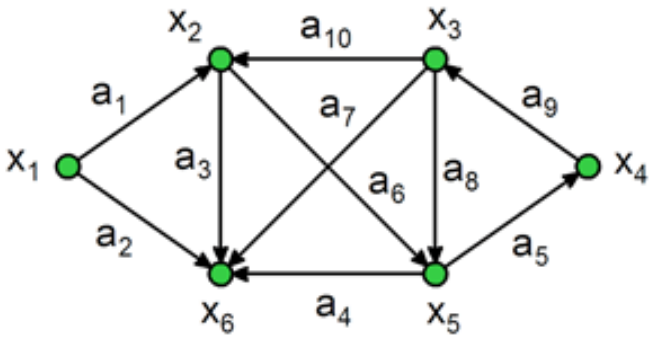

Рисунок 3.1 – Орграф

Аналогічно для матриці суміжності, яка була зведена в третю степінь A3, а $^{(3)}$ i_k дорівнює числу шляхів довжиною 3, йдуть від x_i до x_k. З четвертого рядка матриці A3 видно, що колії довжиною 3 існують: один з x_4 в x_3 (a_9, a_8, a_5), один з x_4 в x_5 (a_9, a_{10}, a_6) і два шляхи з x_4 в x_6 (a_9, a_{10}, a_3 і a_9, a_8, a_4). Матриця A4 показує існування шляхів довжиною 4. Таким чином, якщо a_{ik}^p є елементом матриці Ap, то a_{ik}^p дорівнює числу шляхів (не обов'язково орланцюгів або простих орланцюгів) довжини p, що йдуть від x_i до x_k.

Завдання побудови тестових сценаріїв включає в себе наступні умови побудови і проходження:

• існує тільки один початковий стан (вершина);

• граф містить кілька кінцевих станів (вершин);

• граф містить в собі цикли, які повинні бути пройдені тільки один раз;

• до виконання пошуку шляхів, граф повинен бути перевірений на наявність вузлів які не беруть участі в побудові маршрутів (зайвих вершин);

• граф повинен бути орієнтованим;

• відсутність петлі на вихідній вершині.

Таким чином перед виконанням алгоритму розбиття орграфа необхідно виконати знаходження множини вершин, що входить в кінцеві маршрути. Для цього виконуємо перетин транзитивного замикання для початкової вершини і зворотного транзитивного замикання для кінцевих вершин. У разі відсутності будь-якої вершини, можна сказати що вона є зайвою і не приймає участі в будь-якій перевірці. Після виконання перевірки необхідно побудувати всі можливі шляхи для описаного графа.

Спрощена схема алгоритму пошуку шляхів представлена у Додатку Г (рис.Г1). Першою дією будується матриця суміжності (Табл. 2.1).

Таблиця 2.1 – Матриця суміжності

	X0	X1	X2	X3	X4
X0	0	1	0	0	0
X1	0	0	1	0	0
X2	0	0	1	1	1
X3	0	1	0	0	0
X4	0	0	0	0	0

За отриманою матрицею вибираються рядки, що містять нульові елементи. Отримані вершини будуть кінцевими вузлами графа. За створеною матрицею будується масив шляхів, який буде собою представляти послідовність вершин, які необхідно пройти для досягнення кінцевого стану.

Для цього по таблиці суміжності для вершини X4 визначаються попередні вершини. Для цього по стовпцю X4 знаходяться рядки в яких є 1. Це вершина X2. Таким чином ми отримуємо шлях 2> 4. Тепер таким же чином визначаємо попередню вершину для X2. У нашому випадку вершина X2 має дві попередні вершини - це X2 і X1. Таким чином один шлях розбивається на два: 2> 2> 4 і 1> 2> 4.

Виконуємо пошук попередньої вершини для кожного нового маршруту. При цьому необхідно виконати перевірку, що даний маршрут не

повторюється. Для цього виконується перевірка наявності одних і тих же вершин: в нашому випадку це маршрут 2> 2> 4. У ньому повторюються вершини 2.

Беремо більш ранню вершину 2 (та, яка стоїть після 4) і дізнаємося, яка вершина стоїть перед нею - це 2. Тепер визначаємо, які можливі вершини можуть бути в останньому випадку - це 1 і 2.

Якщо певна і можлива вершини збігаються, то дана вершин не береться до уваги. Таким чином ми отримуємо тільки наступний шлях - 1> 2> 2> 4. І для маршруту 1> 2> 4: 0> 1> 2> 4 і 3> 1> 2> 4

Таким чином є три маршрути:

• 1> 2> 2> 4;

• 0> 1> 2> 4;

• 3> 1> 2> 4

Один з маршрутів досяг вихідної вершини, він є завершеним. При повторенні виконання даного алгоритму ми отримаємо наступні результати:

$$\begin{pmatrix} 0>1>2>4;\rightarrow END \\ 1>2>2>4;\rightarrow \begin{matrix} 0>1>2>2>4;\rightarrow END \\ 3>1>2>2>4;\rightarrow 2>3>1>2>2>4;\rightarrow 1>2>3>1>2>2>4;\rightarrow 0>1>2>3>1>2>2>4;\rightarrow END \end{matrix} \\ 3>1>2>4;\rightarrow 2>3>1>2>4;\rightarrow 1>2>3>1>2>4;\rightarrow 0>1>2>3>1>2>4;\rightarrow END \end{pmatrix}$$

Рисунок 3.2 – Можливі шляхи проходження графу

Таким чином у нас є 4 можливих маршрути:

- 0>1>2>4;
- 0>1>2>2>4;
- 0>1>2>3>1>2>2>4;
- 0>1>2>3>1>2>4.

4 РЕАЛІЗАЦІЯ ДОДАТКУ

4.1 Вибір програмного продукту для побудови орієнтованого графу

До розгляду було прийнято наступні програмні продукти для роботи з графічним представленням складних систем:

- Програма GraphEd [26].
- Програма GRph INterface (GRIN) [27].
- yEd Graph [24].
- Microsoft Vision Professional [28].
- Програма Gnet.
- Програмний продукт CharGer.

В якості критеріїв вибору експертами компанії «NetCracker» обрано наступні аспекти: вартість, трудомісткість процесів формалізації даних, простота і зручність використання.

Таким чином, дана задача може бути вирішена методом аналізу ієрархії. За результатами вирішеного завдання побудовано діаграму оцінки якості альтернатив (Рис.4.1), де лідерську позицію зайняв ПП yEd Graph.

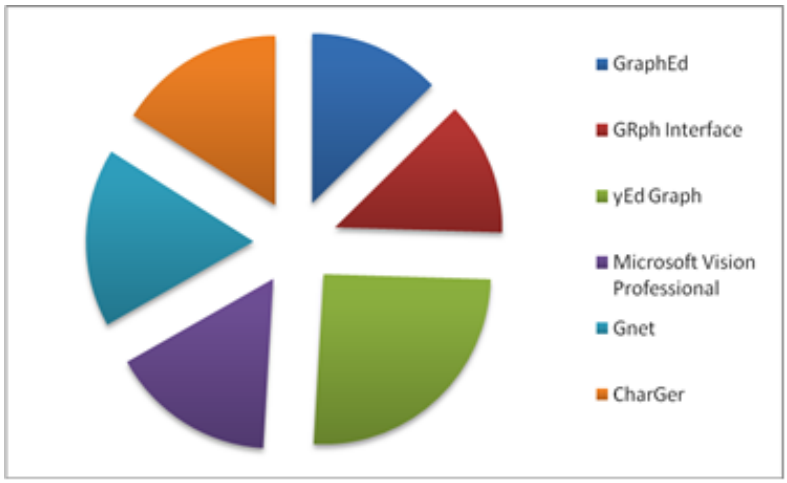

Рисунок 4.1 – Альтернативні варіанти ПП візуалізації графової моделі

4.2 Створення формалізованого опису моделі програми у термінах yEd Graph

Використання програмного засобу yEd Graph дозволяє після введення структури графа отримувати його формалізований опис на спеціальній мові. Цей опис надалі з використанням програми, яку потрібно розробити, може бути приведений в формат Excel документа. Для розробки такої програми перекладу розглянемо основні характеристики формалізованого опису графової моделі на мові graphml. Для цього побудуємо модель фрагмента програми. Показаний на рис. 4.2 граф містить 11 вузлів і 12 ребер.

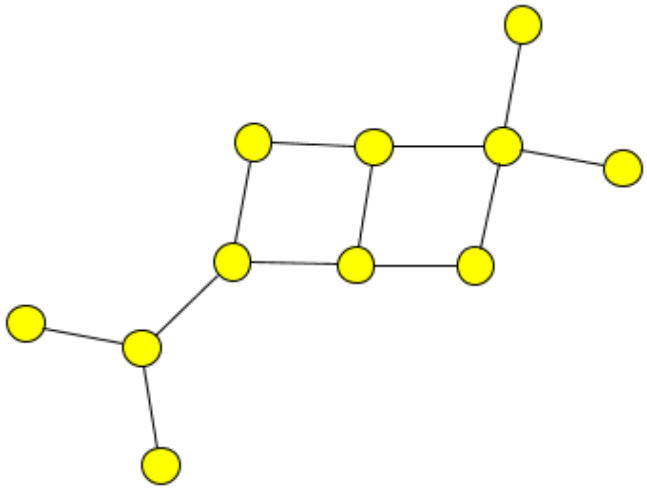

Рисунок 4.2 – Приклад графу, що описує частину програмного компонента «Workforce Management»

Формалізований опис програми презентований у вигляді рисунків у Додатку Д. Розглянемо основні особливості даної моделі, необхідні для подальшої роботи програми.

Перший рядок документа - це інструкція обробки, яка визначає, що документ є підмножиною стандарту XML 1.0, і що документ виконаний в кодуванні UTF-8 (Додаток Д, Рисунок Д1). Звичайно, для GraphML-документів можуть бути обрані й інші кодування.

GraphML-документ складається з елемента graphml і ряду піделементів: graph, node, edge. В кінці розділу розглянемо перераховані елементи докладніше, і покажемо, як вони визначають граф. Розглянемо фрагмент, загальний для всіх GraphML-документів, заснований на елементі graphml (Додаток Д, Рисунок Д2).

Другий рядок містить кореневий елемент GraphML-документа: graphml. Елемент graphml, також як і всі інші елементи мови GraphML, належить іменному простору http://graphml.graphdrawing.org/xmlns. Наступні два XML-атрибута визначають XML-схему даного документа. У нашому прикладі ми використовуємо стандартну схему GraphML-документа, розташовану на сервері graphdrawing.org. Перший атрибут, xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance" - визначає, xsi як префікс іменного простору XML-схеми. А другий, xsi: schemaLocation = "http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd" визначає місцезнаходження XML-схеми для елементів іменного простору GraphML.

