Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Комплекс автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ЗІПЗ–20

_____/ С. В. Подольніков /

Керівник кваліфікаційної роботи _____/ І. А. Котов /

Завідувач кафедри _____/ А. М. Стрюк /

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«____» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ЗІПЗ–20 Подольнікову Сергію Вікторовичу

1. Тема: Комплекс автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов затверджена наказом по КНУ № 279с від «15» квітня 2024р.

2. Термін подання студентом закінченої роботи: «31» травня 2024р.

3. Вихідні дані по роботі: розроблений комплекс має спростити оцінку ризиків та оцінку надійності вузлів та агрегатів систем автомобільного транспорту.

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): проаналізувати існуючі на ринку аналогічні програмні рішення, обґрунтувати необхідні функції розроблюваного додатку, спроектувати додаток, розробити програмне забезпечення, здійснити тестування розробленого додатку.

5. Перелік ілюстративного матеріалу: функціональна схема, блок–схема алгоритму, зображення екранних форм додатку.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Розгляд літературних джерел та пошук інтернет-ресурсів з заданої тематики	14.01.24 – 21.01.24
2	Аналіз існуючих методів вирішення проблеми	22.01.24 – 02.02.24
3	Формулювання актуальності роботи і постановка завдань	03.02.24 – 19.02.24
4	Оформлення матеріалів першого розділу роботи	20.02.24 – 02.03.24
5	Створення функціональної системи та алгоритму додатку	03.03.24 – 19.03.24
6	Оформлення матеріалів другого розділу роботи	20.04.24 – 09.04.24
7	Розробка баз даних, інтерфейсу програмного забезпечення, програмних модулів	10.04.24 – 01.05.24
8	Оформлення додатків	02.05.24 – 07.05.24
9	Тестування розробленої програми	08.05.24 – 16.05.24
10	Оформлення пояснювальної записки	17.05.24 – 30.05.24

Дата видачі завдання: «12» січня 2024 р.

Студент: _____ / С. В. Подольніков /

Керівник роботи: _____ / І. А. Котов /

РЕФЕРАТ

ДЕРЕВО ВІДМОВ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ, СЦЕНАРІЇ ВІДМОВ, РОЗРАХУНОК ПОКАЗНИКІВ НАДІЙНОСТІ, АВТОМОБІЛЬНИЙ ТРАНСПОРТ.

Пояснювальна записка: 92 с., 20 рис., 1 табл., 1 дод., 42 джерела.

Мета кваліфікаційної роботи: розробка комплекса автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов.

Об’єкт проектування: комплекс автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов.

У теоретичній частині роботи виконано аналіз існуючих сьогодні аналогічних програмних рішень. Зазначені сильні та слабкі сторони існуючих програмних продуктів. Обґрунтовані актуальність роботи, мета та сформульовані завдання для комплексу автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов, що створюється.

У практичній частині кваліфікаційної роботи реалізовано функціональну схему розроблюваного продукту та алгоритм його роботи. Розроблено інтерфейс програмного продукту, програмну логіку роботи додатку. Проведено ретельне тестування розробленого програмного забезпечення.

Розроблений комплекс автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов може застосовуватися в машинобудівних компаніях різних масштабів.

ABSTRACT

FAULT TREE, INTELLIGENT ANALYSIS, FAILURE SCENARIOS, CALCULATION OF RELIABILITY INDICATORS, ROAD TRANSPORT.

Explanatory note: 93 p., 20 fig., 1 table, 1 app., 42 references.

The aim of the qualifying work: development of the automated system for assessing the reliability of critical vehicle systems based on intelligent fault tree analysis.

Design object: the complex for automating the reliability assessment of critical vehicle systems based on intelligent fault tree analysis.

In the theoretical part of the work, an analysis of existing similar software solutions in the market has performed. The strengths and weaknesses of existing software products have indicated. The relevance of the work, the purpose and the objectives for the development of the complex for automating the reliability assessment of critical vehicle systems based on intelligent fault tree analysis have formulated.

In the practical part of the qualification work, the functional scheme of the product under development and the algorithm of its work have implemented. The interface of the software product, the program logic of the application have developed. The developed software has tested.

The developed complex for automating the reliability assessment of critical vehicle systems based on intelligent fault tree analysis can be used in machine-building companies of various sizes.

ЗМІСТ

ВСТУП

Безпека і функціональність сучасних автомобілей значною мірою залежать від складних і взаємопов'язаних систем, що забезпечують їх роботу. Теорія надійності відіграє важливу роль у забезпеченні функціонування цих систем за призначенням, мінімізуючи ризик настання відмов. Поточна робота зосереджена на застосуванні теорії надійності, зокрема, через призму використання інтелектуального аналізу дерева відмов [1], для оцінки критично важливих систем транспортних засобів.

Теорія надійності забезпечує основу для розуміння і кількісної оцінки ймовірності того, що система буде виконувати свою функцію протягом певного періоду часу за певних умов. Для автомобілей це означає забезпечення бездоганної роботи таких систем, як гальмівна система, рульове управління та управління двигуном, щоб гарантувати безпеку пасажирів і запобігти виникненню аварій. Однак зростаюча складність сучасних транспортних засобів створює нові виклики. Критично важливі системи автомобіля часто включають в себе складні взаємодії між електронними та механічними компонентами, що робить традиційні методи аналізу відмов менш ефективними.

В роботі пропонується використання інтелектуального аналізу дерева відмов як інструменту для вирішення зазначених проблем. Описаний метод ґрунтується на традиційному аналізі дерева несправностей [2], що дозволяє проводити комплексний і динамічний аналіз потенційних режимів відмов у критично важливих системах автомобілей. Розглядаючи складні взаємодії між компонентами та враховуючи статистичні дані, інтелектуальний аналіз дерева відмов може ідентифікувати та визначати пріоритетність сценаріїв відмов з більшою точністю.

Досліджуючи застосування інтелектуального аналізу дерева відмов до критично важливих систем автомобілей, поточна робота має на меті внести свій вклад у реалізацію комплексу мір, спрямованих на підвищення безпеки та

надійності автомобілей. Це дослідження може бути корисним при проектуванні та розробці майбутніх транспортних засобів, що в кінцевому підсумку призведе до безпечнішого та надійнішого досвіду експлуатації транспортних засобів їх кінцевими користувачами - водіями.

1 АНАЛІЗ МЕТОДІВ І ЗАСОБІВ ОЦІНКИ НАДІЙНОСТІ КРИТИЧНИХ СИСТЕМ АВТОМОБІЛЯ І АКТУАЛЬНІСТЬ ПРОБЛЕМИ ЇХ АВТОМАТИЗАЦІЇ

1.1 Аналіз базових положень загальної теорії надійності технічних систем

Теорія надійності [3,4] являє собою наукову дисципліну, що займається питаннями забезпечення високої надійності технічних виробів за найменших витрат. Теорія надійності охоплює розроблення та вивчення методів забезпечення ефективності роботи об'єктів (виробів, пристроїв, систем тощо) у процесі експлуатації.

Під час розв'язання проблем загальної теорії надійності та окремих її розділів використовують аналітичний апарат і методи таких розділів математики, як: теорія ймовірностей і математична статистика [5], теорія випадкових процесів, стохастичний аналіз [6], чисельні методи, методи моделювання, марковські процеси [7-9].

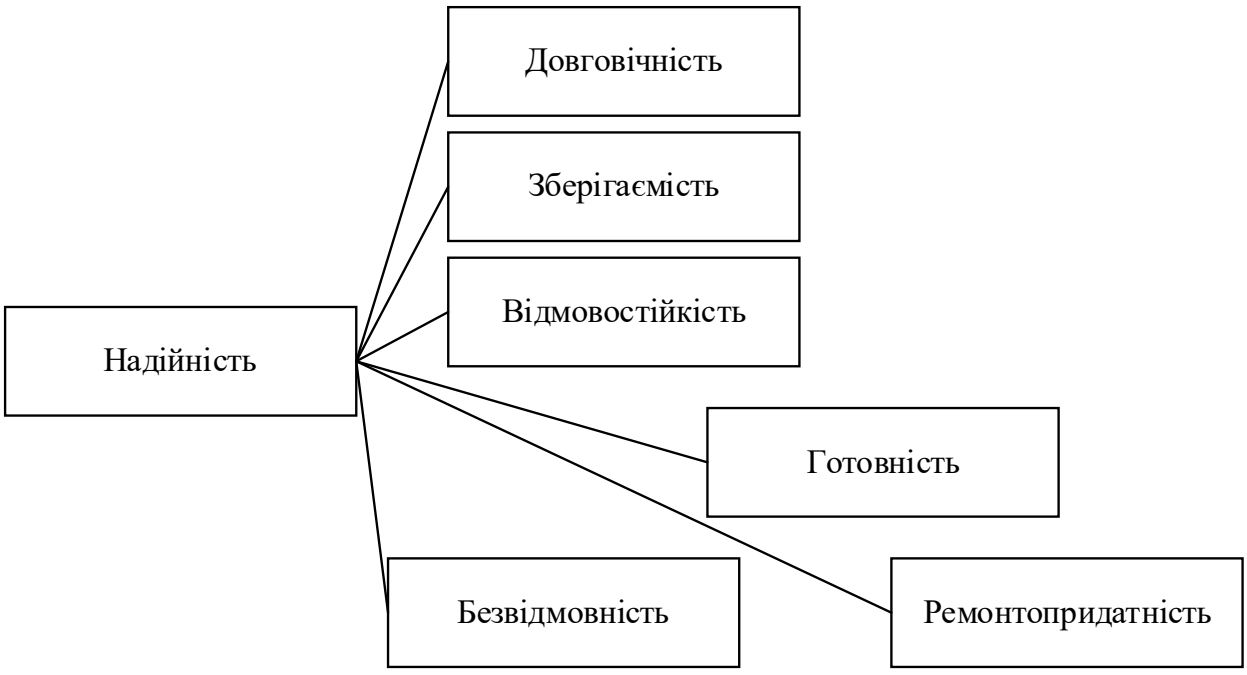

Рисунок 1.1 – Складові надійності

У галузі інженерії поняття надійності займає ключове місце. Воно означає здатність технічної системи стабільно виконувати свої функції за певних умов і протягом заздалегідь визначеного часу. Справжня надійність зазвичай виходить за рамки простої експлуатації - вона вимагає успішної роботи, яка відповідає чітко встановленим критеріям. Хоча кількісні показники, такі як середній час напрацювання на відмову (MTBF) [10], дають цінну інформацію про надійність системи, досвід користувача, який відчуває стабільну роботу системи, має значно більшу вагу. Значення надійності виходить за межі окремих технічних систем, та задіює такі ролі, як інженери, оператори та користувачі:

а) надійні системи слугують підтвердженням правильності дизайнерського вибору інженерів та забезпечують вагомий початковий внесок в успіх проекту;

б) завдяки надійним системам робота їх операторів стає ефективнішою та безпечнішою. Це забезпечує зменшення потреб в технічному обслуговуванні, зниженню рівня небезпеки і створення можливостей для зосередження операторів на своїх основних завданнях;

в) головним чинником для кінцевих користувачів є стабільна робота системи, яка відповідає їхнім очікуванням і дозволяє уникнути незрозумілих реакцій системи на їхні дії. Ненадійні системи не тільки порушують робочі процеси користувачів, але й можуть створювати ризики для їх безпеки.

Вплив надійності виходить за межі вищезазначених ролей, впливаючи на безпеку, економічну ефективність, задоволеність клієнтів і навіть на екологічні міркування:

а) у системах, критично важливих для безпеки, таких як літаки або медичні прилади, надійна робота безпосередньо впливає на безпеку користувачів цих систем. Несправність в такого роду системах може мати катастрофічні наслідки для тих, хто користується ними, що зумовлює першорядну важливість надійності в цих сферах застосування;

б) надійні системи сприяють підвищенню економічної ефективності при вирішенні задач, за для яких вони використовуються. Вони мінімізують витрати на технічне обслуговування за рахунок зменшення частоти ремонтів обладнання та його замін. Крім того, вони зменшують час простою обладнання, що пов'язаний з виникаючими несправностями, сприяючи ефективній роботі в різних галузях промисловості;

в) стабільна продуктивність використовуваних систем формує довіру і задоволеність кінцевих користувачів, що призводить до конкурентної переваги на ринку. Ненадійні системи можуть зашкодити лояльності компанії та репутації клієнтів;

г) в деякій мірі надійність може сприяти зменшенню негативного впливу експлуатації систем на навколишнє середовище. Надійні системи з меншою кількістю відмов продукують менше матеріальних відходів, що виникають при появі необхідності заміни деталей. Крім того, вони вимагають меншого споживання енергії на ремонт і технічне обслуговування.

Охоплюючи як кількісні, так і якісні аспекти, і забезпечуючи свою критичну важливість для різного переліку вищезазначених ролей , надійність утверджується як основоположний принцип в загальній теорії надійності технічних систем. Це багатогранна концепція, яка впливає не лише на технічні характеристики системи, але й на безпеку, вартість, досвід користувача та екологічні міркування.

Таким чином, надійність представляє собою властивість об'єкта до збереження у часі своїх параметрів, що визначають здатність забезпечення виконання необхідних функцій у відповідних режимах в умовах експлуатації, виконання технічного обслуговування, транспортування та зберігання. Надійність є комплексною властивістю, яка є залежною від кейсів застосування системи та умов її використання. Вона може поєднувати низку властивостей, зображених на рисунку 1.2, для зазначеної системи як в цілому, так і для окремих її вузлів чи підсистем [11].

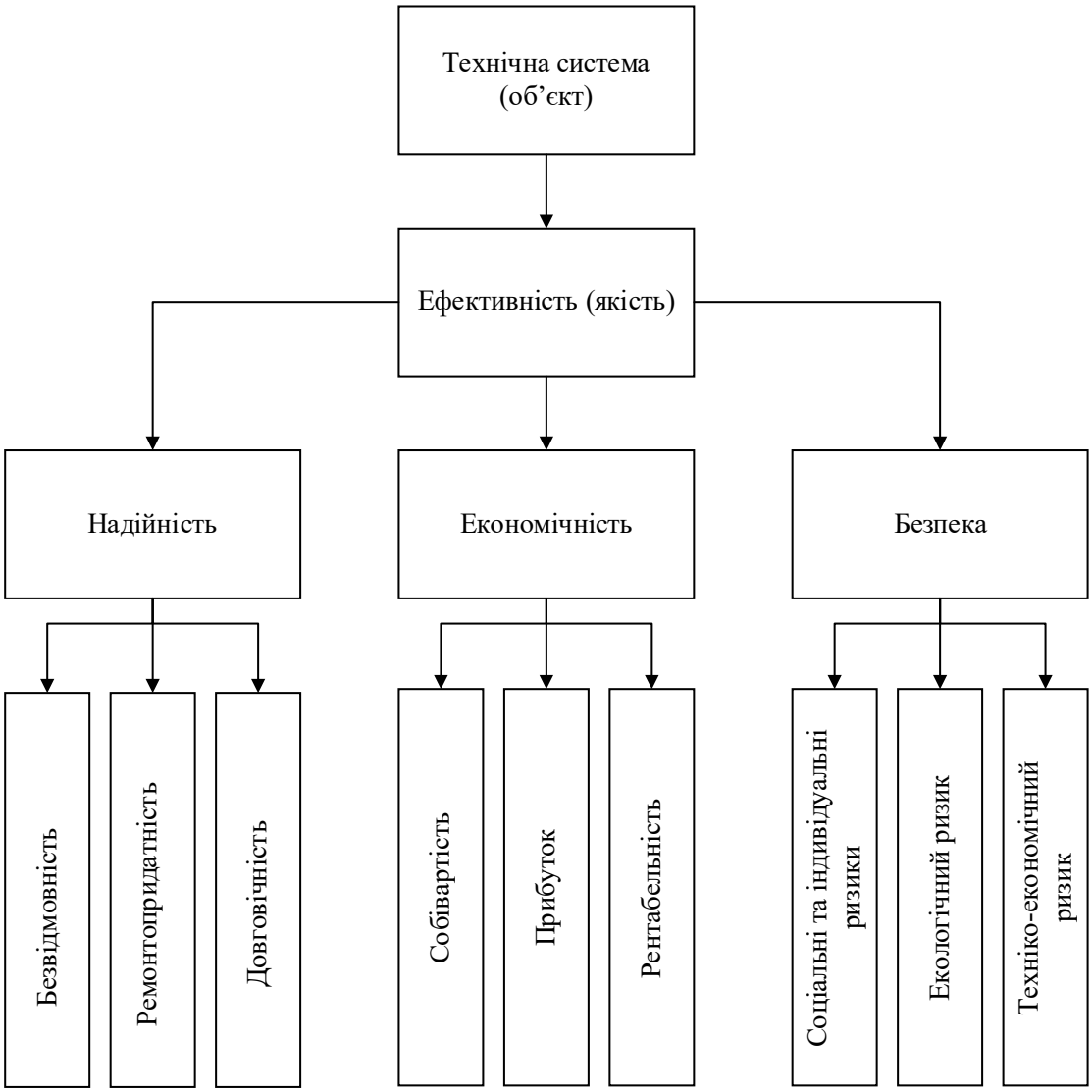

Рисунок 1.2 – Основні властивості технічних систем

Ввідповідно до міжнародного стандарту ISO 9000:2015 [12], в контексті теорії надійності технічних систем можна виділити наступні терміни та означення (таблиця 1.1) [4].