Граф позначається за допомогою елемента graph. Елементи розташовані всередині елемента graph забезпечують оголошення вузлів і ребер. Вузол оголошується за допомогою елемента node, а ребро за допомогою елемента edge (Додаток Д, Рисунок Д3).

Граф в GraphML - змішаний, іншими словами він може містити спрямовані і неспрямовані ребра одночасно. Якщо при оголошенні ребра його спрямованість не визначена, то застосовується спрямованість, що задана за замовчуванням. Спрямованість ребер, що привласнюється за замовчуванням, задається XML-атрибутом edgedefault елемента graph. Цей XML-атрибут може приймати одне з двох значень: directed і undirected. Значення за замовчуванням має бути задано обов'язково.

Додатково, за допомогою атрибута id, графу може бути присвоєно ідентифікатор. Ідентифікатор привласнюють тоді, коли на даний граф потрібно організувати посилання.

Вузол в графі оголошується за допомогою елемента node. Кожен вузол має унікальний (в межах даного документа) ідентифікатор. Ідентифікатор вузла задається за допомогою XML-атрибута id.

Ребро в графі оголошується за допомогою елемента edge. Кожне ребро має дві кінцеві точки, що задаються за допомогою XML-атрибутів source і target. Значення атрибутів source і target повинні містити ідентифікатори вузлів, визначених у тому ж документі, що і ребро.

Ребра з одною кінцевою точкою, звані петлями, циклами, або замкнутими ребрами, визначаються за допомогою однакових значень, заданих в атрибутах source і target.

За допомогою механізму розширення, який називається GraphML-атрибутом для елементів графа може бути задана додаткова інформація простого типу. Простий тип має на увазі, що дані обмежені скалярними величинами. Наприклад, числами і рядками.

Якщо в елементи графа вам необхідно додати структуровані дані, то ви повинні використовувати механізм розширення GraphML під назвою ключ / дані (data / key).

GraphML-атрибут оголошується за допомогою елемента key який задає ідентифікатор, ім'я, тип, і домен атрибута. Ідентифікатор задається XML-атрибутом id і використовується для посилання на даний GraphML-атрибут всередині документа.

Ім'я GraphML-атрибута визначається за допомогою XML-атрибута attr.name і має бути унікальним серед всіх оголошених в документі GraphML-атрибутів. Ім'я потрібно для того, щоб програми могли ідентифікувати даний атрибут. Зверніть увагу, що ім'я GraphML-атрибута не використовується для посилань усередині документа, для цього використовується ідентифікатор.

Тип GraphML-атрибута може бути boolean, int, long, float, double, або string. Ці типи визначені відповідно до аналогічних типів у мові Java (TM).

Домен GraphML-атрибута визначає перелік елементів в яких GraphML-атрибут може бути оголошений. Можливі значення: graph, node, edge, і all.

Значення GraphML-атрибута в елементі графа задається за допомогою елемента data, вкладеного в даний елемент. Елемент data має XML-атрибут key, який посилається на ідентифікатор GraphML-атрибута. Значення GraphML-атрибута задається текстовим вмістом елемента data. Це значення повинно мати тип, оголошений у відповідному елементі key (Додаток Д, Рисунок Д4).

Для оптимізації синтаксичного розбору документа за допомогою парсера використовуються спеціальні метадані, які можуть бути додані до деяких GraphML-елементів за допомогою XML-атрибутів. Всі XML-атрибути, які визначають метадані мають префікс parse. Є два види метаданих: інформація про кількість елементів і інформація про спосіб кодування даних.

По-перше, розглянемо інформацію про кількість елементів. Для елемента graph визначені наступні XML-атрибути: parse.nodes задає кількість вузлів в графі, parse.edges - кількість ребер, parse.maxindegree визначає максимальну кількість вхідних в вершину ребер, parse.maxoutdegree - максимальну кількість вихідних з вершини ребер. Для елемента node parse.indegree визначає максимальну кількість вхідних ребер для даної вершини, parse.outdegree - максимальну кількість вихідних ребер для даної вершини.

По-друге, розглянемо інформацію, пов'язану з кодуванням. Для елемента graph визначені наступні XML-атрибути: якщо parse.nodeidsi має значення canonical, то всі вузли мають ідентифікатор виду nX, де X означає кількість елементів node, що передують даному елементу. Друге значення XML-атрибута - free. Аналогічно цьому, XML-атрибут parse.edgeids задає вид ідентифікатора для вузлів. Відмінність полягає лише в тому, що ідентифікатор має вигляд eX. XML-атрибут parse.order визначає порядок в якому вузли і ребра розташовуються в документі. При значенні, рівному nodesfirst, всі елементи node розташовуються раніше елементів edge. При значенні, рівному adjacencylist, оголошення вузла передує оголошенню його

суміжних вершин. Для значення free порядок проходження не встановлюється. Рисунок Д5 у Додатку Д ілюструє використання інформації для парсера.

4.3 Розробка інформаційної технології TCGenerator

У процесі розробки програмного продукту було визначено основні класи, що представлені у Таблиці 4.1.

Таблиця 4.1 – Основні класи програми TCGenerator

Назва класу	Опис
GNode	Призначений для зберігання інформації про вузол графу
GEdge	Призначений для зберігання інформації про гілку графу
GPath	Призначений для зберігання інформації про шлях, а саме послідовність (GNode)
GMatrix	Представляє собою матрицю суміжності
GRAPHMLParcer	Призначений для перетворення graphml файлу в програмне представлення і навпаки
Excel	Реалізує методи перетворення програмного представлення графа в Excel документ
Algorithm	Реалізує алгоритм проходу по графу
GMLView	Користувацький інтерфейс

Приклад алгоритму реалізованого у класі Algorithm представлено у Додатку Г, Рисунок Г2.

Опис основних класів за допомогою нотації UML представлено у Додатку Ж, Рисунки Ж1 – Ж7. Відповідно до загального вигляду користувацького інтерфейсу, який представлено на Рисунку 4.3, приведемо опис основних полів додатку і їх призначення:

- «File> Open» - дозволяє вибрати потрібний файл графічної моделі в форматі GRAPHML.

- «File> Exit» - вихід з програми.
- «Help> About» - коротка довідка.
- «Curent file» - поле для відображення обраного graphml файлу.
- Кнопка «Generate Paths» - кнопка для генерації всіх можливих шляхів побудованих на основі обраного графа.
- «Таблиця» - таблиця, яка заповнюється після генерації шляхів. Зберігає інформацію про номер шляху, послідовності вузлів, імені graphml і excel файлів, створених програмою.
- Кнопка «Generate Excel» - кнопка для генерації Excel файлів, на основі обраних в таблиці шляхів.

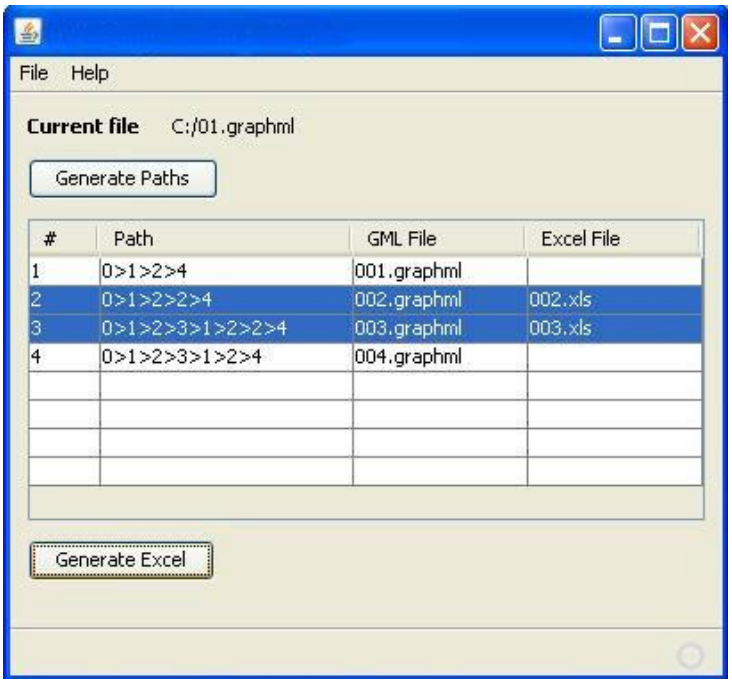

Рисунок 4.3 – Користувацький інтерфейс фрагменту програми TC Generator

Розроблена технологія являє собою послідовність робіт з перетворення вимог і специфікацій до детального сценарію тестування програми. Структура технології представлена на рис. 4.4. Проведений якісний аналіз зниження трудомісткості наведено на рис. 4.5. При цьому враховано наступні критерії: зменшення кількості помилок в програмах, зниження трудомісткості тестування.

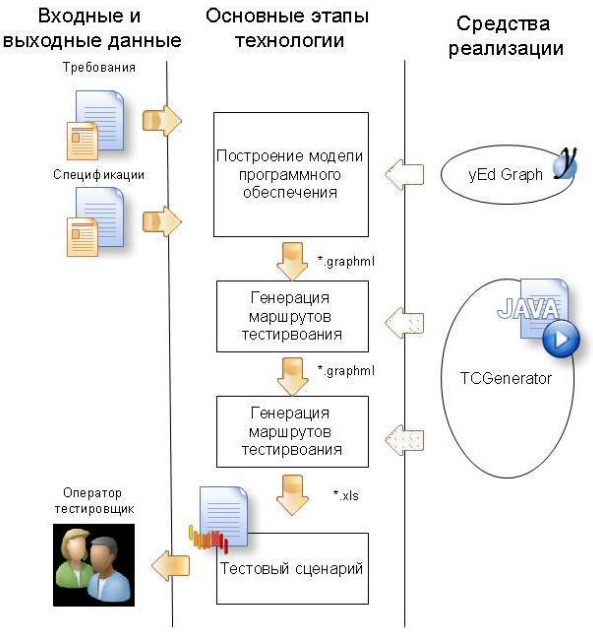

Рисунок 4.4 – Структура технології створення тестового сценарію

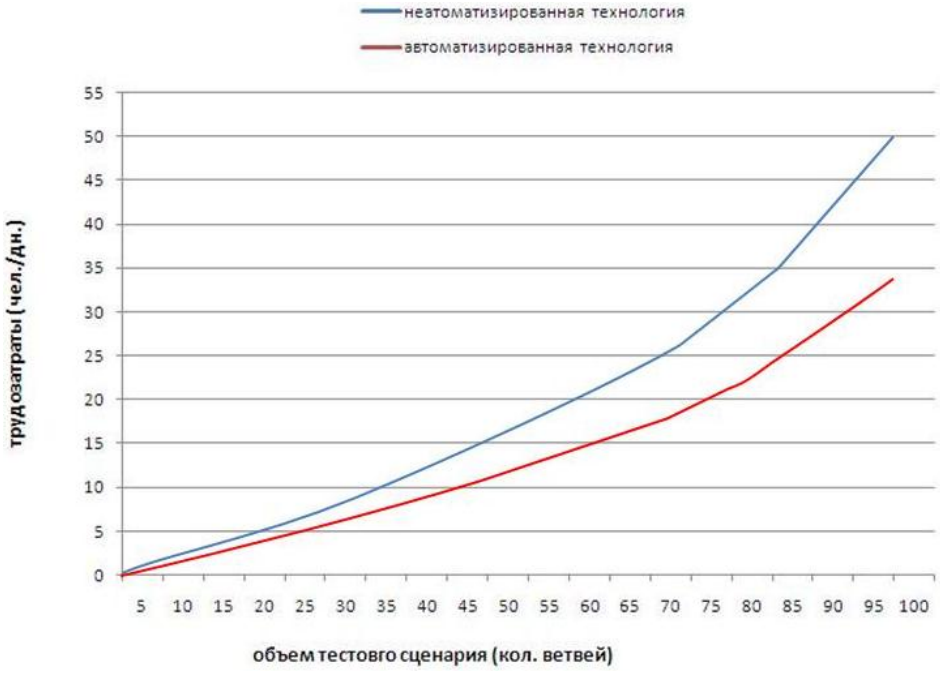

Рисунок 4.5 – Якісний аналіз впливу автоматизації написання ТС на трудомісткість

ВИСНОВКИ

У ході виконання даної роботи було проаналізовано діяльність підприємства «NetCracker». В ході аналізу виявлено актуальне завдання автоматизації створення тестових сценаріїв для перевірки телекомунікаційного програмного забезпечення.

Прикладними завданнями роботи є: планування робіт, аналіз основної діяльності підприємства із забезпечення якості програмних продуктів, аналіз особливостей тестування, аналіз технологій для генерації тестових сценаріїв, вибір альтернативного механізму, розробка і впровадження інформаційної технології створення тестових сценаріїв.

Проведення аналізу діяльності підприємства дало змогу зробити основні висновки стосовно його основної проблематики у напрямку формулювання тестових сценаріїв для ефективної діяльності із забезпечення якості програмних продуктів. Виявлено протиріччя між потребами практики і відсутністю реалізації моделей автоматичної генерації тестових сценаріїв, що повинні забезпечувати діяльність операторів-тестувальників. Тому, основним завданням роботи стало проектування технології автоматизації розробки тестових сценаріїв для перевірки якості програмного забезпечення.

Для вибору оптимального інструментарію було проаналізовано математичний апарат, що стосується зв’язаності поведінкових одиниць на графах. Було докладно описано матричний метод знаходження шляхів у графах, що очікувано найкраще підходить для формалізації і інтерпретації за допомогою використання інформаційних технологій.

Спочатку для вирішення поставленого завдання було проаналізовано можливі підходи до складання тестових сценаріїв:

1. Технологія «High Level TC».
2. Технологія «Handmade Graph».
3. Технологія «ATSG».

Сформовано множину критеріїв вибору базової технології генерації сценарію, що є основою для проведення тестування. Проведено оцінку альтернатив із залученням в якості експертів фахівців підприємства «NetCracker». Проведено розрахунки інтегрованого показника якості технології та вибрано оптимальну технологію генерації тестових сценаріїв.

Для розрахунків використовувався метод аналізу ієрархій і програмне забезпечення СППР «Вибір». В якості базової обрана технологія «ATSG».

Для генерації варіантів тестових сценаріїв програмне забезпечення описане у вигляді графової моделі, що підтверджує наукову новизну даної роботи. Завдання генерації тестових маршрутів зводиться до задачі пошуку маршрутів в графі.

Функціонал розробленої інформаційної системи передбачає реалізацію наступних завдань:

1. Підготовка специфікаційє

2. Подання моделі програми з використанням технології yEd Graph.

3. Генерація варіантів за допомогою програми TCGenerator.

4. Формування документа Excel, що містить інструкції для оператора тестувальника.

Заснована на принципах автоматизованої генерації тестових маршрутів технологія забезпечує істотне зниження трудомісткості тестування програм підтримки телекомунікаційних операторів, що оптимізує основні процеси підприємства з тестування і прямопропорційно впливає на рівень якості програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NetCracker Official Site сайт [Електронний ресурс] – Режим доступу – http://www.netcracker.com/en/
2. Burnstein. Practical software testing : a process-oriented approach. / Burnstein, Ilene – Springer-Verlag New York, Inc. – 2003. – v. 161. – No. 4098. – P. 777-778.
3. Utting. Practical model-based testing : a tools approach./ Utting, Mark – Elsevier Inc. – 2007. – v. 52. – No. 10. – P. 1123-1130.
4. Брауде Э. Технология разработки программного обеспечения / Пер. с англ. – Спб.: ПИТЕР, 2004. – 655 с.
5. Винниченко И. В. Автоматизация процессов тестирования. – СПб.: Питер, 2005. – 203 с.
6. Дастин Э. Автоматизированное тестирование программного обеспечения./ Рэшка Дж., Пол Дж. – Издательство ‘ЛОРИ’, Москва, 2003. – 538 с.
7. Канер Сэм и др. Тестирование программного обеспечения. Фундаментальные последовательности менеджмента бизнес-приложений./Пер. с англ. – К.: Издательство «ДиаСофт», 2001. – 544 с.
8. Макгрегор Д. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие./Пер. с англ./ Сайкс Д. – К.: ООО «ТИД «ДС», 2002. – 432 с.
9. Орлов С. Технологии разработки программного обеспечения. – Спб.: ПИТЕР, 2002. – 464 с.
10. Савин Р. Тестирование Дот Ком, или Пособие по жесткому обращению с багами в интернет-стартапах. – М.: Дело, 2007. – 312 с.
11. Соммервилл И. Инженерия программного обеспечения /Пер. с англ./ Минько А. А., Момотюк А. А., Сингаевская Г.И. – М.: ВИЛЬЯМС, 2002. – 624 с.
12. Тамре Л. Введение в тестирование программного обеспечения./ Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 268 с.

13. Хант Э. Программист-прагматик. Путь т подмастерья к мастеру / Томас Д.– Издательство “ЛОРИ”, Москва, 2004.
14. Хорошилов А. В. Спецификация и тестирование систем с асинхронным интерфейсом. – Препринт 12 ИСП РАН, 2006. – 251 с.
15. Берж К. Теория графов и ее приложения. – М.: ИЛ, 1962. – 320с.
16. Емеличев В. А. Лекции по теории графов./ Мельников О. И., Сарванов В. И., Тышкевич Р. И. – М.: Наука, 1990. – 384с.
17. Зыков А. А. Основы теории графов. — М.: «Вузовская книга», 2004. — С. 664.
18. Кирсанов М. Н. Графы в Maple. – М.: Физматлит, 2007. – 168 с.
19. Кормен Т. Х. и др. Часть VI. Алгоритмы для работы с графами – М.: Вильямс, 2006. — С. 1296.
20. Оре О. Теория графов. — М.: Наука, 1980. — С. 336.
21. Свами М. Графы, сети и алгоритмы./ Тхулалираман К. – М: Мир, 1984. – 455с.
22. Татт У. Теория графов./Пер. с англ. – М.: Мир, 1988. – 424 с.
23. Уилсон Р. Введение в теорию графов./Пер с англ. – М.: Мир, 1977. – 208с.
24. yEd Graph Official Site сайт [Електронний ресурс] – Режим доступу – http://www.yworks.com/en/
25. ППП СППР «Выбор» сайт [Електронний ресурс] – Режим доступу – http://www.sppr.pp.ua/
26. GraphEd сайт [Електронний ресурс] – Режим доступу – http://www.cs.sunysb.edu/~algorith/implement/graphed/ implement.shtml
27. GRph INterface (GRIN) сайт [Електронний ресурс] – Режим доступу – http://graph-software.narod.ru/
28. Microsoft Vision Professional сайт [Електронний ресурс] – Режим доступу – http://office.microsoft.com/en-us/visio/

Додаток А. Код програмного продукту

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import jxl.Workbook;
import jxl.write.Label;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;
import jxl.write.biff.RowsExceededException;

public class Excel
{
    public static void writeExcel(List<GNode> nodes, List<GEdge>
edges, List<GPath> paths, String filename) throws IOException,
RowsExceededException, WriteException
    {
        WritableWorkbook workbook = Workbook.createWorkbook(new
File(filename));
        for (int i = 0; i < paths.size(); i++)
        {
            WritableSheet sheet = workbook.createSheet("Path # "
+ (i+1), i);
            Excel.writeRow(sheet, "#", "Test Case Name",
"Input/Action", "Expected Results", 0);

            List<Integer> path = paths.get(i).getList();

            for (int j = path.size()-1, k = 0; j >= 0; j--, k++)
            {
                if(j == path.size()-1)
                {
                    Excel.writeRow(sheet, new
Integer(k+1).toString(), "", "", getNodeLabel(nodes,
path.get(j).intValue()), k+1);
                }
                else
                {
                    String edge = Excel.getEdgeLabel(edges,
path.get(j+1).intValue(), path.get(j).intValue());
                    Excel.writeRow(sheet, new
Integer(k+1).toString(), edge, edge, getNodeLabel(nodes,
path.get(j).intValue()), k+1);
                }
            }
        }

        workbook.write();
        workbook.close();
    }
```

```
    private static void writeRow(WritableSheet sheet, String
num, String name, String ia, String er, int row) throws
RowsExceededException, WriteException
    {
        Label _num = new Label(0, row, num);
        sheet.addCell(_num);
        Label _name = new Label(1, row, name);
        sheet.addCell(_name);
        Label _ia = new Label(2, row, ia);
        sheet.addCell(_ia);
        Label _er = new Label(3, row, er);
        sheet.addCell(_er);
    }

    private static String getNodeLabel(List<GNode> nodes, int
id)
    {
        String nid = new
StringBuffer("n").append(id).toString();
        for (int i = 0; i < nodes.size(); i++)
        {
            if(nodes.get(i).getID().equals(nid))
            {
                return nodes.get(i).getLabel();
            }
        }
        return "-";
    }

    private static String getEdgeLabel(List<GEdge> edges,int
sid, int tid)
    {
        String esid = new
StringBuffer("n").append(sid).toString();
        String etid = new
StringBuffer("n").append(tid).toString();
        for (int i = 0; i < edges.size(); i++)
        {
            if(edges.get(i).getTarget().equals(etid)
                && edges.get(i).getSource().equals(esid))
            {
                return edges.get(i).getLabel();
            }
        }
        return "-";
    }
}
```