Таблиця 1.1. Терміни та означення в контексті надійності систем

Термін	**Опис**
Працездатність	Стан технічного засобу, при якому він здатний виконувати задані функції з параметрами, встановленими вимогами нормативно-технічної та конструкторсько-технологічної документації

Продовження таблиці 1.1.

Термін	**Опис**
Відмова	Подія, що вказує на порушення роботоздатності технічного засобу
Критерій відмови	Ознака, за якою оцінюється надійність різних технічних засобів
Безвідмовність	Властивість технічних засобів безупинно зберігати роботоздатний стан протягом деякого часу
Напрацювання	Тривалість роботи технічних засобів в годинах, циклах, календарних днях та ін.
Напрацювання до відмови	Напрацювання технічного засобу від початку його експлуатації до виникнення першої відмови
Граничний стан	Стан технічного засобу, при якому його подальше застосування за призначенням стає неприпустимим чи недоцільним
Довговічність	Властивість технічного засобу зберігати роботоздатний стан до настання граничного стану при встановленій системі технічного обслуговування і ремонтів
Ремонтопридатність	Властивість технічного засобу, яка полягає в можливості попередження і виявлення причин виникнення відмов, підтримання і відновлення роботоздатного стану шляхом проведення технічного обслуговування і ремонтів
Збережність	Властивість технічного засобу зберігати значення показників безвідмовності, довговічності і ремонтопридатності протягом експлуатації, зберігання та транспортування

Продовження таблиці 1.1.

Термін	**Опис**
Ресурс	Напрацювання технічного засобу від початку його експлуатації чи відновлення після ремонту до переходу в граничний стан
Термін експлуатації	Календарна тривалість від початку експлуатації технічного засобу чи відновлення після ремонту до переходу в граничний стан
Середній час відновлення	Математичне сподівання часу відновлення роботоздатного стану
Показник надійності	Кількісна характеристика однієї або декількох властивостей, що визначають надійність технічного засобу

При виконанні проектування системи має бути забезпечена відповідність усім заявленим технічним вимогам. Їх можна інтуітивно поділити на головні вимоги, що уможливлюють виконання системою необхідних функцій, та на допоміжні, що в більшій мірі забезпечують зручність використання створюваної системи, ергономіку при її використанні та інші супутні функції. Користаючись поняттями теорії надійності, будь-яку систему або технічний засіб можна описати за його характеристиками, поточним технічним станом, а також наявністю можливостей його відновлення у випадку несправностей.

1.2 Методи оцінки надійності технічних систем на основі аналізу дерева відмов

Аналіз дерева відмов (англ. Fault Tree Analysis), також відомий як аналіз дерева подій - це метод визначення можливих причин відмови системи. В ньому дерево несправностей використовується для графічної ілюстрації

різних потенційних причин збою у вигляді діаграми. За допомогою аналіза дерева відмов можна визначити, які фактори вплинули на подію (відому як відмова), а також ймовірність її виникнення. Після виявлення та усунення основних причин, аналіза дерева відмов може допомогти командам технічного обслуговування визначити пріоритетність коригувальних дій.

Аналіз дерева відмов використовується системними проектувальниками, проектувальниками процесів, менеджерами проектів та інженерами на виробництві. Ці фахівці часто використовують зазначений метод разом з методологією Кайдзен [13] і аналізом першопричин для запобігання або усунення системних збоїв.

Аналіз дерева відмов використовує діаграму аналізу дерева несправностей, щоб показати різні події або умови, які можуть призвести до небажаного результату, наприклад, до відмови обладнання. Процес аналізу складається з трьох основних етапів:

а) створення діаграми дерева несправностей;

б) визначення події відмови, ініціюючої події та факторів, що їх спричиняють;

в) оцінка взаємозв'язку між відмовами і подіями, що їх ініціюють (або факторами, що їх спричиняють).

При цьому символи, що використовуються на діаграмі дерева несправностей, називаються подіями, умовами або станами [14]. Вони можуть виникати в будь-який момент часу під час роботи системи. Лінії з'єднують символи разом, щоб показати, як одна подія може призвести до іншої, поки не дійдено до кінця лінії - небажаної події (тобто несправності). Несправності відображають те, що йде не так у досліджуваній системі. Нижче на рисунку 1.3 наведено приклад того, як можуть виглядати ці діаграми.

У зазначеному прикладі діаграми дерева несправностей наведено ілюстрацію відсутності потоку в насосі або двигуні. Ця подія є основною несправністю, а під нею демонструються ініціюючі події: механічна і електрична несправність. Праворуч від діаграми дерева несправностей, під

електричною несправністю, наведені інші події та несправності, одна з яких - несправність двигуна, а інша - несправність запобіжника. Під відмовою запобіжника показано, що відбувається подія перевантаження ланцюга, а під нею - дві різні основні події: коротке замикання дроту та/або стрибок напруги.

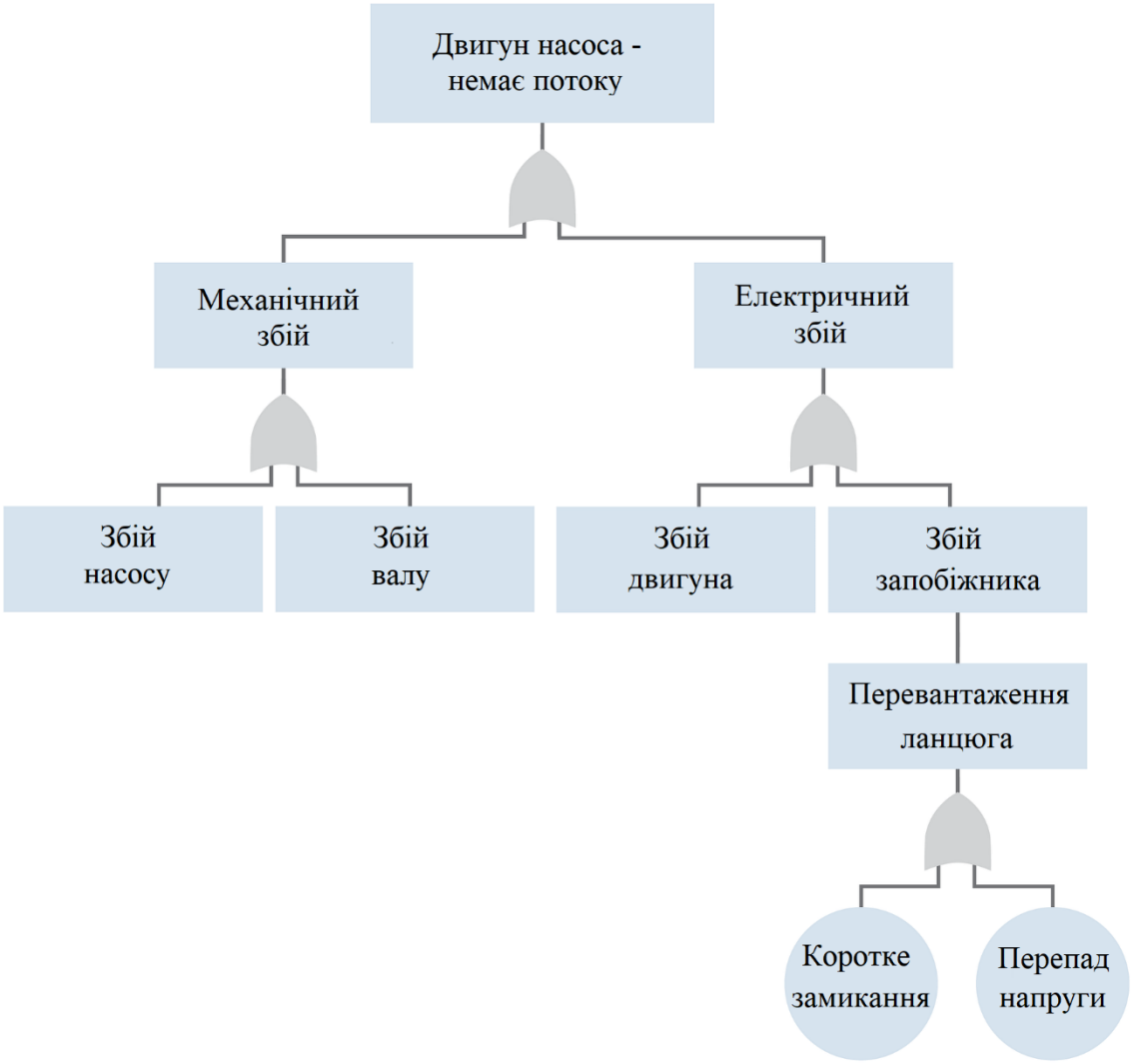

Рисунок 1.3 – Приклад діаграми дерева несправностей

Кожна галузь використовує однаковий набір символів і умовних позначень для дерев несправностей. Дерево несправностей ілюструє зв'язок і потік між різними видами діяльності і читається зверху вниз. Події та шлюзи (відомі як ворота) - це дві категорії, на які поділяються види діяльності. Події

виникають, коли система або процес виходить з ладу. Типи подій, які з'являються на деревах несправностей, наведені на рисунку 1.4.

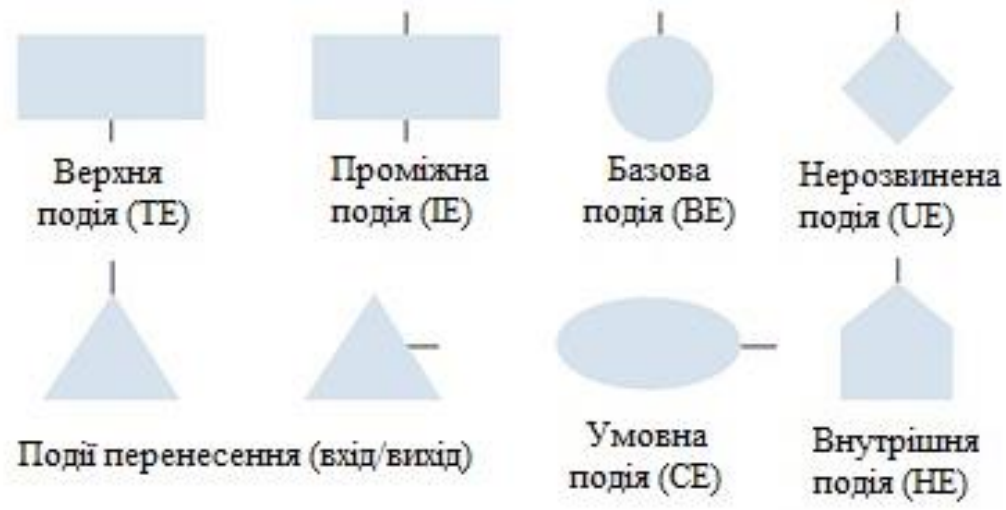

Рисунок 1.4 – Типи подій на деревах несправностей

Нижче наведено визначення кожного типу подій, проілюстрованих вище:

а) Верхня подія (TE): тип події, що знаходяться у верхній частині дерева несправностей і спонукають до розслідування несправності системи. Вона має єдиний вхід, але не має відносних виходів, оскільки є початком несправності;

б) Проміжні події (IE). Ці події, як правило, спричинені однією або кількома подіями. Вони мають як вхід, так і вихід. Інша подія може спричинити її відмову і, найімовірніше, спричинить подальші відмови вниз по дереву несправностей;

в) Базові події (BE): Ці типи подій, як правило, є першопричиною головної події. Вони знаходяться внизу дерева несправностей;

д) Нерозвинені події (UE): Ці події не мають достатньо інформації і розміщуються в піддереві;

е) Події перенесення (TE): Цей тип подій трапляється, коли дерево несправностей занадто довге, щоб помістится на папері. Більші частини дерева ховаються за допомогою відповідного символу і розгортаються в

окремому дереві. Існує два їх типи - події передачі-виведення та передачі-введення. Події передачі-виведення мають трикутник і вихід праворуч, а події передачі-введення мають вхід у верхній частині трикутника;

ж) Умовні події (CE) - ці події визначаються як умови для типу воріт, які називаються забороняючими воротами;

і) Внутрішні події (HE). Цей тип подій використовується для вмикання та вимикання події. Якщо подію встановлено на 0, це означає, що вона не відбудеться, а якщо на 1 - то це означає, що вона відбудеться. Внутрішні події використовуються для того, щоб дозволити включення або виключення частин дерева несправностей.

Вентилі (шлюзи, ворота), у свою чергу, представляють різні способи, якими можуть відбуватися збої в ресурсі або системі. Іноді одна подія може спричинити відмову найвищого рівня (або катастрофічну відмову). Іноді комбінація різних подій може спричинити відмову найвищого рівня. Типи вентилів наведені на рисунку 1.5.

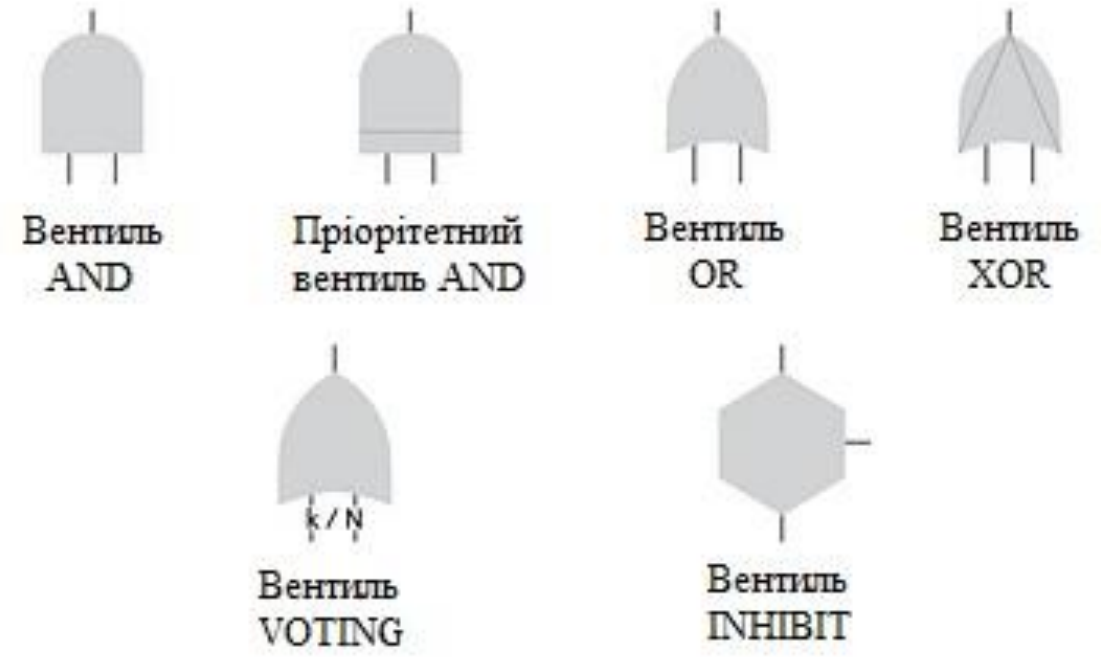

Рисунок 1.5 – Типи вентилів на діаграмі несправностей

Нижче наведено визначення кожного типу вентилів, проілюстрованого вище:

а) Вентиль AND – зазначений тип шлюзу пов'язаний з вихідними подіями. Події відбуваються лише тоді, коли відбуваються вхідні події для вентилю;

б) Пріоритетний вентиль AND - спрацьовує, якщо всі вхідні події відбуваються в певному порядку;

в) Вентиль OR - цей тип вентилів може мати один або декілька входів, а вихідна подія відбудеться, якщо відбудеться одна або декілька вхідних подій;

д) Вентиль XOR зазначений тип вентилів зустрічається трохи рідше. Вихід відбувається лише тоді, коли відбувається один вхідний елемент;

е) Вентиль VOTING - візуально цей тип вентилів схожий на OR. Містить декілька вхідних подій "N" і одну вихідну подію "k". Вихідна подія відбувається, коли відбувається певна кількість вхідних подій. Для спрацьовування цього типу вентилів необхідна точна кількість вхідних подій.