```
import java.util.LinkedList;
import java.util.List;

public class GPath
{
    private List<Integer> path = new LinkedList<Integer>();

    public GPath() {}

    public void add(int val)
    {
        path.add(new Integer(val));
    }

    public int getLast()
    {
        return path.get(path.size()-1);
    }

    public List<Integer> getList()
    {
        return this.path;
    }

    public Exist alreadyExist(int val)
    {
        Exist rezult = new Exist();
        if(path.size() < 1)
        {
            rezult.exist = false;
            return rezult;
        }

        for (int i = 0; i < path.size()-1; i++)
        {
            if(val == path.get(i).intValue())
            {
                rezult.exist = true;
                rezult.position.add(new Integer(i));
            }
        }
        return rezult;
    }

    public int getNext(int position)
    {
        return path.get(position+1).intValue();
    }

    public String toString()
    {
```

```
        StringBuffer sb = new StringBuffer();
        for (int i = path.size()-1; i >= 0; i--)
        {
            sb.append(path.get(i));
            if(i != 0)
            {
                sb.append(" > ");
            }
        }
        return sb.toString();
    }

    public GPath copy()
    {
        GPath new_path = new GPath();
        for (int i = 0; i < path.size(); i++)
        {
            new_path.add(path.get(i));
        }
        return new_path;
    }
}
```

```
import java.util.LinkedList;
import java.util.List;

class Exist
{
    public boolean exist = false;
    public List<Integer> position = new LinkedList<Integer>();
}
```

```
class GNode
{
    private String id;
    private String label;

    public GNode(String id, String label)
```

```
    {
        this.id = id;
        this.label = label;
    }

    public String getID()
    {
        return id;
    }

    public String getLabel()
    {
        return label;
    }

    public String toString()
    {
        return "NodeID = " + id + "; Label = " + label;
    }
}
```

```
class GEdge
{
    private String id;
    private String label;
    private String source;
    private String target;

    public GEdge(String id, String label, String source, String
target)
    {
        this.id = id;
        this.label = label;
        this.source = source;
        this.target = target;
    }

    public String getID()
    {
        return id;
    }

    public String getLabel()
    {
        return label;
    }
```

```
    public String getSource()
    {
        return source;
    }

    public String getTarget()
    {
        return target;
    }

    public String toString()
    {
        return "EdgeID = " + getID() + "; Label = " + getLabel()
+ "; Source = " + getSource() + "; Target = " + getTarget();
    }
}
```

```
import java.util.LinkedList;
import java.util.List;

public class GMatrix
{
    private int[][] Matrix;
    private int size;
    private int _start = -1;
    private List<Integer> _end;

    public GMatrix(int size)
    {
        Matrix = new int[size][size];
        this.size = size;
    }

    public void set(int x, int y)
    {
        Matrix[x][y] = 1;
    }

    public boolean get(int x, int y)
    {
        return Matrix[x][y] == 1;
    }

    public int[][] getMatrix()
```

```
    {
        return this.Matrix;
    }

    public String toString()
    {
        StringBuffer sb = new StringBuffer("MATRIX:
").append('\n');
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                sb.append(Matrix[i][j]);
            }
            sb.append('\n');
        }
        return sb.toString();
    }

    private void calcStart()
    {
        int start = 0;
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                start += Matrix[j][i];
            }
            if (start == 0)
            {
                this._start = i;
            }
            start = 0;
        }
    }

    public int getStart()
    {
        if (this._start == -1)
        {
            calcStart();
        }
        return this._start;
    }

    private void calcEnd()
    {
        this._end = new LinkedList<Integer>();
        int end = 0;
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
```

```
                end += Matrix[i][j];
            }
            if (end == 0)
            {
                this._end.add(i);
            }
            end = 0;
        }
    }

    public List<Integer> getEnd()
    {
        if (this._end == null)
        {
            calcEnd();
        }
        return this._end;
    }

    public List<Integer> getPrevious(int column)
    {
        List<Integer> previous;
        previous = new LinkedList<Integer>();

        for (int i = 0; i < size; i++)
        {
            if(Matrix[i][column] == 1)
            {
                previous.add(i);
            }
        }
        return previous;
    }
}
```

```
import java.io.IOException;
import java.util.LinkedList;
import java.util.List;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
```

```
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class GRAPHMLParser
{
    private String fileName;
    private boolean parsed;

    private LinkedList<GNode> nodes;
    private LinkedList<GEdge> edges;

    private GMatrix Matrix;

    public GRAPHMLParser(String file)
    {
        fileName = file;
        parsed = false;
    }

    public void doParse() throws ParserConfigurationException,
SAXException, IOException
    {
        nodes = new LinkedList<GNode>();
        edges = new LinkedList<GEdge>();

        DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Document doc = builder.parse(fileName);
        NodeList nodesList = doc.getElementsByTagName("node");
        for (int i = 0; i < nodesList.getLength(); i++)
        {
            Node node = nodesList.item(i);
            NamedNodeMap attrs = node.getAttributes();
            String nodeID =
attrs.getNamedItem("id").getTextContent();

            Document childTree = node.getOwnerDocument();
            NodeList labels =
childTree.getElementsByTagName("y:NodeLabel");
            Node labelNode = labels.item(i);
            String nodeLabel = labelNode.getTextContent();

            nodes.add(new GNode(nodeID, nodeLabel));
        }

        NodeList edgesList = doc.getElementsByTagName("edge");
        for (int i = 0; i < edgesList.getLength(); i++)
        {
            Node node = edgesList.item(i);
            NamedNodeMap attrs = node.getAttributes();
            String edgeID =
attrs.getNamedItem("id").getTextContent();
```

```
            String source =
attrs.getNamedItem("source").getTextContent();
            String target =
attrs.getNamedItem("target").getTextContent();

            Document childTree = node.getOwnerDocument();
            NodeList labels =
childTree.getElementsByTagName("y:EdgeLabel");
            Node labelNode = labels.item(i);
            String edgeLabel = "empty";
            if(labelNode != null)
                edgeLabel = labelNode.getTextContent();

            edges.add(new GEdge(edgeID, edgeLabel, source,
target));
        }

        buildMatrxix();

        parsed = true;
    }

    public List<GNode> getNodes() throws
ParserConfigurationException, SAXException, IOException
    {
        if(!parsed)
            doParse();
        return nodes;
    }

    public List<GEdge> getEdges() throws
ParserConfigurationException, SAXException, IOException
    {
        if(!parsed)
            doParse();
        return edges;
    }

    private void buildMatrxix() throws
ParserConfigurationException, SAXException, IOException
    {
        Matrix = new GMatrix(this.nodes.size());

        List<GEdge> edges = this.edges;
        for(int i = 0; i < edges.size(); i++)
        {
            String begin = edges.get(i).getSource();
            String end = edges.get(i).getTarget();
            begin = begin.substring(1);
            end = end.substring(1);
            int source = Integer.parseInt(begin);
            int target = Integer.parseInt(end);
            Matrix.set(source, target);
```

```
        }
    }

    public GMatrix getMatxrix()
    {
        return Matrix;
    }
}
```

```
import java.util.LinkedList;
import java.util.List;

public class Algorithm
{
    public static List<GPath> getPaths(GMatrix matrix)
    {
        List<GPath> paths = new LinkedList<GPath>();

        int start = matrix.getStart();
        List<Integer> end = matrix.getEnd();

        for (int i = 0; i < end.size(); i++)
        {
            GPath p = new GPath();
            p.add(end.get(i).intValue());
            paths.add(p);
        }

        while(!pathsComplete(paths, start))
        {
            for (int i = 0; i < paths.size(); i++)
            {
                GPath p = paths.get(i);
                if(pathComplete(p, start)) continue;

                int last = p.getLast();
                List<Integer> possibleValues =
matrix.getPrevious(last);
                Exist ex = p.alreadyExist(last);
                if(ex.exist)
                {
                    List<Integer> values =
Algorithm.copyIntList(possibleValues);
                    for (int j = 0; j < ex.position.size(); j++)
                    {
```

```
                        int next =
p.getNext(ex.position.get(j).intValue());
                        possibleValues.remove(new
Integer(next));
                    }
                    if(!possibleValues.isEmpty())
                    {
                        for (int k = 1; k <
possibleValues.size(); k++)
                        {
                            GPath p_new = (GPath) p.copy();

p_new.add(possibleValues.get(k).intValue());
                            paths.add(p_new);
                        }
                        p.add(possibleValues.get(0).intValue());
                    }
                    else
                    {
                        values.remove(new Integer(last));
                        for (int k = 1; k < values.size(); k++)
                        {
                            GPath p_new = (GPath) p.copy();
                            p_new.add(values.get(k).intValue());
                            paths.add(p_new);
                        }
                        p.add(values.get(0).intValue());
                    }
                }
                else
                {
                    for (int j = 1; j < possibleValues.size();
j++)
                    {
                        GPath p_new = (GPath) p.copy();

p_new.add(possibleValues.get(j).intValue());
                        paths.add(p_new);
                    }
                    p.add(possibleValues.get(0).intValue());
                }
            }
        }

        return paths;
    }

    private static boolean pathComplete(GPath p, int start)
    {
        return p.getLast() == start;
    }
```

```
    private static boolean pathsComplete(List<GPath> paths, int
start)
    {
        for (int i = 0; i < paths.size(); i++)
        {
            if(!Algorithm.pathComplete(paths.get(i), start))
            {
                return false;
            }
        }
        return true;
    }

    private static List<Integer> copyIntList(List<Integer> list)
    {
        List<Integer> l = new LinkedList<Integer>();
        for (int i = 0; i < list.size(); i++)
        {
            l.add(list.get(i));
        }
        return l;
    }
}
```

Додаток Б. Схеми створення тестових сценаріїв з використанням різних технологій

Рисунок Б1 – Схема створення тестового сценарію з використанням технології «High Level TC»

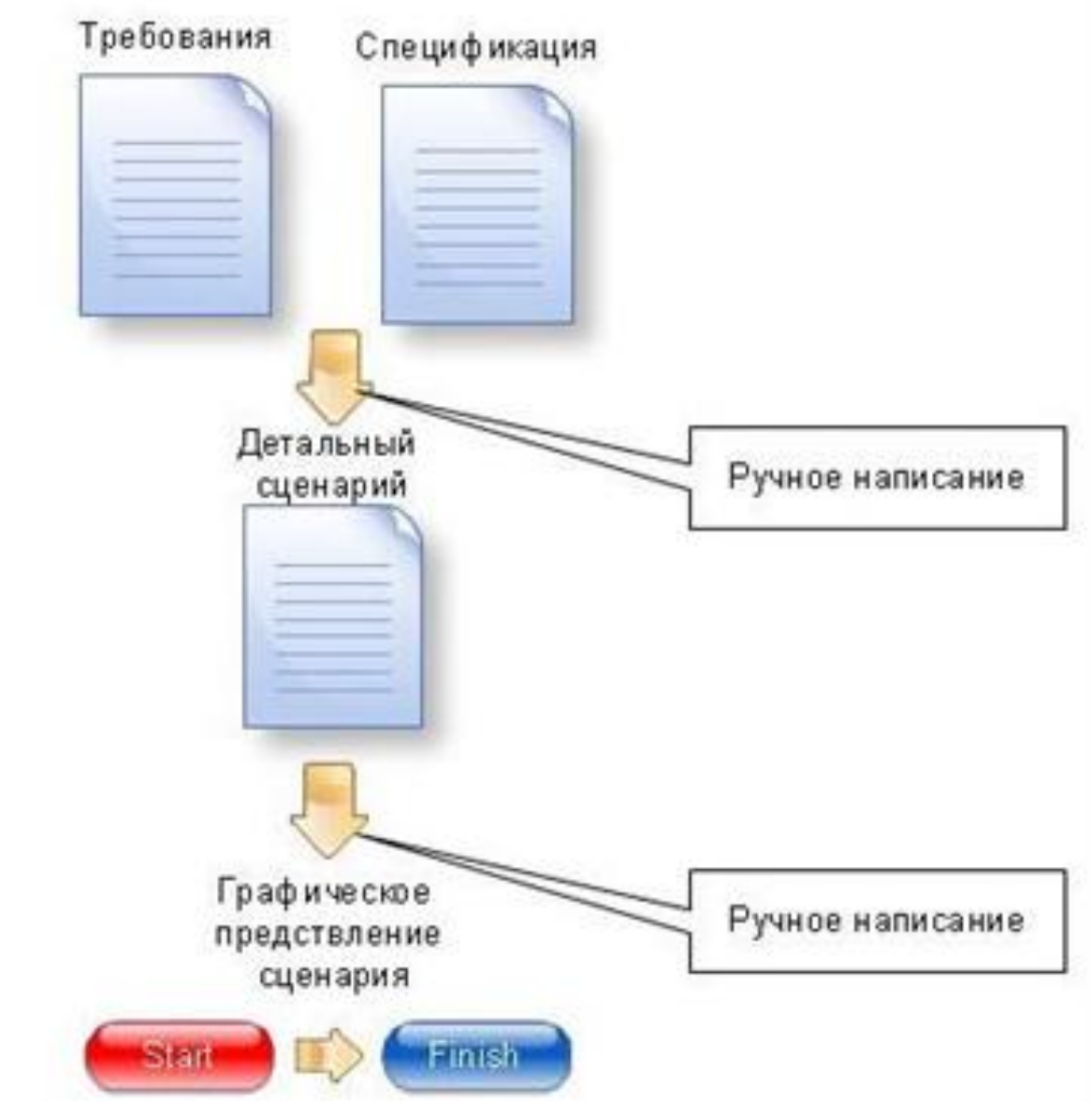

Рисунок Б2- Схема створення тестового сценарію з використанням технології «Handmade Graph»

Рисунок Б3- Схема створення тестового сценарію з використанням технології «ATSD»