ж) Вентиль INHIBIT - цей тип вентилів матиме вихідну подію, коли відбудуться всі вхідні та умовні події.

Що стосується переваг використання аналізу дерева несправностей, то зазначений метод є низхідним методом, який можна використовувати для аналізу впливу однієї несправності на систему. Нижче наведені деякі інші переваги використання аналізу дерева несправностей:

а) Він звужує причину події збою, що економить ваш час і гроші на пошук першопричини;

б) Він визначає способи пом'якшення наслідків збою ще до того, як він станеться. Наприклад, при проектуванні пропелера літака можна використовувати аналіз дерева несправностей щоб визначити, що станеться, якщо пропелер зламається, і як його можна відремонтувати. Використовуючи цю інформацію, можна з'ясувати, як запобігти цим поломкам у першу чергу;

в) Він допомагає визначити, які поломки є найбільш ймовірними, що дозволить зосередити свої зусилля на запобіганні цим несправностям проектованої системи;

д) Він визначає загальні режими відмов у різних системах або продуктах (наприклад, схожих компонентах), що може допомогти визначити, де зміни в дизайні системи є найбільш необхідними.

Аналіз дерева несправностей допомагає запобігти потенційним збоям і вирішити їх. Зазначений метод пошуку несправностей може використовуватися командами для аналізу збоїв у складних системах. Він допомагає визначити причину збою та першопричини, щоб запобігти їм у майбутньому. Аналіз дерева несправностей може допомогти визначити пріоритетність проблем, щоб команди могли приймати кращі рішення щодо вдосконалення [15-17] .

1.3 Аналіз програмних засобів оцінки надійності технічних систем на основі аналізу дерева відмов

1.3.1 TopEvent FTA

Сьогодні на ринку програмного забезпечення існує низка рішень для оцінки надійності технічних систем. Вони можуть бути застосовані в тому числі і для розрахунку надійності критичних вузлів транспортних засобів. Зазначені програмні рішення мають свої переваги та недоліки, реалізують цілий спектр функціональних можливостей, орієнтовані на різні платформи та мають відмінності в плані інтерфейсу. Декілька найбільш використовуваних програмних продуктів розглянуті далі, де виконано аналіз їх можливостей, зручності роботи з ними, цінової політики та умов використання.

Програма TopEvent FTA [18] є спеціалізованим інструментом для аналізу дерева відмов, яка призначена для інженерів і аналітиків, які займаються аналізом надійності та безпеки складних систем. Відповідно до заяв розробників, TopEvent FTA – це інтерактивне програмне забезпечення для якісного та кількісного аналізу дерев несправностей, що підтримує як

когерентні, так і некогерентні дерева. TopEvent FTA включає два методи оцінки дерева несправностей: класичний метод мінімальних розрізів і метод бінарних діаграм прийняття рішень.

Зовнішній вигляд додатку TopEvent FTA наведено на рисунку 1.6.

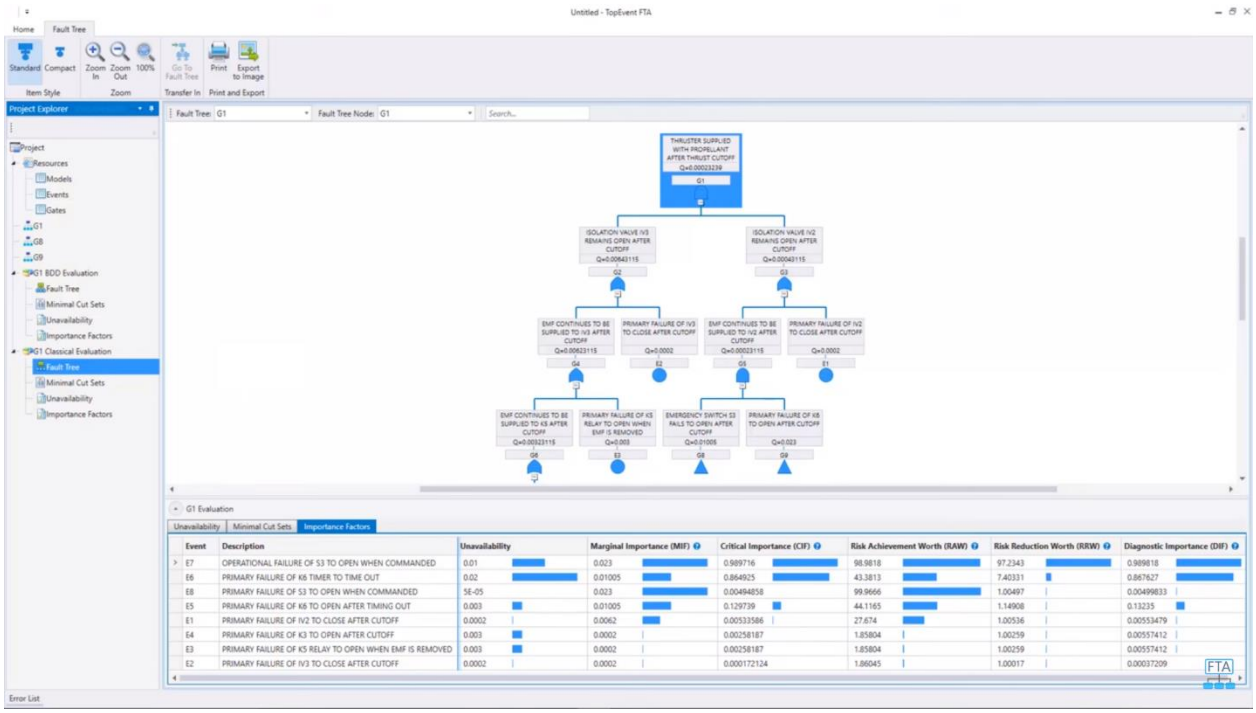

Рисунок 1.6 – Додаток TopEvent FTA

Зазначене програмне забезпечення дозволяє будувати дерева відмов шляхом використання графічних діаграм, на яких комбінується взаємозв’язок логічних вентилів для моделювання різних комбінацій відмов, несправностей, помилок і нормальних подій, що призводять до виникнення певної небажаної ситуації. Як зазначається на офіційному сайті компанії, TopEvent FTA в основному використовується в галузі інженерії безпеки та надійності, щоб зрозуміти, як системи можуть вийти з ладу, визначити найкращі способи зниження ризику або визначити частоту подій, які можуть призвести до нещасного випадку. Результати роботи додатку можуть бути використані в аерокосмічній галузі, ядерній енергетиці, хімічній, переробній, фармацевтичній, нафтохімічній та інших галузях з високим рівнем небезпеки.

В той же час, модель дерева відмов може бути перетворена в математичну модель для обчислення ймовірності відмов і міри важливості системи. Дерево несправностей може моделювати всі аспекти системи, включаючи обладнання, програмне забезпечення, дії людини і навколишнє середовище.

Робота з цим компонентом в додатку TopEvent FTA дійсно є зручною, через те, що:

а) дерево відмов явно показує всі різні взаємозв'язки, які необхідні для того, щоб призвести до головної події;

б) при побудові дерева відмов досягається глибоке розуміння логіки і основних причин, що призводять до головної події;

в) дерево відмов є матеріальним записом систематичного аналізу логіки і основних причин, що призводять до головної події;

д) дерево відмов забезпечує основу для ретельної якісної та кількісної оцінки головної події;

В програмі можна створювати складні дерева несправностей за допомогою редактора дерев відмов. Редактор дерева відмов включає два подання: стандартне подання діаграми дерева відмов і подання списку дерева. Ці подання надають повний огляд дерева відмов. Відповідно, TopEvent FTA дозволяє створювати як когерентні, так і некогерентні дерева відмов.

Програма підтримує основні типи вентилів і подій, що наведені на рисунках 1.4 та 1.5.

За допомогою TopEvent FTA можна швидко оцінити складні дерева несправностей з великою кількістю мінімальних наборів розрізів. Якісну оцінку дерева відмов можна отримати за допомогою класичного методу мінімальних наборів розрізів (Classical MCSs Method) або методу двійкових діаграм прийняття рішень (BDD Method). TopEvent FTA підтримує пороги відсікання, засновані на максимальному порядку мінімальних наборів відсікання (Order cut-off) і мінімальній ймовірності мінімальних наборів відсікання (Probability cut-off).

Зазначений інструментарій наведено на рисунку 1.7.

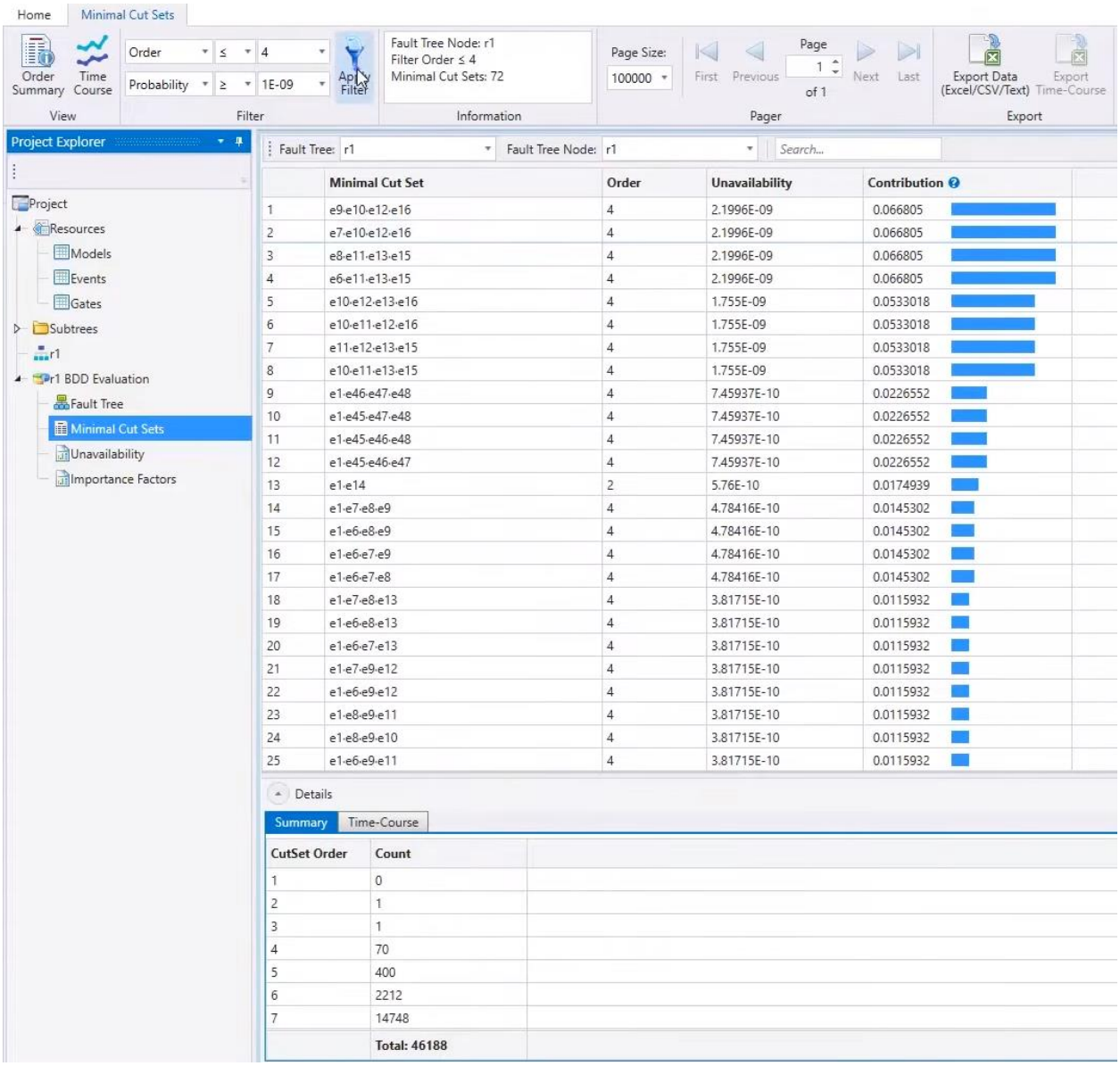

Рисунок 1.7 – Додаток TopEvent FTA: оцінка складних дерев відмов

Варто зазначити, що метод мінімальних шляхів відмов (Classical MCSs Method) [19] є однією з ключових технік в аналізі дерева відмов. Його використовують для виявлення мінімальних комбінацій базових подій, які можуть призвести до верхівкової події. Розуміння та використання методу даного дає змогу проводити якісний аналіз системи, допомагаючи виявити найуразливіші місця та критичні комбінації відмов.

В свою чергу, метод Binary Decision Diagrams [20] є не менш потужним інструментом у зазначеній галузі. Він використовується для представлення та

аналізу булевих функцій, що дає змогу ефективно моделювати та оцінювати складні системи. Binary Decision Diagrams надає компактне та структуроване представлення логічних виразів, що спрощує аналіз і обробку даних.

TopEvent FTA також може розрахувати точні значення коефіцієнтів недоступності та важливості. Кількісна оцінка дерева несправностей, як зазначалося вище, може бути отримана за допомогою класичного методу мінімальних розрізів або методу двійкових діаграм рішень. Метод BD Binary Decision Diagrams забезпечує точні значення коефіцієнтів недоступності та важливості, а мінімальних шляхів відмов забезпечує апроксимацію ймовірностей рідкісних подій.

TopEvent FTA надає методи для розрахунку мінімальних наборів скорочень, доступності, відсутності, надійності, безумовної інтенсивності відмов та інших засобів оцінки дерева відмов.

Що стосується вартості використання TopEvent FTA, пропонується чотири тарифних плани – Express, Starter, Standard та Professional [21]. Перший пропонується безкоштовно, в тім він обмежений можливістю застосування максимум 30 елементів дерева відмов, можливістю використання лише методу мінімальних шляхів відмов, в той час як Binary Decision Diagrams не є доступним. Серед засобів оцінки дерева відмов є доступними лише методи мінімальних наборів скорочень та недоступності.

Наступні три тарифні плани коштують відповідно $450, $950, $3500 на місяць і забезпечують використання обох методів - Binary Decision Diagrams та методу мінімальних шляхів відмов, а також повний набір засобів оцінки дерева відмов, що включає в себе перелік з декількох десятків параметрів. Зазначені тарифні плані відрізняються один від одного лише максимальною кількістю застосовуваних елементів дерева несправностей, ця кількість відповідно складає 100, 300 елементів та безкінечну величину (у плані Professional).

Стосовно підтримки української мови в зазначеному програмному забезпеченні не повідомляється.

1.3.2 Reliability Workbench

Reliability Workbench - це комплексне програмне забезпечення для аналізу надійності та управління ризиками, розроблене компанією Isograph. Воно надає широкий спектр інструментів для моделювання, аналізу та поліпшення надійності складних систем [22]. Як зазначається на офіційному сайті компанії, додаток дозволяє проводити точне прогнозування надійності системи, застосовувати високоінтегровані дані і методи, та проводити аналіз безпеки на рівні підприємства.

Зовнішній вигляд програмного забезпечення Reliability Workbench при реалізації дерева відмов зображено на рисунку 1.8.