Додаток В. Скріншоти роботи ППП СППР «Выбор»

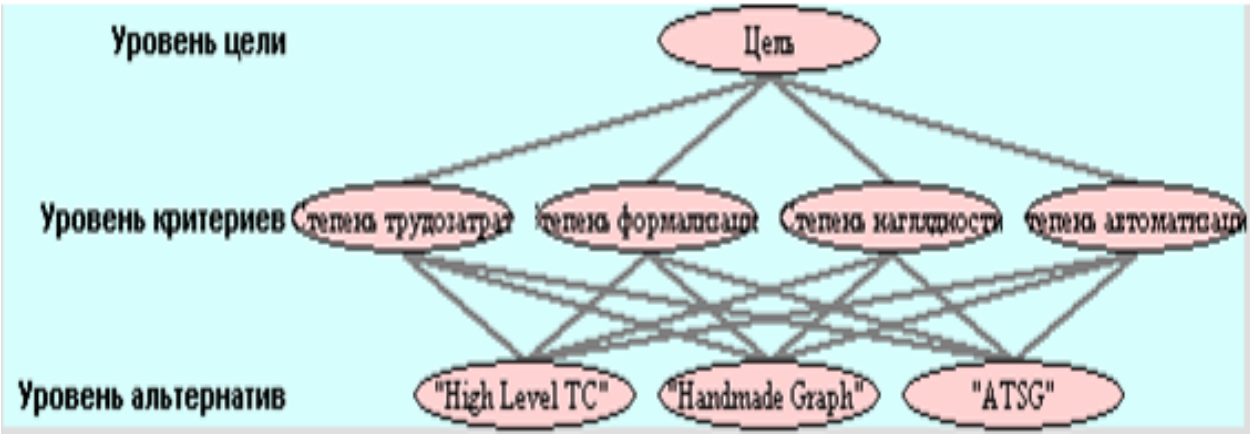

Рисунок В1- Ієрархія цілі, критеріїв, альтернатив

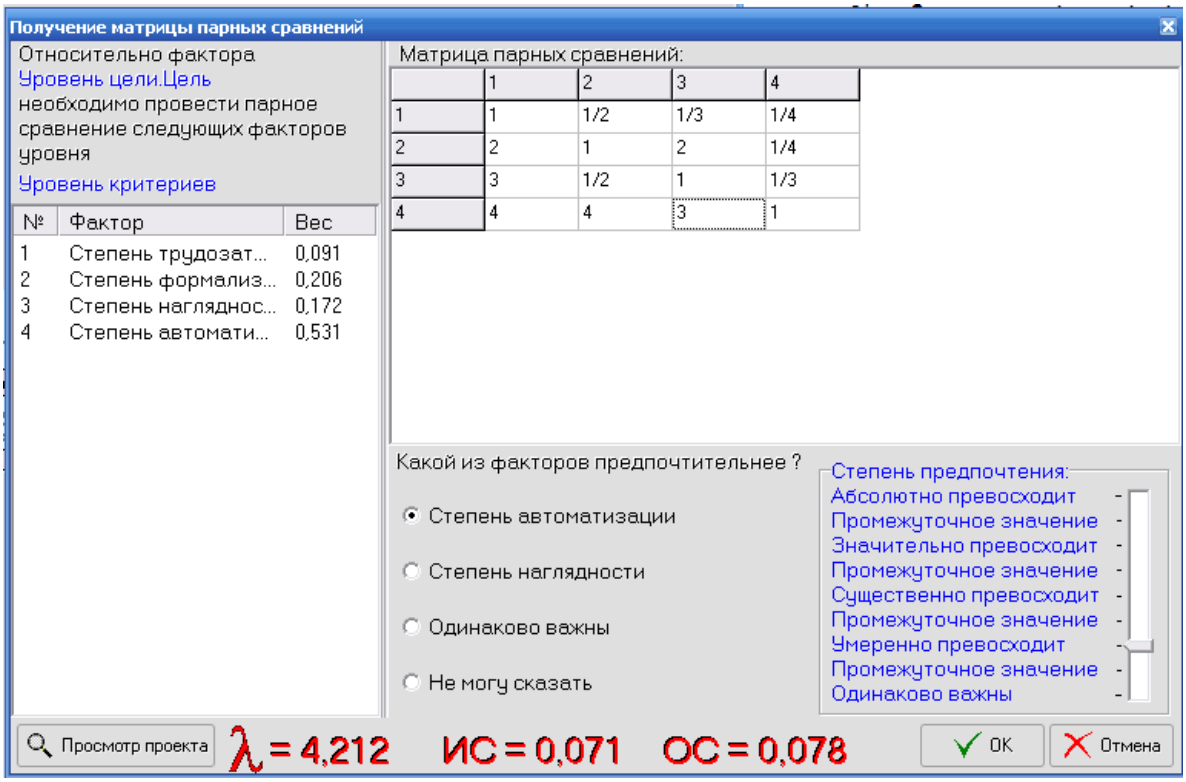

Рисунок В2- Матриця парних порівнянь критеріїв

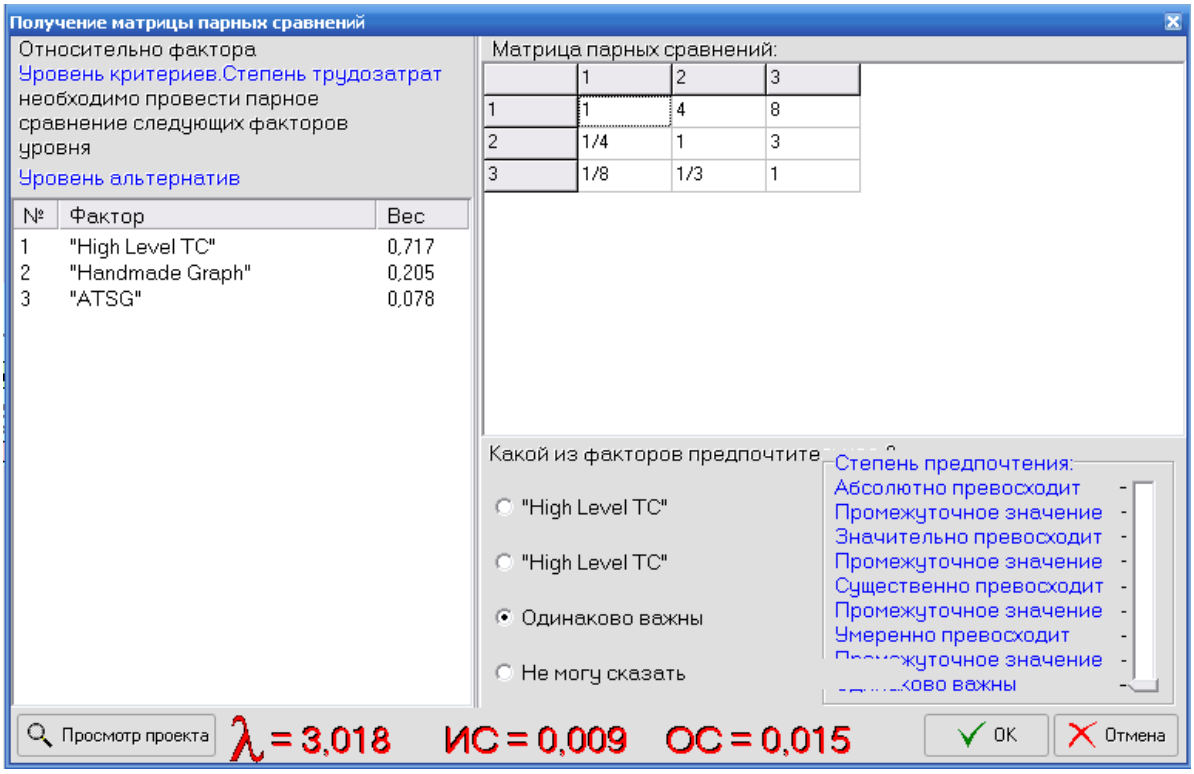

Рисунок В3- Матриця парних порівнянь альтернатив по критерію трудомісткості

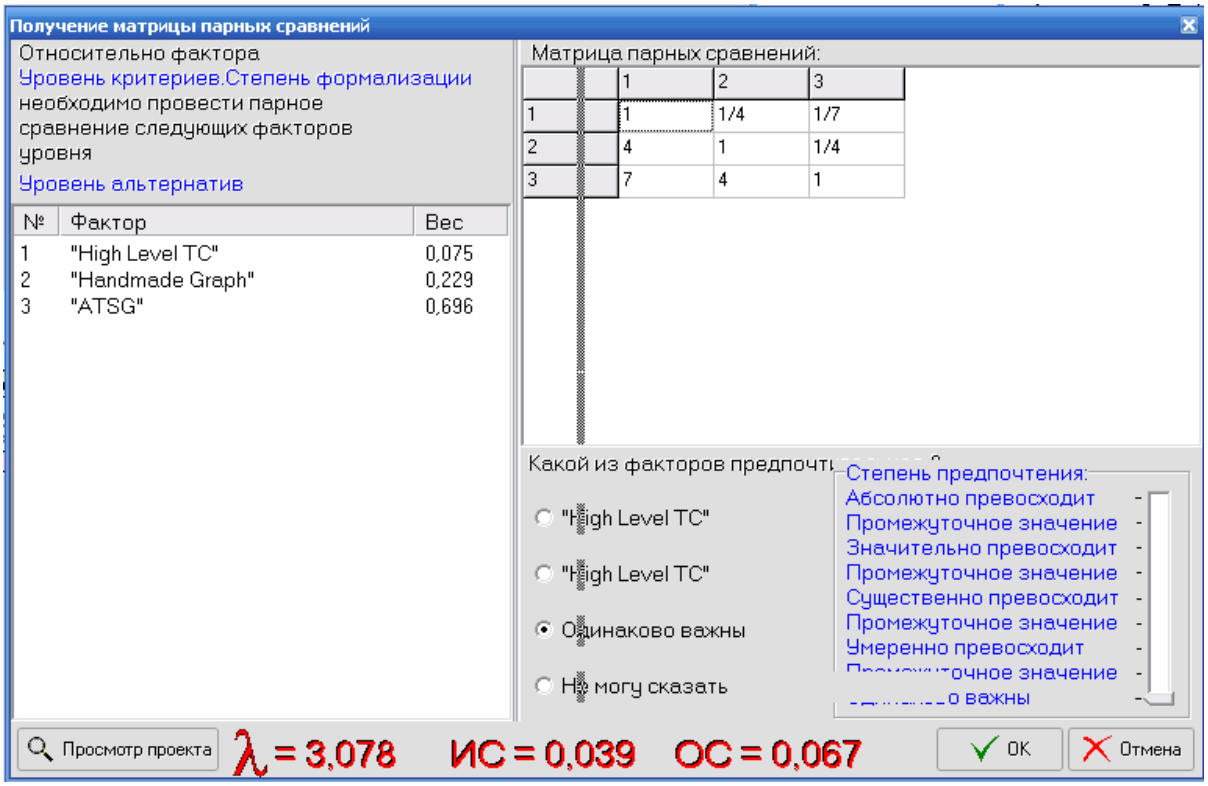

Рисунок В4- Матриця парних порівнянь альтернатив по критерію формалізації

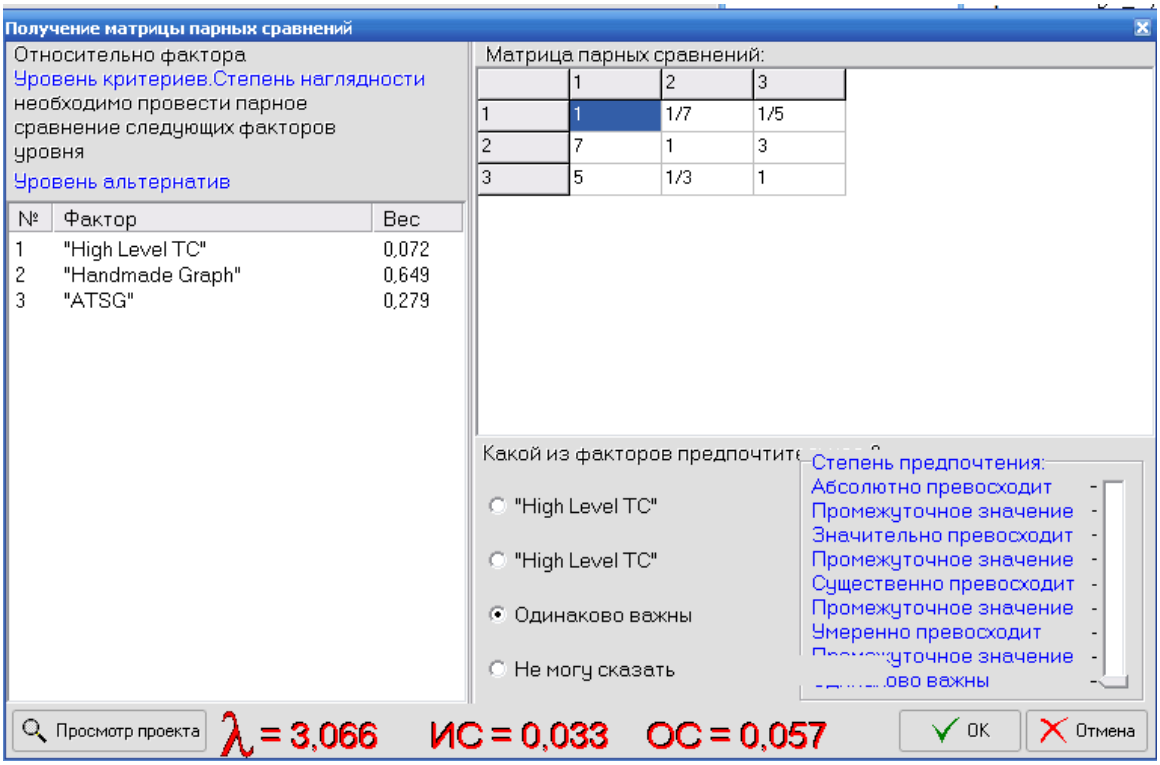

Рисунок В5- Матриця парних порівнянь альтернатив по критерію наочності

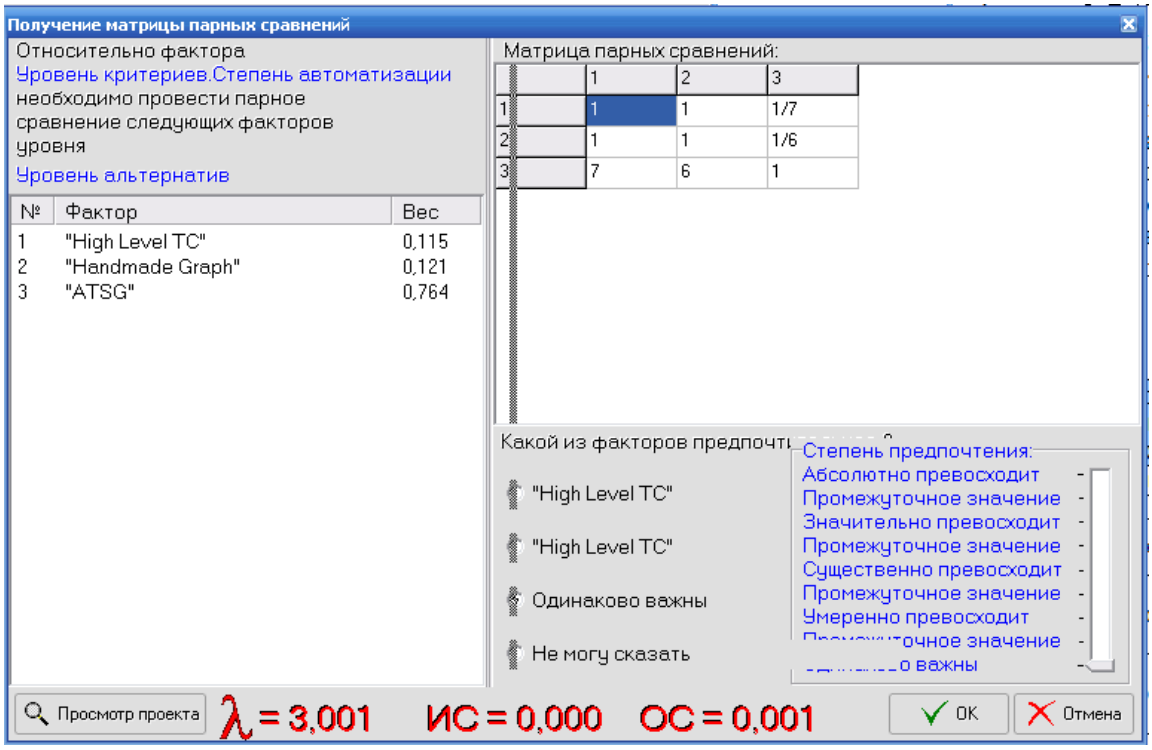

Рисунок В6 – Матриця парних порівнянь альтернатив по критерію автоматизації

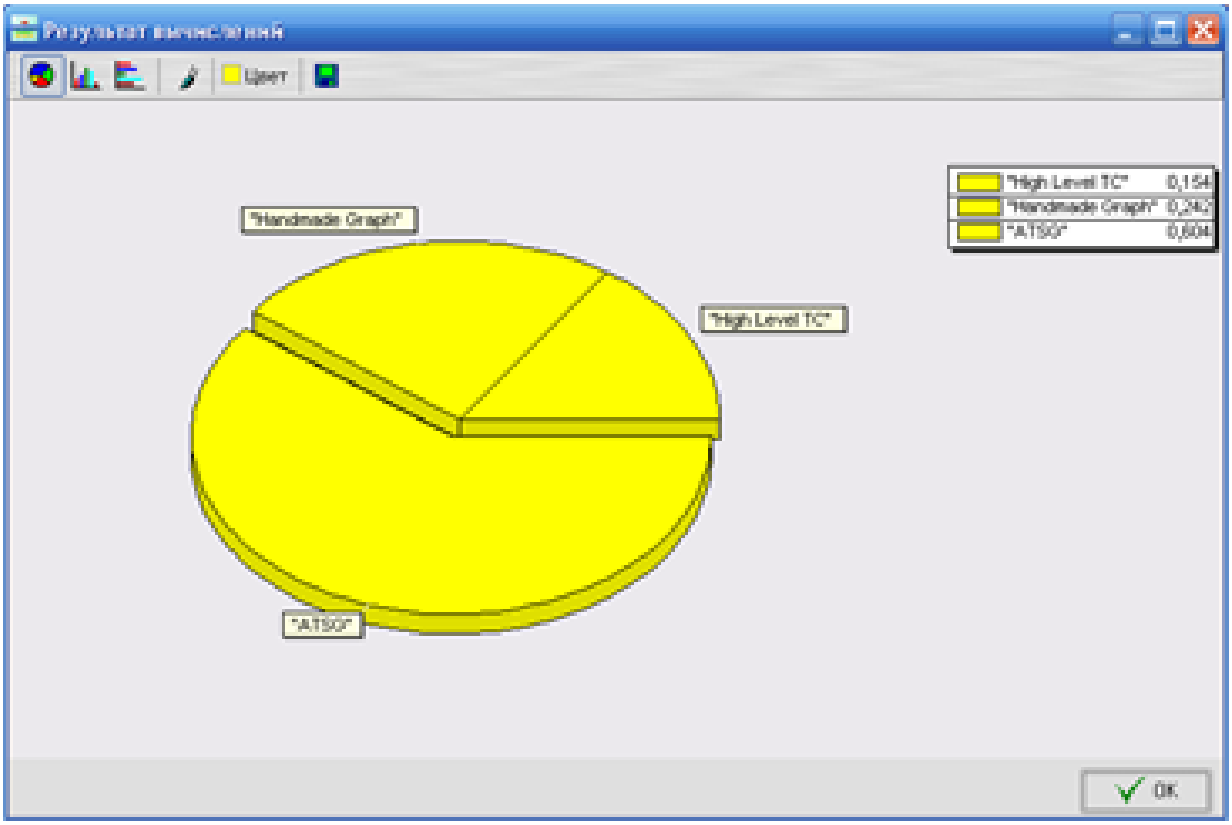

Рисунок В7 – Інтегрована оцінка критеріїв альтернатив

Додаток Г. Схеми алгоритмів пошуку і побудови шляхів

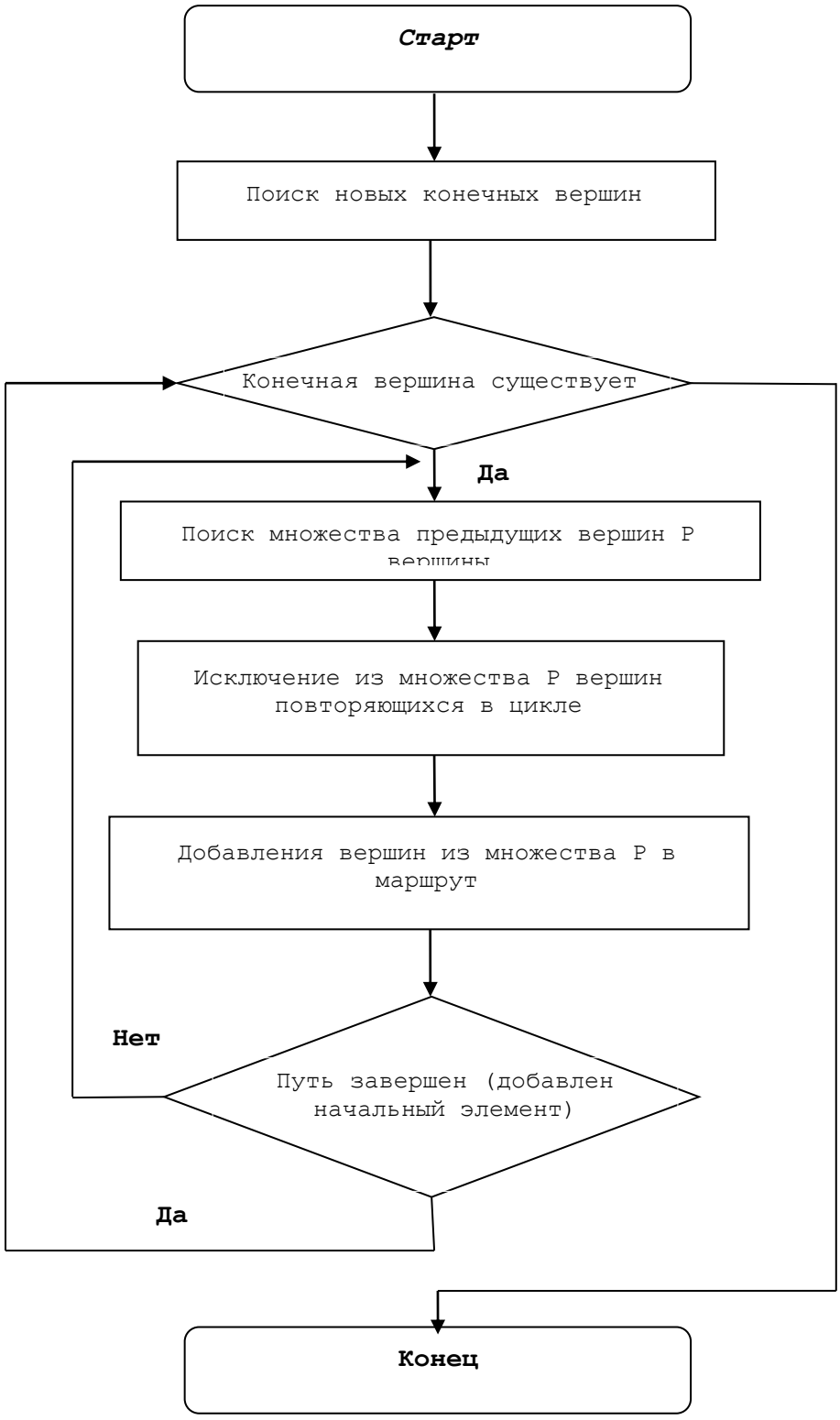

Рисунок Г1- Схема алгоритму пошуку шляхів

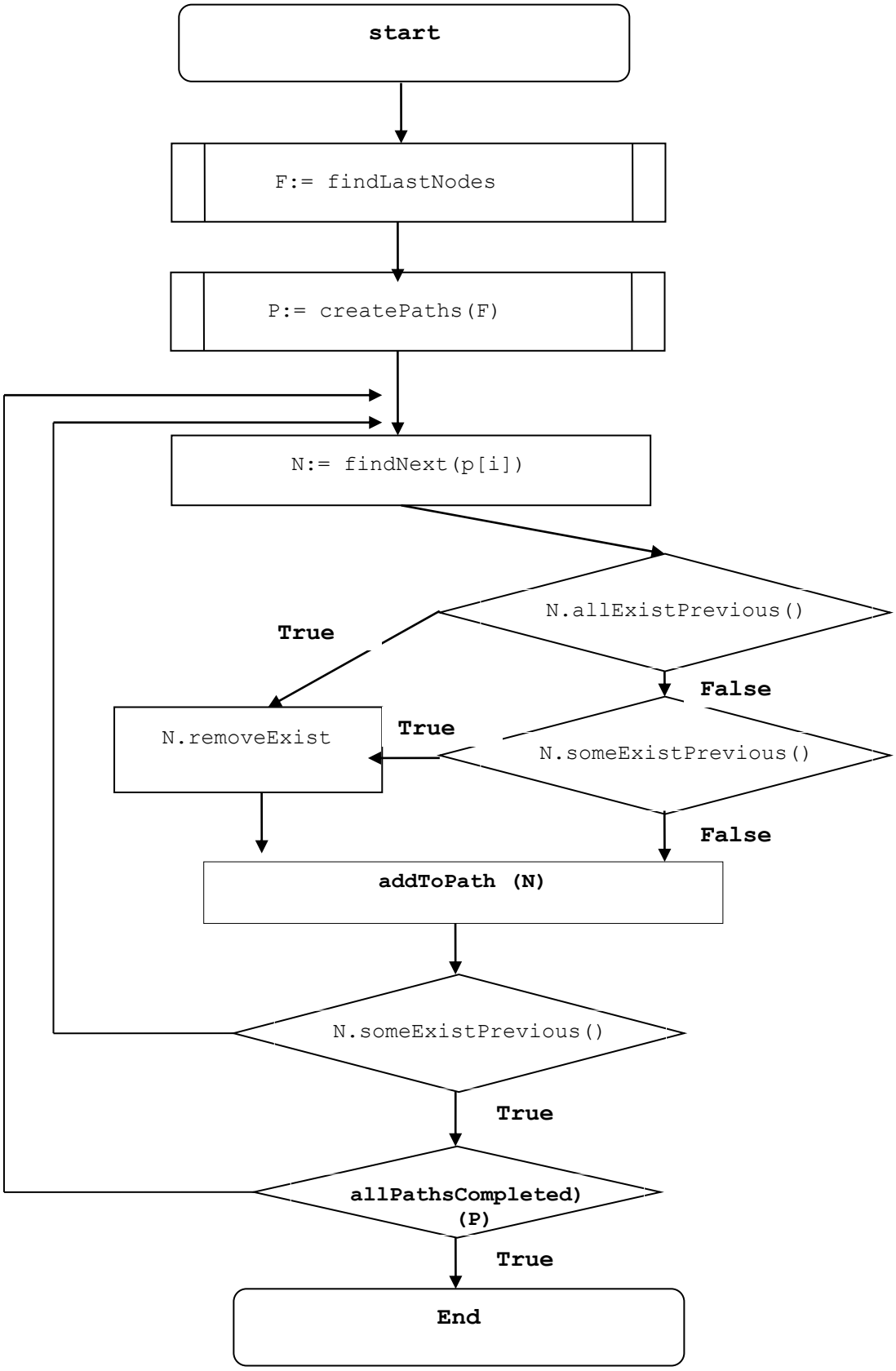

Рисунок Г2- Схема алгоритму побудови шляхів

Додаток Д. Скріншоти формалізації моделі програми в yEd Graph

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
    http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
   <node id="n0"/>
   <node id="n1"/>
   <node id="n2"/>
   <node id="n3"/>
   <node id="n4"/>
   <node id="n5"/>
   <node id="n6"/>
   <node id="n7"/>
   <node id="n8"/>
   <node id="n9"/>
   <node id="n10"/>
   <edge source="n0" target="n2"/>
   <edge source="n1" target="n2"/>
   <edge source="n2" target="n3"/>
   <edge source="n3" target="n5"/>
   <edge source="n3" target="n4"/>
   <edge source="n4" target="n6"/>
   <edge source="n6" target="n5"/>
   <edge source="n5" target="n7"/>
   <edge source="n6" target="n8"/>
   <edge source="n8" target="n7"/>
   <edge source="n8" target="n9"/>
   <edge source="n8" target="n10"/>
  </graph>
</graphml>
```