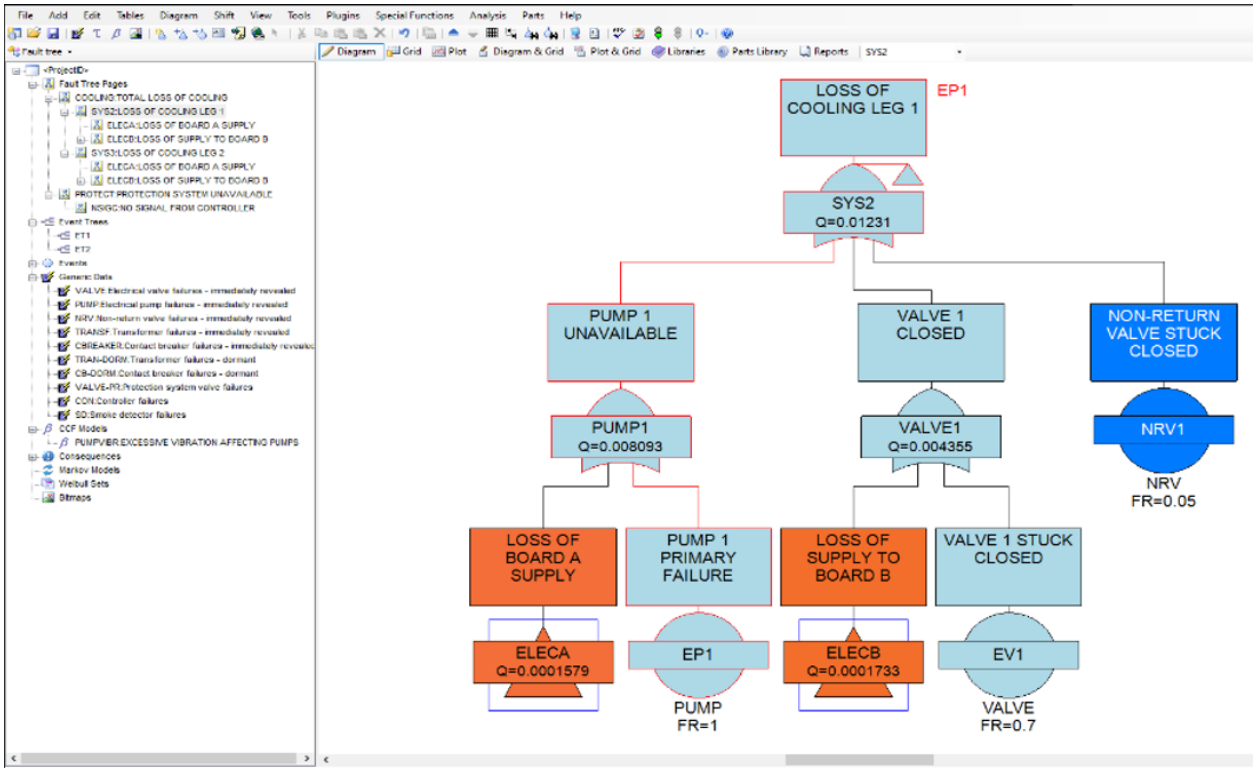

Рисунок 1.8 – Програмне забезпечення Reliability Workbench – побудова дерева відмов

Серед переваг використання Reliability Workbench можна виділити наступне:

а) Аналіз дерева відмов, що включає в себе побудову та аналіз зазначених дерев, якісний і кількісний аналіз з використанням мінімальних шляхів відмов і ймовірностей відмов;

б) Аналіз видів і наслідків відмов (FMEA/FMECA) [23,24], що реалізує складання та управління таблицями, оцінювання критичності відмов і розроблення заходів щодо їх запобігання;

в) робота з блок-схемами надійності (RBD) [25], яка поєднує в собі моделювання системи з використанням блок-схем для аналізу надійності, обчислення показників надійності та доступності системи;

д) аналіз і моделювання подій (ETA) [26] - побудову та аналіз дерев подій для оцінювання ймовірності різних результатів після початкової події;

е) прогнозування надійності з використанням декількох стандартів для прогнозування надійності компонентів і систем;

ж) аналіз даних щодо відмов для визначення закономірностей і тенденцій.

Також Reliability Workbench підтримує безліч модулів, одним з яких є реалізація підтримки марківського аналізу. Це дозволяє використовувати програму для створення моделей, поділу аналізу на окремі етапи, редагування атрибутів стану за допомогою простих і діалогових вікон, верифікації даних для перевірки узгодженості, моделювання залежних від часу перехідних процесів, розрахунку широкого діапазону ймовірностей і частот, і головне - повну інтеграцію з деревом несправностей і аналізом блок-схем надійності [27].

Проведення марківського аналізу шляхом використання Reliability Workbench наведене на рисунку 1.9.

Що стосується вартості використання Reliability Workbench, то на офіційному сайті взагалі відсутня інформація стосовно варіантів купівлі або тарифних планів для підписки. В часто задаваних питаннях фігурує інформація стосовно того, що для отримання вартості необхідно зконтактувати з менеджерами продажів через електрону пошту. Зазвичай це є

надійним індикатором того, що вартість продукту перевищує середню вартість серед конкурентів. В тім, в неофіційних джерелах повідомляється, що програма розповсюджується за методом щомісячної підписки.

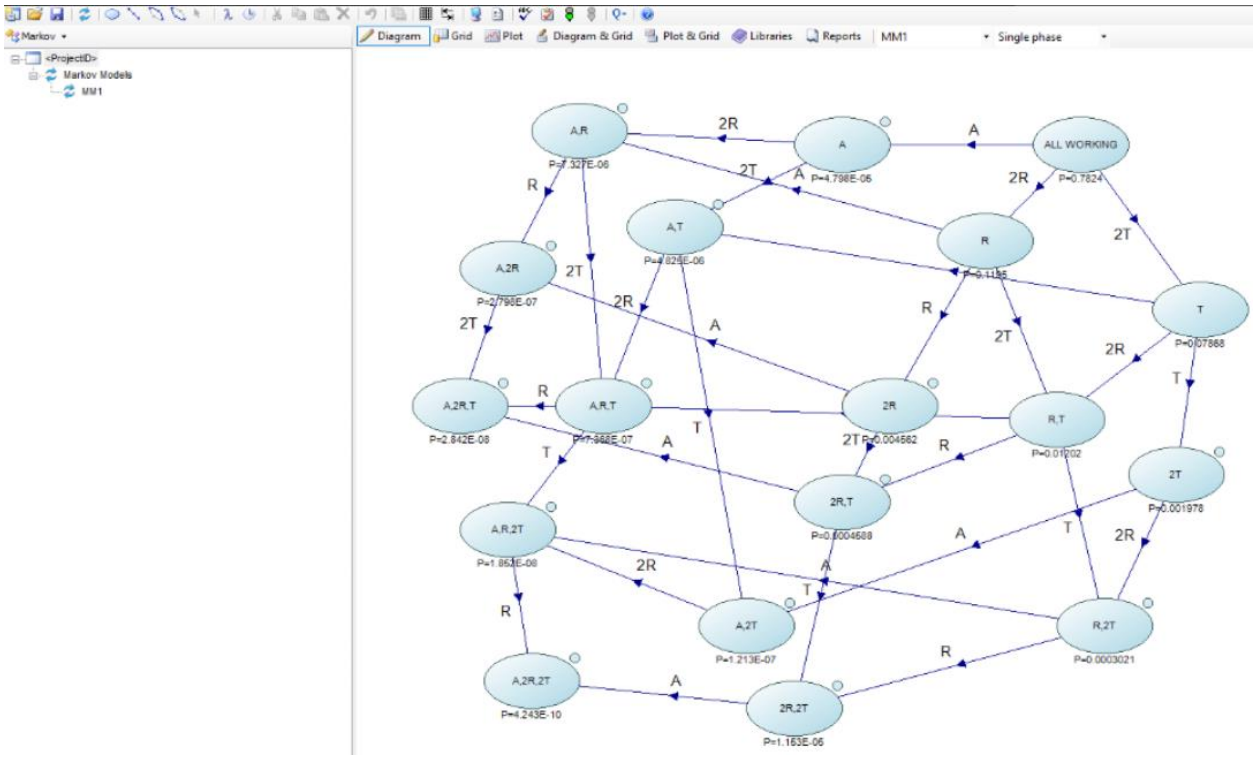

Рисунок 1.9 – Програмне забезпечення Reliability Workbench – марківський аналіз

Програма Reliability Workbench реалізована з англомовним інтерфейсом, про можливість застосування української локалізації не повідомляється а ні на сайті, а ні в інших джерелах.

1.4 Аналіз критичних систем автомобіля і вимог до їх надійності

Як продемонстрував аналіз відповідної літератури [28-33], виявлення деталей транспортних засобів, що є критичними за показниками надійності, проводиться методом комплексного аналізу безвідмовності, довговічності та ремонтопридатності зазначених деталей у наступній послідовності:

По-перше, виявляються деталі, що лімітують безвідмовну роботу вузла, системи чи агрегату. Під деталями, що лімітують безвідмовність, і вузлами

розуміються такі деталі і вузли, гамма-відсотковий ресурс [34] яких на розглянутому пробігу автомобіля нижче 90%, а також нижче 95% для деталей, що безпосередньо впливають на безпеку руху.

По-друге, визначаються деталі та вузли, що лімітують довговічність. Під такими деталями та вузлами розуміються такі, ресурс яких менший за ресурс агрегату або автомобіля до капітального ремонту або менший за ресурс, заданий відповідно до технічних умов заводу-виробника.

По-третє, при відпрацюванні первинного матеріалу обираються деталі та вузли, що лімітують безвідмовність та довговічність агрегатів автомобіля, за якими визначаються трудові та вартісні витрати, що йдуть на усунення відмов деталей.

Після визначення деталей, що лімітують окремо безвідмовність, довговічність та ремонтопридатність, виявляються деталі, що лімітують надійність агрегатів автомобіля.

До деталей і вузлів, критичних за надійністю для автомобілей чи агрегатів, відносяться деталі і вузли, які мають щонайменше 50% відмов від загальної їх кількості і щонайменше 70% вартісних витрат від суми витрат, які необхідні для усунення цих відмов, тобто на запасні частини та проведення робіт із заміни деталей.

У процесі експлуатації автомобілів на утримання їх на високому технічному рівні витрачається набагато разів більше коштів, аніж на виготовлення цих автомобілів.

Одним із суттєвих резервів зниження витрат на підтримку працездатності, підвищення технічної готовності, забезпечення безпеки руху автомобілів та зниження їх шкідливого впливу на навколишнє середовище є підвищення ефективності їх використання та своєчасне і якісне проведення профілактичних заходів.

При технічному обслуговуванні автомобілів необхідно в першу чергу встановити перелік вузлів, деталей, що мають проходити технічне обслуговування та попереджувальний ремонт. До переліку робіт з

вищезазначеного технічного обслуговування та попереджувального ремонту мають бути включені операції не тільки за деталями, що лімітують надійність, а й за деталями, що лімітують перелік профілактичних впливів.

Виявити деталі, що лімітують перелік профілактичних впливів, можна виходячи із значень показників властивостей надійності автомобілів. Кількість відмов дозволяє визначити деталі, що лімітують перелік профілактичних впливів за критерієм безвідмовності. Однак при цьому не враховується величина витрат, пов'язаних із їх заміною.

Не виключається, що мала ймовірність потреби в заміні дорогої деталі може викликати більше втрат, ніж дві заміни дешевих деталей, що легко встановлюються.

У зв'язку з цим можна сформулювати наступне твердження: до лімітуючих перелік профілактичних впливів слід відносити ті деталі, які не відповідають комплексу критеріїв, або хоча б одному з них – властивостей надійності, безвідмовності, довговічності, ремонтопридатності.

За критерієм безвідмовності до переліків, що лімітують, профілактичних впливів необхідно віднести ті деталі, за якими кількість відмов не менше, ніж середня кількість відмов, що припадають на одну деталь агрегату. Середня кількість відмов, що припадає на одну деталь агрегату, дорівнює:

$$N_{\text{середнє}} = \frac{N_{\text{загальне}}}{M} \tag{1.1}$$

де $N_{\text{загальне}}$ – загальна кількість відмов;

M – номенклатура деталей, що підлягають заміні.

У цьому випадку за критерієм безвідмовності до деталей, що лімітує перелік профілактичних впливів, повинні бути віднесені всі деталі даного агрегату, які мають відмови за аналізований період:

$$N_{\text{деталі}} \geq N_{\text{середнє}} \tag{1.2}$$

де $N_{\text{деталі}}$ – кількість відмов деталі.

На рисунку 1.10 наведено взаємозв’язок технологічних показників якості автомобіля.

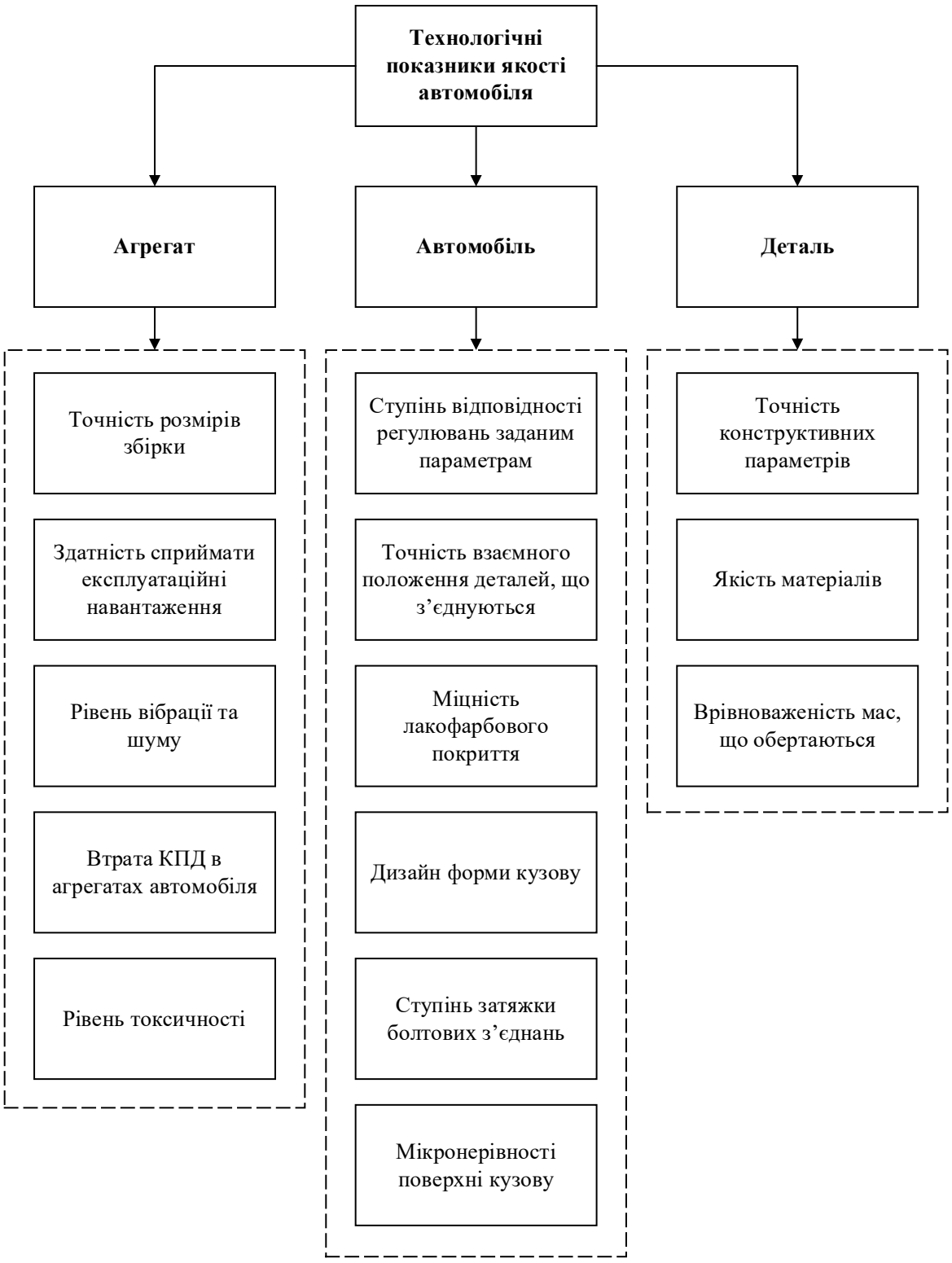

Рисунок 1.10 – Демонстрація взаємозв’язку технологічних показників якості автомобіля

Якщо розглядати економічний аспект експлуатації за умов наявності великого автопарку, то при цьому вибір рухомого складу за економічними показниками здійснюється за результатами експлуатації рухомого складу [33].