Рисунок Д1 – Модель графа, що описує програмний продукт мовою GRAPHML

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
    http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">

 ...

</graphml>
```

Рисунок Д2 – Заголовок із посиланням на XML-схему

```
 <graph id="G" edgedefault="directed">
  <node id="n0"/>
  <node id="n1"/>
  ...
  <node id="n10"/>
  <edge source="n0" target="n2"/>
  <edge source="n1" target="n2"/>
  ...
  <edge source="n8" target="n10"/>
 </graph>
```

Рисунок Д3 – Визначення графу

```
 ...
<key id="d0" for="node" attr.name="color" attr.type="string">
  <default>yellow</default>
</key>
<key id="d1" for="edge" attr.name="weight" attr.type="double"/>
<graph id="G" edgedefault="undirected">
  <node id="n0">
    <data key="d0">green</data>
  </node>

  <edge id="e1" source="n0" target="n1">    ...
</graph>  ...
```

Рисунок Д4 – Значення GraphML-атрибута

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file was written by the JAVA GraphML Library.-->
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
                             http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="directed"
         parse.nodes="11" parse.edges="12"
         parse.maxindegree="2" parse.maxoutdegree="3"
         parse.nodeids="canonical" parse.edgeids="free"
         parse.order="nodesfirst">
    <node id="n0" parse.indegree="0" parse.outdegree="1"/>
    <node id="n1" parse.indegree="0" parse.outdegree="1"/>
…..
…..
    <edge id="edge0009" source="n6" target="n8"/>
    <edge id="edge0010" source="n8" target="n7"/>
    <edge id="edge0011" source="n8" target="n9"/>
    <edge id="edge0012" source="n8" target="n10"/>
  </graph>
</graphml>
```

Рисунок Д5 – Граф з додатковою інформацією для парсера

Додаток Ж. Опис основних класів розробленого програмного продукту TCGenerator за допомогою нотації UML

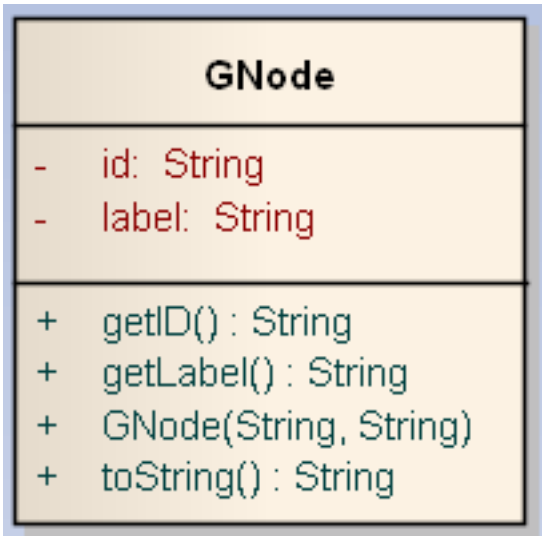

Рисунок Ж1 – Класс GNode

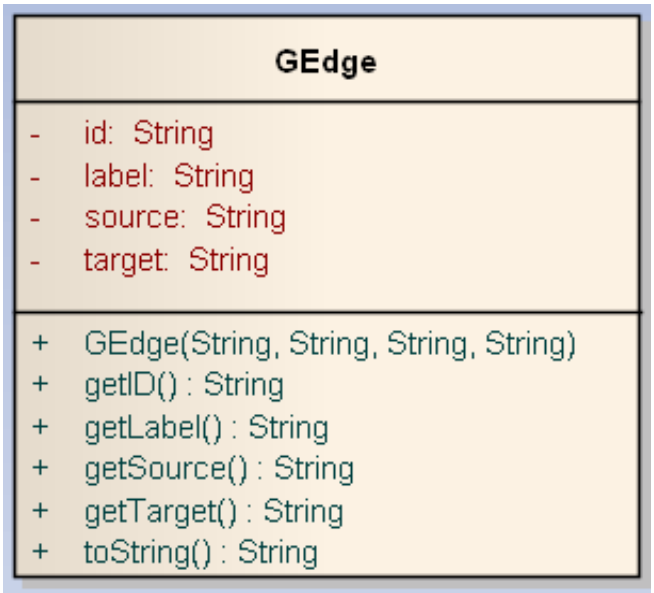

Рисунок Ж2 – Класс GEdge

Рисунок Ж3 – Класс GPath

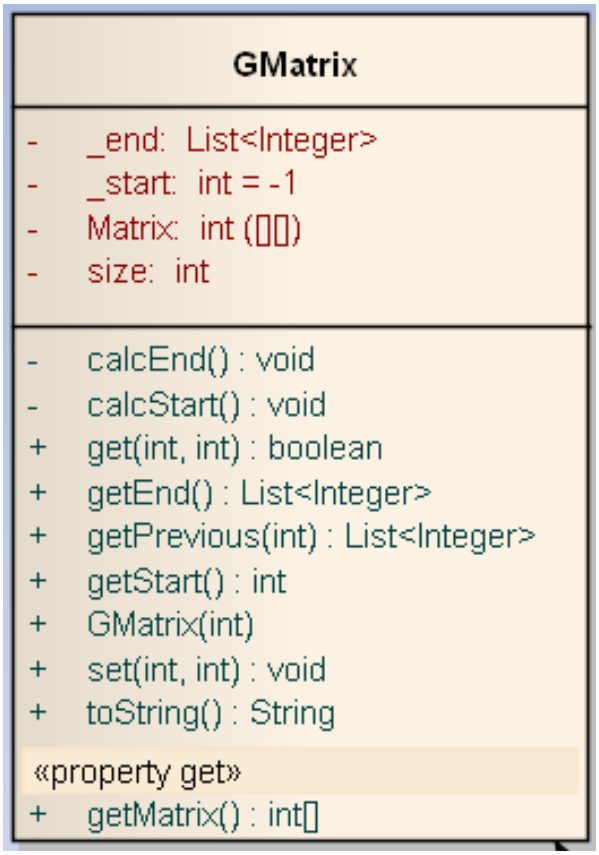

Рисунок Ж4 – Класс GMatrix

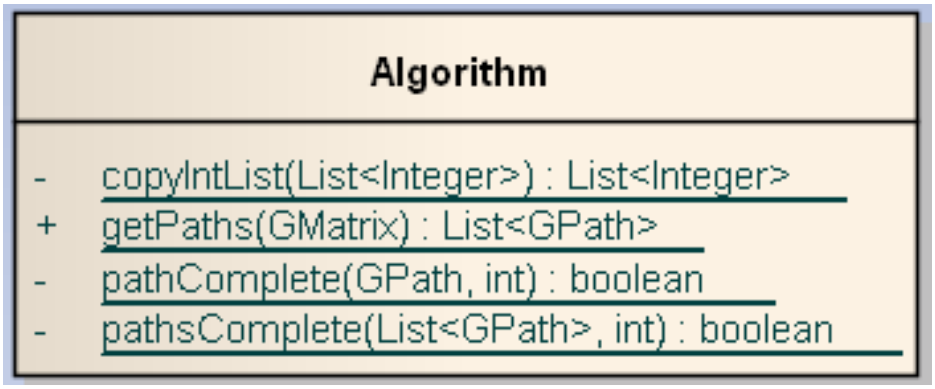

Рисунок Ж5 – Класс Algorithm

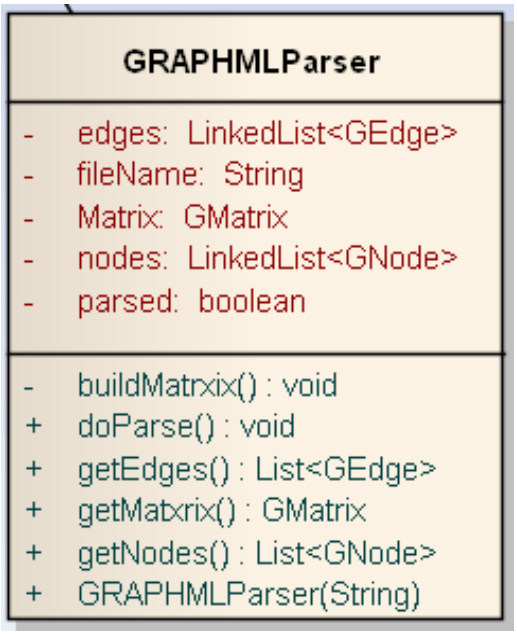

Рисунок Ж6 – Класс GRAPHMLParser

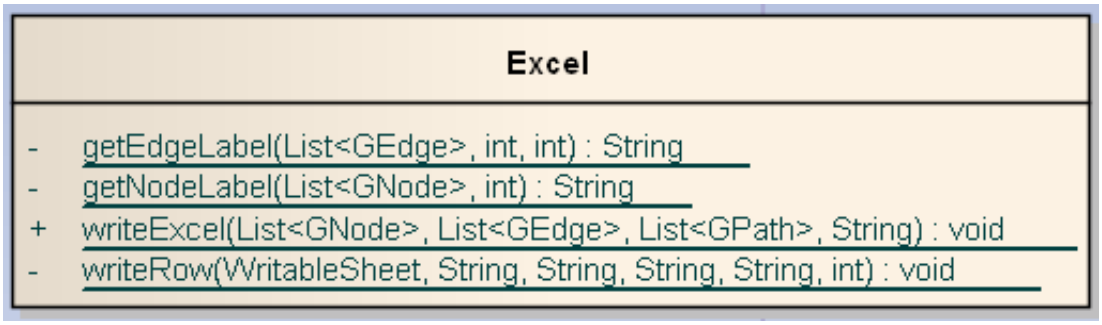

Рисунок Ж7 – Класс Excel