На рисунку 1.11 наведена схема вибору рухомого складу автотранспорту за економічними показниками.

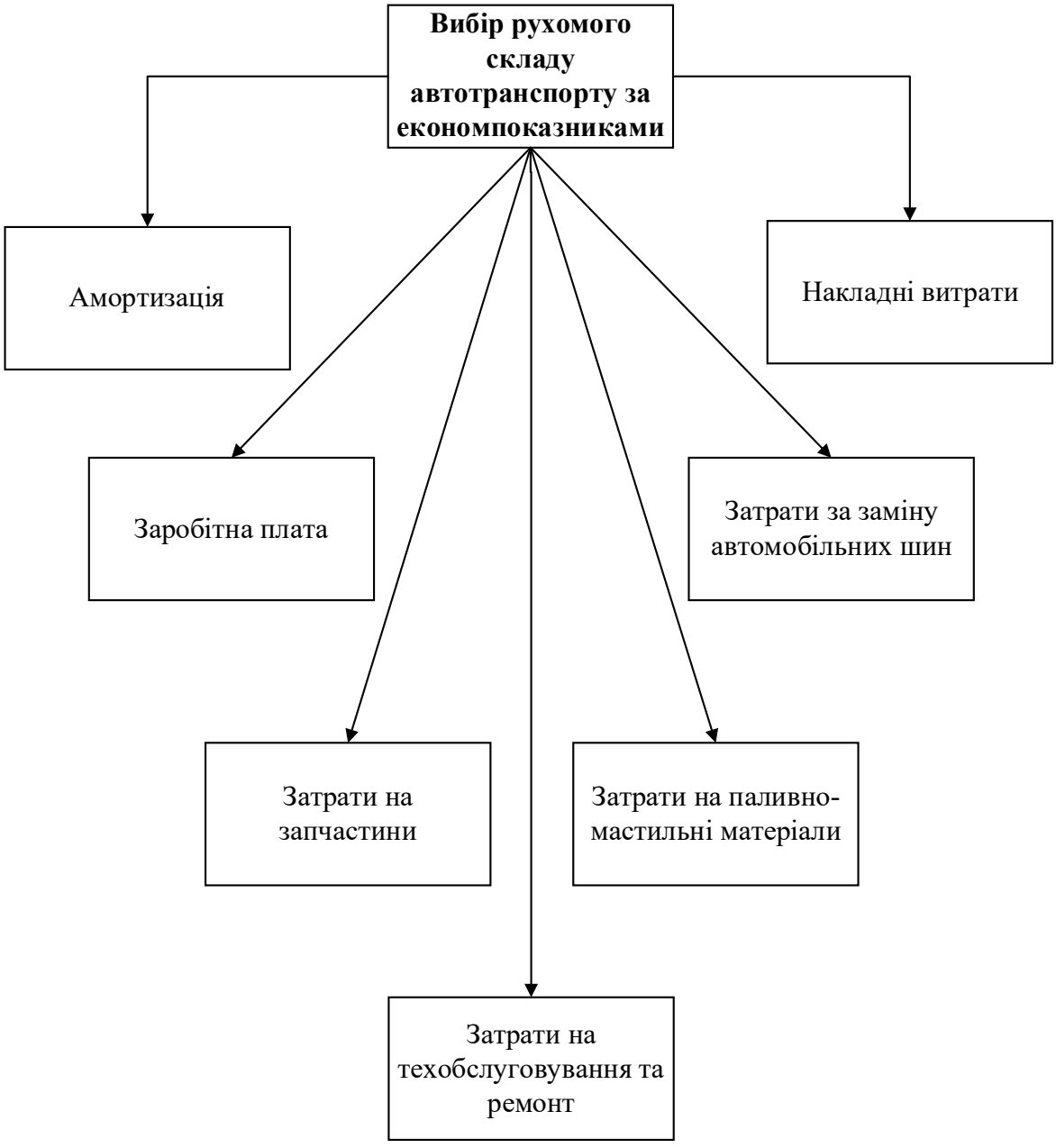

Рисунок 1.11 – Функціональна схема вибору рухомого складу автотранспорту за економічними показниками

Величина витрат, що припадають на одиницю продукції, визначається на основі калькуляції собівартості, в якій всі витрати розподіляються за статтями залежно від їхнього характеру та призначення.

1.5 Результати порівнянь продуктів-аналогів

Виконаний аналіз програмного забезпечення для проведення аналізу дерев відмов продемонстрував, що на поточний час на ринку програмних засобів існує певна кількість комплексних систем для вирішення зазначеної задачі. Деякі з проаналізованих в ході досліджень додатків були описані в підрозділах 1.3.1 та 1.3.2.

Цінова політика пропонованих програмних засобів для проведення аналізу дерев відмов також розрахована здебільшого на великі компанії та організації. Запропоновані ціни є дещо зависокими для українського ринку, що ускладнює використання даних систем в сьогоденних реаліях. Прості і відносно доступні тарифні плани зазвичай не включають в себе необхідні для ефективної роботи майстерні функції. Навпаки, більш функціональні рішення мають досить високу вартість і не є доступними для пересічних користувачів. До того ж, вищезазначені рішення потребують помісячної оплати та в них відсутня можливість придбання пожиттєвої ліцензії, яка, зазвичай, є більш дешевшою при використанні в довгостроковій перспективі. Крім того, розробники завжди можуть різко підвищити ціни на використання продукту. При великій напрацьованій базі даних швидко мігрувати на інший додаток буде досить складно через особливості експорту та імпорту даних в різних програмних засобах. До того ж розробники будь-якого програмного забезпечення, що очевидно, не зацікавленні в реалізації прозорих механізмів експорту баз даних за для зменшення відтоку клієнтів. З огляду на вищезазначене, виглядає актуальним створення безкоштовного програмного забезпечення, або програмного засобу з пожиттєвою ліцензією, що реалізовуватиме можливість вирішення задачі побудови дерев відмов та їх інтелектуального аналізу.

Важливим критерієм є і реалізація україномовного інтерфейсу з огляду на те, що в розглянутих програмних засобах для аналізу дерев відмов зазначена локалізація була відсутня. Враховуючи традиційно скромний рівень володіння технічною англійською мовою серед широких верств населення, підтримка української мови в розроблюваній системі стане в нагоді та спростить впровадження та експлуатацію програмного забезпечення, що створюється.

1.6 Постановка задачі

Завданням на кваліфікаційну роботу є розробка комплексу автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов під управлінням операційної системи Windows. Розроблена система має спростити аналіз дерева відмов та допомогти визначити найбільш критичні вузли та агрегати автомобілів.

Інформаційна система буде створюватися для використання на платформі Windows, що пов’язано з її значним поширенням при використанні на персональних комп’ютерах та ноутбуках. Згідно даних [35], операційні системи Windows в 2024 році охоплюють сегмент «desktop/laptop» більше, ніж на 50%.

Розроблюване програмне забезпечення для оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов має реалізовувати функціональні можливості:

а) аналіз ризиків та оцінка надійності вузлів та агрегатів систем автомобільного транспорту;

б) інтуїтивний інтерфейс для роботи з деревами відмов;

в) графічне представлення вузлів дерева відмов та зв'язків між ними.

д) моделювання можливих сценаріїв відмов;

е) кількісний аналіз ймовірностей відмов та їх впливу на систему;

ж) розрахунок показників надійності, таких як середній час до відмови (MTTF) [36] та середній час відновлення (MTTR) [37];

і) ідентифікація найбільш критичних компонентів системи;

к) рекомендації щодо підвищення надійності на основі проведеного аналізу;

л) можливість експорту даних, забезпечення документування результатів аналізу та створення звітності.

Впровадження зазначеного функціоналу програмного забезпечення може значно підвищити ефективність управління надійністю критичних систем автомобіля та зменшити ризики, пов'язані з відмовами.

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА СТРУКТУР ДАНИХ, ФУНКЦІОНАЛЬНИХ МОДЕЛЕЙ І АЛГОРИТМІВ ПРОГРАМНОГО КОМПЛЕКСУ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДЕРЕВА ВІДМОВ КРИТИЧНИХ СИСТЕМ АВТОМОБІЛЯ

2.1 Розробка структурної моделі бази знань програмного комплексу

В ході проектування програмного комплексу інтелектуального аналізу дерева відмов критичних систем автомобіля було встановлено доцільним зупинитися на аналізі надійності та відмово стійкості п’ятьох підсистем, що входять до склада автомобілей. Це підсистеми:

а) гальмівна система;

б) система управління двигуном;

в) система охолодження двигуна;

д) електрична система;

е) трансмісія.

Розглянемо детальніше кожну з них.

Дерево відмов для гальмівної системи схематично виглядає наступним чином:

Загальне зниження надійності гальмівної системи (OR)

├── *Втрата ефективності ABS (OR)*

│ ├── *Відмова датчиків швидкості колеса*

│ └── *Електронний збій в блоці управління ABS*

├── *Зниження гальмівного зусилля (OR)*

│ ├── *Протікання гальмівної рідини*

│ └── *Механічний знос головного гальмівного циліндра*

└── *Обрив або замикання електропроводки*

Опис логічних зв'язків гальмівної системи можна представити наступним чином: кореневий вузол представляє загальне зниження надійності гальмівної системи. Він об'єднує основні причини зниження надійності гальмівної системи за допомогою логічного оператора OR, що означає, що будь-яка з нижчезазначених причин може призвести до загальної ненадійності гальмівної системи.

«Втрата ефективності ABS» використовує логічний оператор OR, що означає, що будь-яка з дочірніх причин може призвести до втрати зазначеної ефективності.

«Зниження гальмівного зусилля», у свою чергу, використовує логічний оператор OR і аналогічним чином репрезентує той факт, що проявлення однієї з дочірніх причин призведе до проблем з гальмами.

«Обрив або замикання електропроводки» не має дочірніх вузлів і є безпосередньою причиною, яка може призвести до зниження надійності гальмівної системи.

Наступне дерево відмов репрезентує систему управління двигуном:

Загальне зниження надійності системи управління двигуном (OR)

├── *Зниження потужності двигуна (OR)*

│ ├── *Відмова датчика тиску мастила*

│ │ └── *Недостатній рівень мастила (AND)*

│ └── *Відмова датчика температури*

├── *Втрата зв'язку з датчиками (OR)*

│ ├── *Відмова датчика положення колінвала*

│ └── *Відмова датчика витрати палива*

│ └── *Закінчився бензин (NOT)*

└── *Збій блоку управління двигуном*

Загальне зниження надійності системи управління двигуном при цьому об'єднує основні причини за допомогою логічного оператора OR.

Вузли «Зниження потужності двигуна» та «Втрата зв'язку з датчиками» застосовують оператор логічного додавання, таким чином, одна з дочірніх причин може спричинити одну з цих двох неполадок.

«Збій блоку управління двигуном», у свою чергу, не має дочірніх вузлів і є безпосередньою причиною, яка може призвести до зниження надійності системи управління двигуном.

«Відмова датчика тиску» мастила застосовує логічний оператор AND з дочірнім вузлом «Недостатній рівень мастила», що означає, що відмова датчика тиску мастила відбувається через недостатній рівень мастила.

«Відмова датчика витрати палива» використовує логічний оператор NOT з дочірнім вузлом «Закінчився бензин», таким чином, поломка датчика витрати палива відбувається через відсутність бензину.

Дерево відмов системи охолодження двигуна дещо менше за два попередніх, та складається з наступних компонентів:

Перегрів двигуна (OR)

├── *Відмова вентилятора*

├── *Витік охолоджувальної рідини*

└── *Засорення радіатора (OR)*

├── *Відмова термостата*

└── *Механічний збій помпи*

Кореневий вузол «Перегрів двигуна» поєднує можливі причини перегріву двигуна за допомогою логічного оператора OR.

«Відмова вентилятора» та «Витік охолоджувальної рідини» є безпосередніми причинами, що можуть викликати кореневу несправність.

«Засмічення радіатора» застосовує оператор логічного додавання, таким чином «Відмова термостата» або «Механічний збій помпи» можуть призвести до того, що радіатор буде засмічений.

Дерево відмов електричної системи виглядає так:

Повна відмова електричної системи (OR)

├── *Відмова акумулятора (OR)*

│ ├── Розряд акумулятора

│ └── Старий акумулятор

├── Коротке замикання (OR)

│ ├── Перегрів проводів

│ └── Пошкодження ізоляції

└── Пошкодження проводки

Кореневим вузлом є «Повна відмова електричної системи», яка із застосуванням оператора OR, поєднана з іншими причинами, що призводять до такого роду несправності.

«Відмова акумулятора» та «Коротке замикання» також можуть бути викликані однією будь-якою подією з нижніх рівнів через використання оператора логічного додавання

«Пошкодження проводки» є безпосередньою причиною для виклику кореневої несправності.

Відмова трансмісії є останнім деревом у переліку відмов, і воно репрезентується наступною структурою:

Відмова трансмісії (OR)

├── Знос зчеплення (OR)

│ ├── Невідповідне зчеплення

│ └── Низька якість зчеплення

├── Поломка коробки передач (OR)

│ ├── Знос шестерень

│ └── Поломка валів

├── Низький рівень трансмісійної рідини (OR)

│ ├── Протікання мастила

│ └── Неправильний рівень мастила

└── Пошкодження карданного вала

«Відмова трансмісії» являє собою кореневий вузол, що об’єднує інші причини оператором логічного додавання.

«Пошкодження карданного вала» є безпосередньою причиною, яка може призвести до відмови трансмісії, в той час як «Знос зчеплення», «Поломка коробки передач» та «Низький рівень трансмісійної рідини» використовують оператор OR, та можуть бути викликані лише однією з можливих дочірніх причин.

Описані дерева відмов можуть забезпечити аналіз надійності основних систем автомобілей, в тім цей перелік не є вичерпним та може доповнюватися для проведення більш ретельних розрахунків.

2.2 Функціональна схема автоматизованої системи оцінки надійності

На рисунку 2.1 наведено функціональну схему розроблюваного програмного забезпечення. Вона складається з чотирьох основних сутностей: «Користувач», «Користувацький інтерфейс», «Модуль відображення дерев відмов», «Модуль розрахунку вірогідностей відмов».

Користувач звертається до «Користувацького інтерфейсу», що забезпечує процес інтерактивної взаємодії з додатком. Користувацький інтерфейс включає в себе панель відображення дерев відмов, список вибору дерев відмов, навігаційні елементи та елементи виведення розрахованих ймовірностей відмов.

«Модуль відображення дерев відмов», своєю чергою, забезпечує генерацію та репрезентацію відповідних матеріалів, що допомогають виконати аналіз надійності критичних систем автомобіля. Він включає в себе показ дерев відмов гальмівної системи, системи управління двигуном, системи охолодження двигуна, електричної системи та трансмісії.

«Модуль розрахунку вірогідностей відмов» складається з двох компонент: модулю розрахунку вірогідності відмови за методом Монте-Карло та модулю виведення відповідних результатів на головний екран розроблюваного програмного комплексу.

Рисунок 2.1 - Функціональна схема комплексу автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов

Зазначена функціональна схема описує структуру комплексу автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов.

2.3. Розробка алгоритму програмного комплексу

На рисунку 2.2 наведено блок-схему алгоритму програмного комплексу автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов.

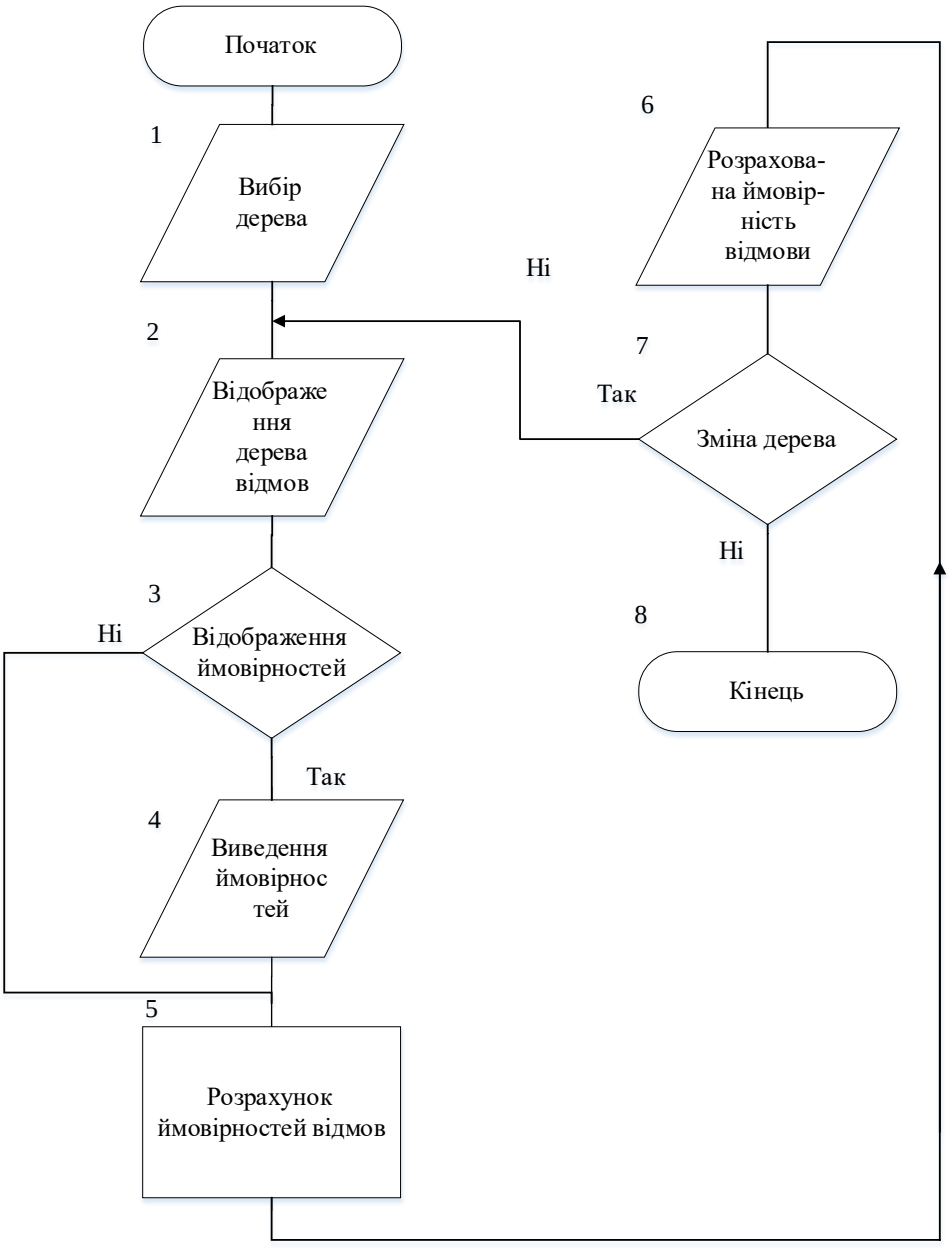

Рисунок 2.2 – Алгоритм роботи програмного комплексу

На початку роботи перед користувачем виникає головне вікно програми, де він бачить перелік дерев відмов для вибору та навігаційні елементи. При виборі потрібного для аналізу дерева відмов воно демонструється на головному вікні.

За необхідності користувач може включити демонстрацію вірогідносте виникнення тих чи інших несправностей на дереві. При натисканні на кнопку розрахунку виконується розрахунок ймовірності відмови за методом Монте-Карло. Цей метод дає змогу моделювати безліч можливих сценаріїв відмов і оцінювати загальну ймовірність відмови системи. Він генерує випадкові сценарії відмов на основі заданих імовірностей відмов для базових подій. Після виконання розрахунків відбувається відображення отриманих ймовірностей у відповідному текстовому полі. За необхідності користувач може змінити дерево та виконати аналіз надійності за іншою підсистемою автомобіля, або завершити роботу з програмою.

3 РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ОЦІНКИ НАДІЙНОСТІ КРИТИЧНИХ СИСТЕМ АВТОМОБІЛЯ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДЕРЕВА ВІДМОВ ТА ЙОГО ОПИС

3.1 Встановлення системних вимог до розроблюваного програмного забезпечення

Розроблюване програмне забезпечення для оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов має працювати на комп'ютерах з таким апаратним забезпеченням:

а) процесор: Pentium M, Pentium D, Core 2 Duo, Core i3, Core i5, Core i7, Core i9, Athlon, Duron, Sempron, Phenom, Ryzen - 1000 МГц та вище, що зумовлено вимогами програми та операційної системи;

б) операційна система: 32-х або 64-х розрядна система Windows 7, Windows 8, Windows 8.1, Windows 10, Windows 11. Також можливе використання серверних версій Windows;

в) кількість оперативної пам'яті: 1 Гб для Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10, та 2 Гб для Windows 11. Заначений об'єм необхідний для коректної роботи операційної системи та програми в ній. Програма має займати в оперативній пам'яті не більше 70 Мб;

г) монітор: роздільна здатність екрану – 1024*768 і більше – для коректного відображення головного вікна додатку;

д) графічна карта: об'єм пам'яті – від 16 Мб;

е) об'єм жорсткого диску: не менше 150 Мб. Програмні файли збільшують свій розмір по мірі збільшення кількості записів в базі даних;

є) клавіатура для вводу даних;

ж) маніпулятор «миша» для зручного користування інтерфейсом.

3.2 Огляд інструментальних засобів для розробки комплексу автоматизації оцінки надійності критичних систем автомобіля

Оскільки розроблювана програма має працювати на платформі Windows в різних версіях цієї операційної системи, то слушно буде використати для розробки бібліотеку Microsoft .NET Framework. Згідно з [38], Microsoft .NET Framework – це програмна платформа, основою якої є загальномовне середовище виконання Common Language Runtime (CLR), яке підходить для різних мов програмування. Можливості CLR є доступними в будь-яких мовах програмування, що використовують це середовище. Також .NET Framework можна назвати програмною технологією, що призначена для створення програм та сайтів, основною ідеєю якої є сумісність різних служб, що написані на різних мовах. Наприклад, служба, що написана на C++ для Microsoft .NET Framework, може звернутися до методу класу з бібліотеки, що написана на Delphi, або на мові C# можна написати клас, що буде унаслідуватися від класу, написаного на Visual Basic .NET тощо [39].

Основною ідеєю при розробці .NET Framework було забезпечення свободи розробника за рахунок надання йому можливості створювати додатки різних типів, що здатні виконуватися на різних типах пристроїв і в різних середовищах. Другим принципом стала орієнтація на системи, що працюють під управлінням сімейства операційних систем Microsoft Windows [40].

Програма для .NET Framework, написана на будь-якій мові програмування, що підтримується платформою, спочатку перекладається компілятором в єдиний для .NET проміжний байт-код Common Intermediate Language (CIL). Далі код виповнюється віртуальною машиною CLR. Використання віртуальної машини є оптимальним рішенням, оскільки позбавляє розробників необхідності враховувати особливості апаратної частини. При використанні віртуальної машини CLR вбудований в неї JIT-компілятор перетворює проміжний байт-код в машинні коди потрібного процесора. Сучасна технологія динамічної компіляції дозволяє досягти високого рівня швидкодії. Віртуальна машина CLR також сама піклується про

базову безпеку, управлінні пам'яттю, обробку виняткових результатів, позбавляючи розробника від частини роботи. [41]

Отже, Microsoft .NET Framework є потужним інструментом, що дозволяє розробляти додатки під Windows на різних мовах програмування. Відповідно, для розробки програми із застосуванням Microsoft .NET Framework пропонується обрати мову програмування C#.

C# є універсальною, безпечною до типів, об'єктно-орієнтованою мовою. Згідно з [42], метою створення мови є підвищення продуктивності роботи програмістів. Для цього в мові підтримується баланс між простотою, виразністю і продуктивністю. В C# реалізована розширена об'єктно-орієнтована парадигма, яка включає інкапсуляцію, успадкування і поліморфізм. Також C# є мовою, безпечною до типів та підтримує статичну типізацію, при якій мова слідкує за безпекою до типів під час компіляції. Статична типізація дозволяє запобігти виникненню великої кількості помилок ще до запуску програми. При автоматичному управлінні пам'яттю C# покладається на виконуюче середовище. Загальномовне виконуюче середовище (CLR) має збирач сміття, який виконується як частина програми, звільняючи пам'ять, що зайнята об'єктами, посилання на які більше не існують. Це знімає з програмістів необхідність в явному звільнення пам'яті для об'єкта, усуваючи проблему некоректних покажчиків, яка зустрічається в таких мовах, як C++.

Для реалізації візуального інтерфейсу пропонується застосування Windows Forms – графічний API, що спрощує розробку графічних інтерфейсів. Він забезпечує швидкість та зручність розробки.

Таким чином, розробка інформаційної системи буде здійснюватися на мові програмування C# з використанням бібліотеки Microsoft .NET Framework.

3.3 Реалізація та опис програмного комплексу оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов

Головне вікно програмного комплексу оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов зображене на рисунку 3.1. Воно містить перелік наявних для відображення та аналізу дерев відмов, елементів керування для відображення дерев, їх зміни, демонстрації вірогідностей виникнення несправностей при проявах базових подій, текстове поле для введення кількості випробувань за методом Монте-Карло, та елемент виклику процедури розрахунку вірогідності появи відмов за даним деревом.

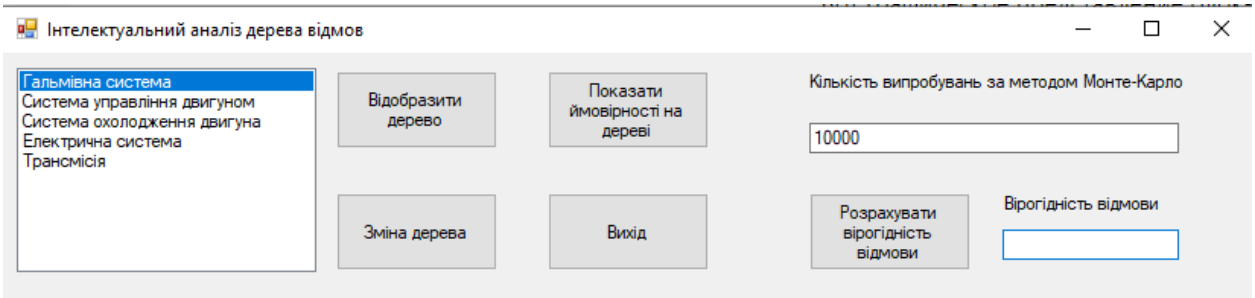

Рисунок 3.1 – Головне вікно програмного комплексу, що демонструється після його запуску

При натисканні на кнопку «Відобразити дерево» у нижній панелі формується та відображається дерево відмов (за замовчуванням це дерево відмов гальмівної системи). Аналогічна подія виникає і при натисканні на один з елементів списку, що розташований на формі зліва, з тією відмінністю, що при цьому формується та відображається саме обране у цьому списку дерево.

Відображення дерева відмов на головному вікні створеного програмного комплексу наведено на рисунку 3.2.

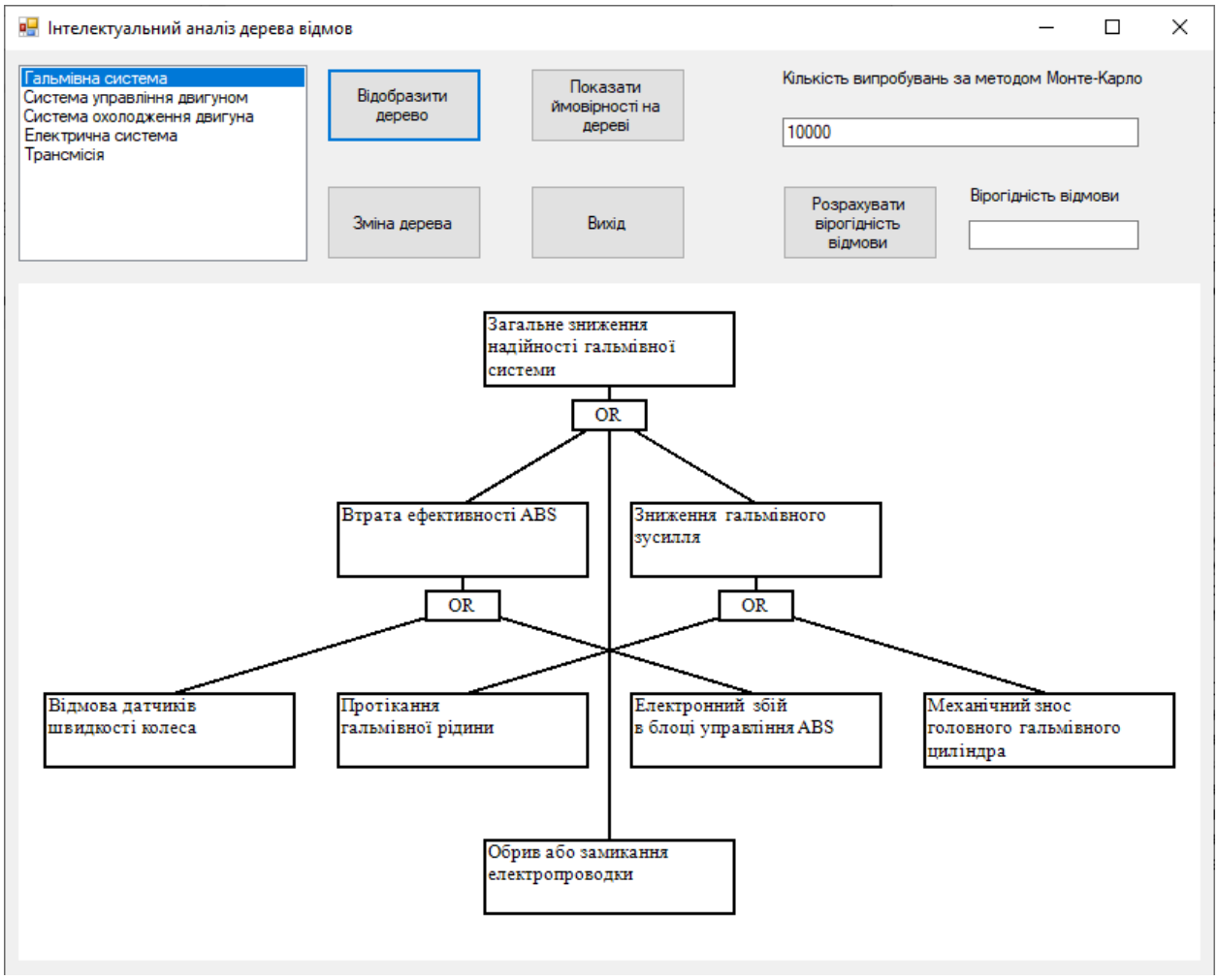

Рисунок 3.2 – Головне вікно створеного програмного комплексу зі сформованим деревом відмов «Гальмівна система»

За необхідності користувач може викликати відображення на наведеній схемі вірогідностей настання базових подій шляхом натискання кнопки «Показати ймовірності на дереві». Після цього над прямокутниками базових подій з’являться відсоткові ймовірності, що наводяться червоним кольором для логічного розділення структури дерева та числових даних. Треба зазначити, що при зміні дерева відмов та завантаженні нового, ймовірності настання базових подій зникають з малюнку, та можуть бути повторно викликані до виведення шляхом застосування кнопки «Показати ймовірності на дереві».

Реалізація зазначеної опції наведена на рисунку 3.3.

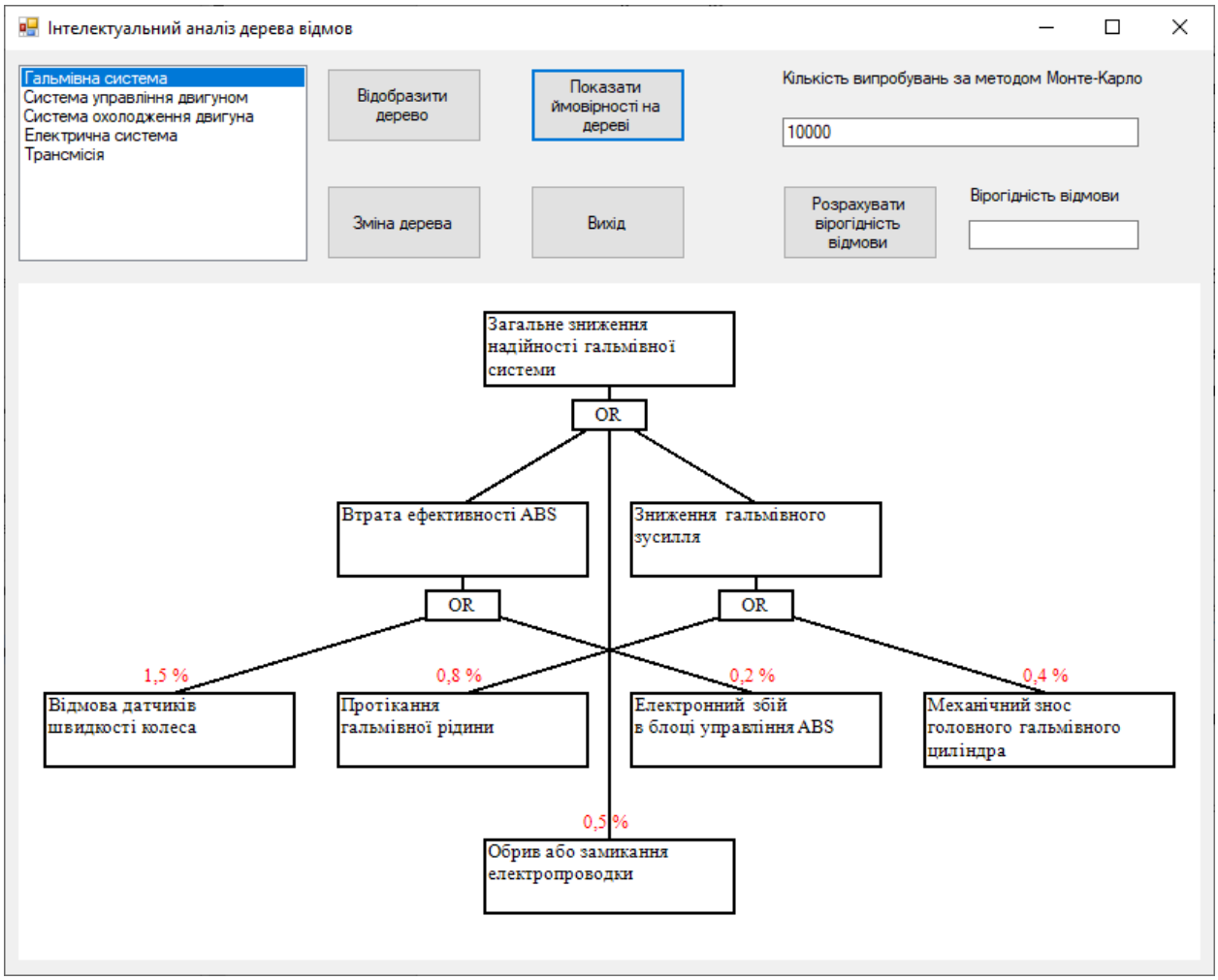

Рисунок 3.3 – Вікно розробленого програмного комплексу зі сформованим деревом відмов «Гальмівна система» (демонстрація ймовірностей настання базових подій)

На рисунку 3.4, своєю чергою, наведено дерево відмов «Система управління двигуном». При цьому демонструється розрахунок вірогідності настання кореневої відмови. Для її розрахунку необхідно після обрання дерева відмов натиснути на кнопку «Розрахунок вірогідності відмови».

Розрахунок зазначеної ймовірності відмови розраховується за методом Монте-Карло. Зазначений метод є статистичним методом, що використовує випадкове моделювання для розв'язання складних математичних задач і оцінки ймовірностей. У контексті аналізу дерева відмов автомобільних систем метод Монте-Карло використовується для оцінки ймовірності відмови всієї системи на основі ймовірностей відмов окремих компонентів.

Метод Монте-Карло можна описати наступним чином: спочатку відбувається ініціалізація вхідних даних, де визначаються ймовірності відмов для кожного компонента дерева відмов. Далі лля кожного компонента дерева відмов генерується випадкове число від 0 до 1 і проводиться визначення стану компонентів - перевіряється, чи менше згенероване випадкове число, ніж задана ймовірність відмови компонента. Якщо так, то вважається, що компонент відмовив, в іншому випадку вважається, що працеспроможність не порушена.

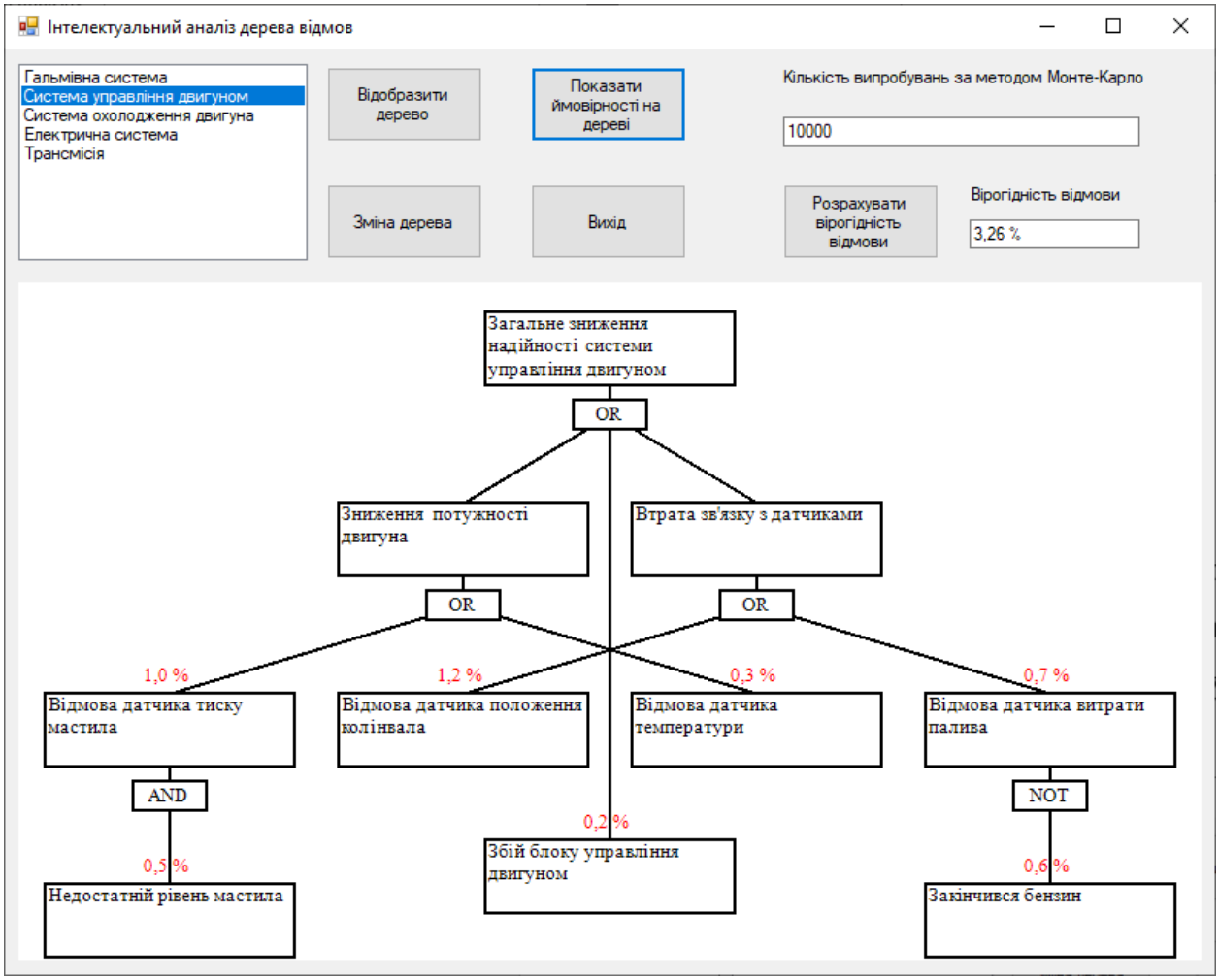

Рисунок 3.4 – Вікно програмного комплексу зі сформованим деревом відмов «Система управління двигуном»

На основі логічних операторів у дереві відмов визначається, чи призвела відмова окремих компонентів до відмови всієї системи. Далі проводиться

повторення експерименту та попередні кроки повторюються велику кількість разів для отримання статистично значущої оцінки ймовірності відмови системи.

Підсумкову ймовірність відмови системи при цьому обчислюють як відношення числа випадків відмови системи до загальної кількості проведених експериментів.

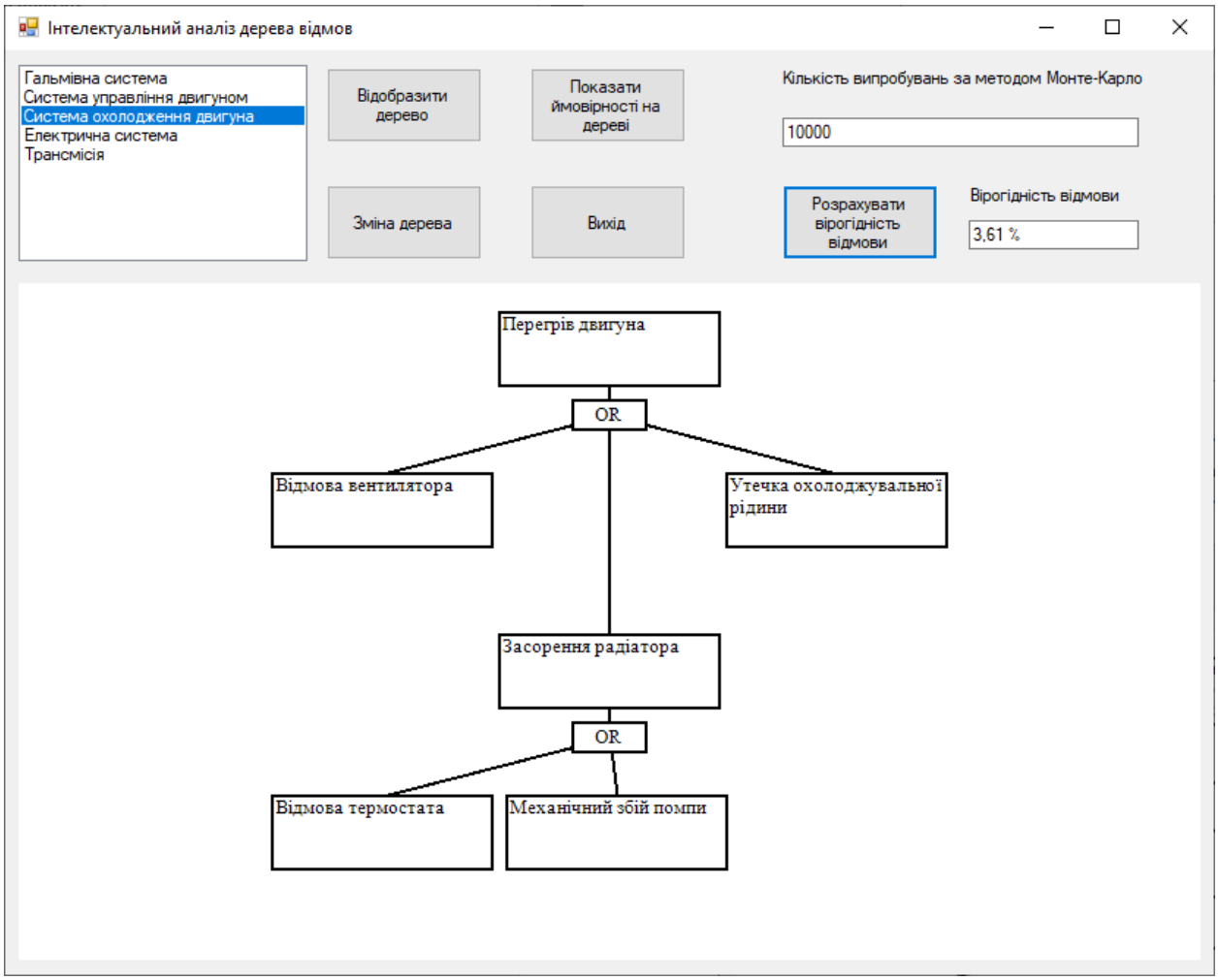

Рисунок 3.5 – Головне вікно створеного програмного комплексу зі сформованим деревом відмов «Система охолодження двигуна»

Розглянемо у наступному лістингу, як у програмному комплексі реалізовано формування та відображення дерев відмов (на прикладі дерева системи охолодження двигуна).

```
private void DrawCoolingSystemTree(Graphics g)
        {
            // Налаштування графіки
            g.Clear(Color.White);
            Pen pen = new Pen(Color.Black, 2);
            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Black;

            // Координати та розміри елементів
            int centerX = panel1.Width / 2;
            int startY = 20;

            int nodeWidth = 150;
            int nodeHeight = 50;
            int verticalSpacing = 60;
            int horizontalSpacing = 160;

            // Малювання вузлів
            Rectangle rootRect = new Rectangle(centerX -
nodeWidth / 2, startY, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, rootRect);
            g.DrawString("Перегрів двигуна", font, brush,
rootRect);

            // Вузли другого рівня
            Rectangle nodeB = new Rectangle(centerX - nodeWidth
- horizontalSpacing / 2, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeB);
            g.DrawString("Відмова вентилятора", font, brush,
nodeB);

            Rectangle nodeC = new Rectangle(centerX +
horizontalSpacing / 2, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeC);
            g.DrawString("Утечка охолоджувальної рідини", font,
brush, nodeC);

            Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 2, nodeWidth,
nodeHeight);
            g.DrawRectangle(pen, nodeD);
            g.DrawString("Засорення радіатора", font, brush,
nodeD);
            // Вузли третього рівня для nodeD
            Rectangle nodeE = new Rectangle(nodeD.X -
horizontalSpacing / 2 - nodeWidth / 2, nodeD.Y + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeE);
            g.DrawString("Відмова термостата", font, brush,
nodeE);
```

```
            Rectangle nodeF = new Rectangle(nodeD.X +
horizontalSpacing / 2 - nodeWidth / 2, nodeD.Y + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeF);
            g.DrawString("Механічний збій помпи", font, brush,
nodeF);

            // Логічні оператори
            int operatorWidth = 50;
            int operatorHeight = 20;
            Rectangle or1 = new Rectangle(centerX -
operatorWidth / 2, rootRect.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or2 = new Rectangle(nodeD.X + nodeWidth /
2 - operatorWidth / 2, nodeD.Y + nodeHeight + 10, operatorWidth,
operatorHeight);

            // Лінії з'єднань
            g.DrawLine(pen, new Point(rootRect.X + nodeWidth /
2, rootRect.Y + nodeHeight), new Point(centerX, or1.Y +
operatorHeight / 2));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeB.X + nodeWidth / 2,
nodeB.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeC.X + nodeWidth / 2,
nodeC.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeD.X + nodeWidth / 2,
nodeD.Y));

            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
nodeD.Y + nodeHeight), new Point(nodeD.X + nodeWidth / 2, or2.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeE.X + nodeWidth / 2,
nodeE.Y));
            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeF.X + nodeWidth / 2,
nodeF.Y));
            Brush whiteBrush = Brushes.White;
            g.FillRectangle(whiteBrush, or1);
            g.DrawRectangle(pen, or1);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or1));

            g.FillRectangle(whiteBrush, or2);
            g.DrawRectangle(pen, or2);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or2));
        }
```

Як видно з наведеного лістингу, функція формування та відображення дерева вимог складається з декількох частин, серед яких налаштування графічних та текстових компонент; визначення координат та розмірів застосовуваних в дереві елементів; безпосередньо малювання вузлів першого, другого та третього рівнів; відображення на дереві логічних операторів; формування ліній з'єднань елементів таким чином, щоб вони не перекривали одна одну.

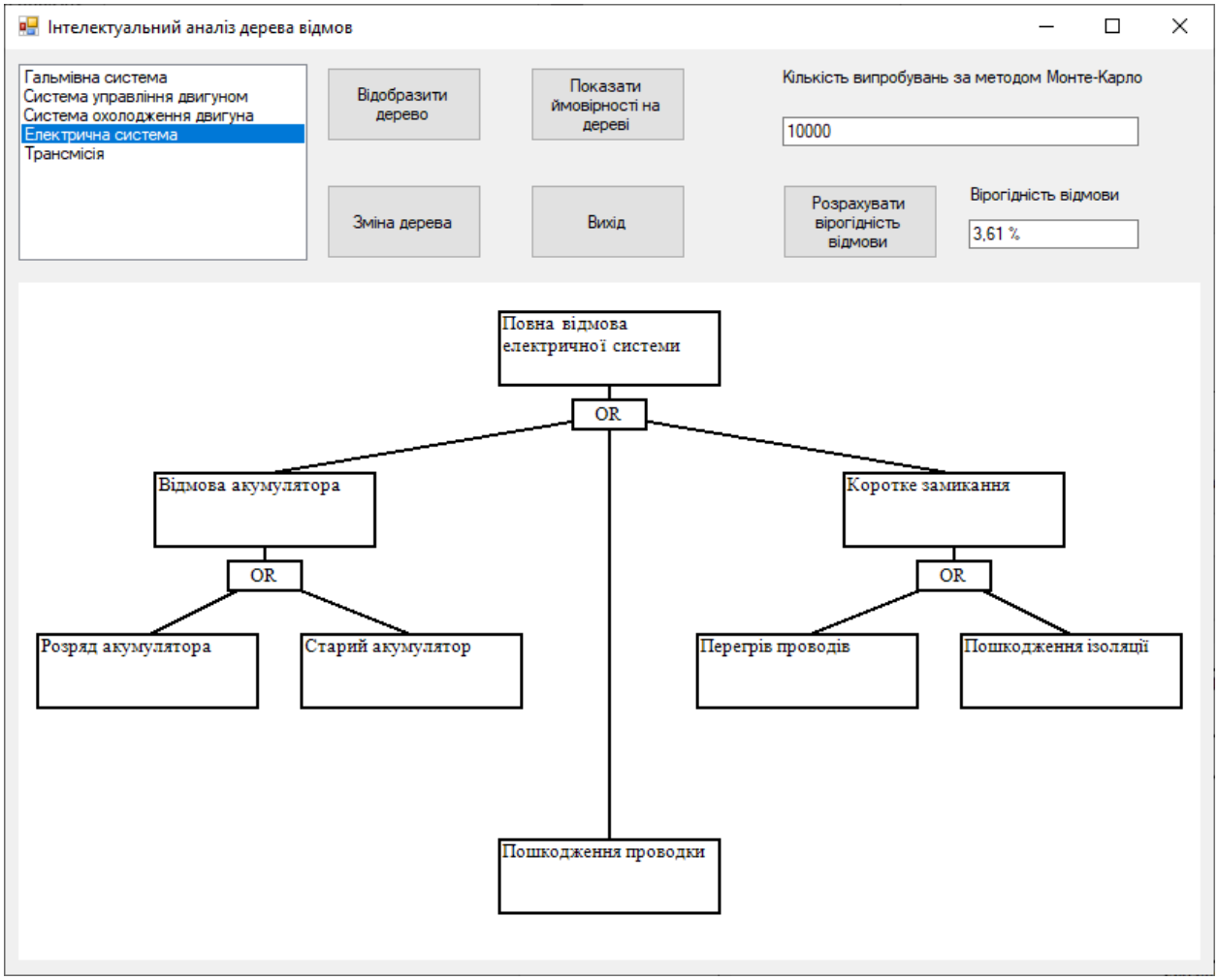

Рисунок 3.6 – Вікно програмного комплексу зі сформованим деревом відмов «Електрична система»

Наведемо лістінг виконання розрахунків вірогідностей відмов за методом Монте-Карло:

```
private void MonteCarloSimulation(int iterations)
        {
            double failureProbability = 0;
            for (int i = 0; i < iterations; i++)
            {
                if (SimulateFailure())
                {
                    failureProbability += 1.0 / iterations;
                }
            }
            textBoxMonteCarloSimulation.Text                    =
$"{failureProbability:P2}";
```

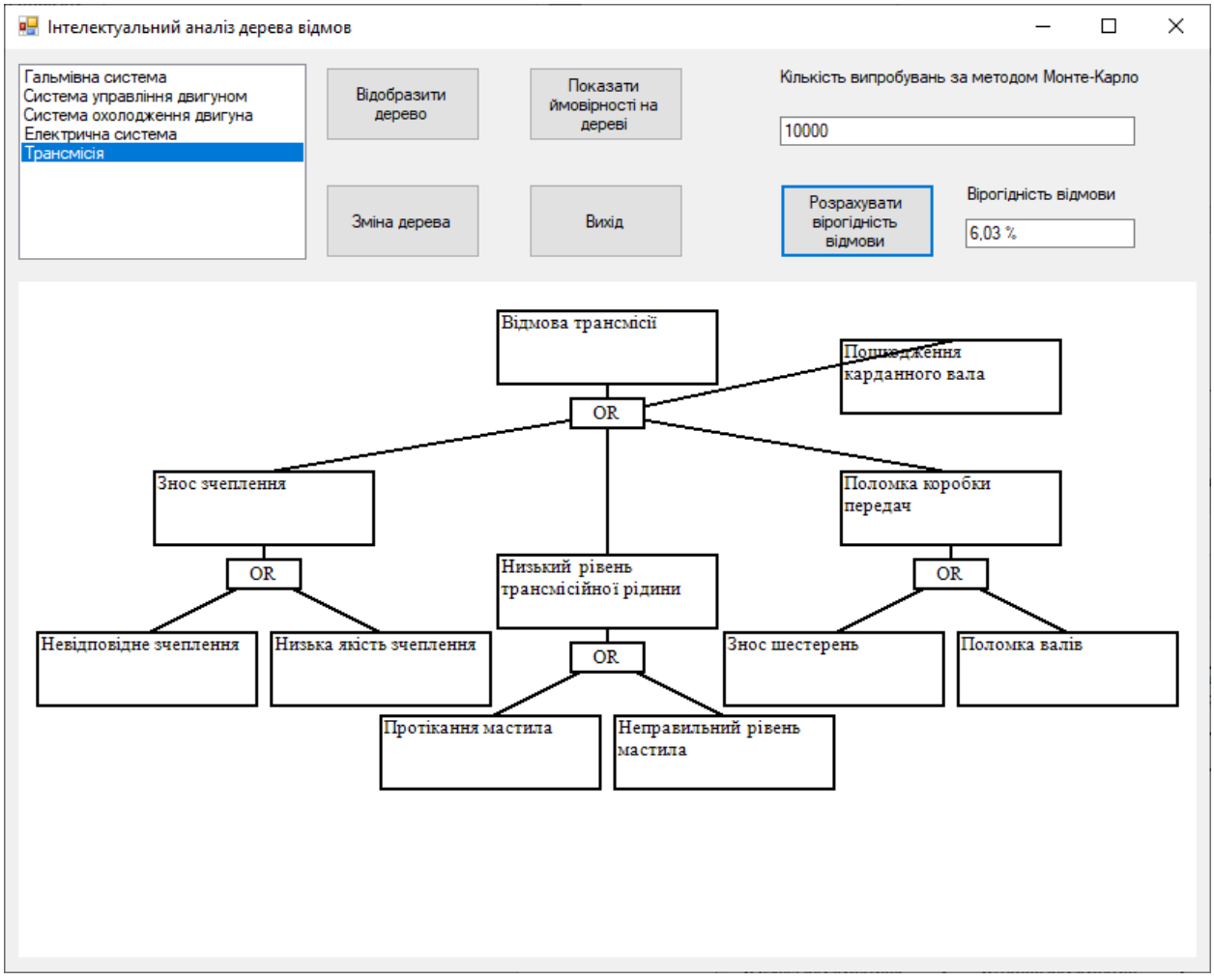

Рисунок 3.7 – Головне вікно створеного програмного комплексу зі сформованим деревом відмов «Трансмісія»

Таким чином, в процесі роботи було розроблено програмний комплекс оцінки надійності критичних систем автомобіля на основі інтелектуального

аналізу дерева відмов, що відповідає встановленим в пунктах 1 та 2 даної дипломної роботи вимогам, та який має забезпечити адекватне оцінювання надійності наведених в програмі підсистем транспортних засобів.

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи було розроблено програмний комплекс оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов, що дозволяє користувачеві проводити тестування безпеки використання транспортних засобів та виконувати розрахунок ймовірностей настання відмов по базовим підсистемам зазначених транспортних засобів.

При виконанні дипломної роботи було:

– проведено аналіз базових положень загальної теорії надійності технічних систем;

– здійснено аналіз методів оцінки надійності технічних систем на основі аналізу дерева відмов;

– проведено аналіз програмних засобів оцінки надійності технічних систем на основі аналізу дерева відмов

– розроблено структурну модель бази знань програмного комплексу;

– створено функціональну схему програмного комплексу оцінки надійності критичних систем автомобіля;

– розроблено алгоритм роботи програмного комплексу;

– створено програмне забезпечення для оцінки надійності критичних систем автомобіля.

Розроблений комплекс автоматизації оцінки надійності критичних систем автомобіля на основі інтелектуального аналізу дерева відмов може застосовуватися в машинобудівних компаніях різних масштабів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ланде Д.В., Субач І.Ю., Бояринова Ю.Є. Основи теорії і практики інтелектуального аналізу даних у сфері кібербезпеки: навчальний посібник. — К.: ІСЗЗІ КПІ ім. Ігоря Сікорського, 2018. — 297 с.

2. ДП «Український науково-дослідний і навчальний центр проблем стандартизації, сертифікації та якості» (ДП «УкрНДНЦ»). (2022). *Аналіз дерева несправностей (FTA)* (ДСТУ EN 61025:2022).

3. Дзюба Л.Ф. Надійність технічних систем і техногенний ризик : навчальний посібник / Л. Ф. Дзюба, М. І. Кусій, О. В. Меньшикова. – Львів: Вид-цтво ЛДУ БЖД, 2017. - 192 с.

4. Василевський, О. М. Нормування показників надійності технічних засобів : навчальний посібник / О. М. Василевський, О. Г. Ігнатенко. – Вінниця : ВНТУ, 2013. – 160 с.

5. Іванюта І.Д., Рибалка В.І., Рудоміно-Дусятська І.А. Елементи теорії ймовірностей та математичної статистики. К.: Слово, 2006.

6. Kusuoka S. Stochastic Analysis. Singapore : Springer Singapore, 2020. URL: https://doi.org/10.1007/978-981-15-8864-8 (дата звернення: 02.06.2024).

7. Яковина В., Симець І. Метод представлення марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку для оцінювання надійності програмного забезпечення. *CSIT*. 2022. № 3. С. 66-73. URL: https://doi.org/10.31891/CSIT-2021-5-9 (дата звернення: 20.05.2024).

8. Павлюк О.М., Медиковський М.О, Лиса Н.К., Ізонін І.В. Основи теорії надійності технічних систем. Львівська політехніка, 2021. 208 с.

9. Болтянська Н.І. Надійність технологічних систем. Курс лекцій. Мелітополь: ВПЦ «Люкс». 2019. 168 с.

10. What is mean time between failure (MTBF)? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/mtbf

11. Геселева Н., Пронюк Г. Особливості марковського моделювання для оцінювання надійності технічних систем. *Економіка і суспільство*. 2018. № 16. С. 965–971.
12. ISO 9000:2015 Quality management systems — Fundamentals and vocabulary [Електронний ресурс] – Режим доступу до ресурсу: https://www.iso.org/ru/standard/45481.html (дата звернення: 24.05.2024).
13. Imai M. Kaizen: The Key To Japan's Competitive Success. McGraw-Hill/Irwin, 1986. 260 p.
14. Mahar D. J. Fault tree analysis application guide. Rome, NY (P.O. Box 4700, Rome, 13440-8200) : The Center, 1990. 109 p.
15. Cui T., Li S. Space Fault Tree Theory and System Reliability Analysis. EDP Sciences, 2020. URL: https://doi.org/10.1051/978-2-7598-2504-2 (дата звернення: 15.05.2024).
16. Fault tree analysis (FTA) [Електронний ресурс] – Режим доступу до ресурсу: https://fiixsoftware.com/glossary/fault-tree-analysis/ (дата звернення: 17.05.2024).
17. Hurdle E. E., Bartlett L. M., Andrews J. D. System fault diagnostics using fault tree analysis. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*. 2007. Vol. 221, no. 1. P. 43–55. URL: https://doi.org/10.1243/1748006xjrr6 (date of access: 21.05.2024).
18. Quantitative Fault Tree Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://www.fault-tree-analysis.com/ (date of access: 20.05.2024).
19. Joseph-McCarthy D., Hogle J. M., Karplus M. Use of the multiple copy simultaneous search (MCSS) method to design a new class of picornavirus capsid binding drugs. Proteins: Structure, Function, and Genetics. 1997. Vol. 29, no. 1. P. 32–58. URL: https://doi.org/10.1002/(sici)1097-0134(199709)29:1%3C32::aid-prot3%3E3.0.co;2-h (date of access: 21.05.2024).
20. Binary Decision Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/binary-decision-diagram/ (date of access: 23.05.2024).

21. TopEvent FTA Pricing [Електронний ресурс] – Режим доступу до ресурсу: https://www.fault-tree-analysis.com/pricing (date of access: 20.05.2024).
22. Reliability Workbench. Fully integrated reliability and safety software for the professional. [Електронний ресурс] – Режим доступу до ресурсу: https://www.isograph.com/software/reliability-workbench/ (date of access: 21.05.2024).
23. Аналіз режиму відмови, наслідків і критичності (FMECA) [Електронний ресурс] – Режим доступу до ресурсу: https://visuresolutions.com/ukКерівництво-fmea-з-управління-ризиками/fmeca/ (date of access: 22.05.2024).
24. Bertolini M., Braglia M., Carmignani G. An FMECA-based approach to process analysis. *International Journal of Process Management and Benchmarking*. 2006. Vol. 1, no. 2. P. 127. URL: https://doi.org/10.1504/ijpmb.2006.009769 (date of access: 22.05.2024).
25. ДСТУ EN 61078:2022 Блок-схеми надійності (EN 61078:2016, IDT; IEC 61078:2016, IDT) [Електронний ресурс] – Режим доступу до ресурсу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=100885 (дата звернення: 22.05.2024).
26. Ткаченко І. О. Ризики у транспортних процесах : навч. посібник / І. О. Ткаченко ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2017. – 114 с.
27. Markov Analysis in Reliability Workbench [Електронний ресурс] – Режим доступу до ресурсу: https://www.isograph.com/software/reliability-workbench/markov-analysis-software/ (дата звернення: 23.05.2024).
28. Бороденко Ю.М., Дзюбенко О.А., Биков О.М. Діагностика електрообладнання автомобілів. Харків: ХНАДУ, 2014. –300 с
29. Біліченко В.В., Крещенецький В.Л., Кукурудзяк Ю.Ю., Цимбал С.В. Основи технічної діагностики колісних транспортних засобів. Вінниця: ВНТУ, 2012. – 118 с.

30. Канарчук В.Є., Лудченко О.А., Чигиринець А.Д. Основи технічного обслуговування і ремонту автомобілів. Київ: Вища школа, 1994. – 599 с.
31. Форнальчик Є.Ю., Оліскевич М.С. Технічна експлуатація та надійність. Львів: Афіша, 2004. – 492 с.
32. Дембіцький В.М., Павлюк В.І., Придюк В.М. Технічна експлуатація автомобілів. Луцьк: Луцький НТУ, 2018. – 473 с.
33. Procedure for acquisition and processing of information on reliability of parts and units of open pit dump trucks / S. Z. Kabikenov et al. *Gornyi Zhurnal*. Academic Research, Uzbekistan. 2015. P. 69–71. URL: https://doi.org/10.17580/gzh.2015.09.15 (date of access: 24.05.2024).
34. Гамма - відсотковий ресурс. Державні будівельні норми України [Електронний ресурс] – Режим доступу до ресурсу: https://dbn.co.ua/blog/gamma_vidsotkovij_resurs/2016-12-09-14766 (дата звернення: 25.05.2024).
35. Market Share Statistics for Internet Technologies [Електронний ресурс] – Режим доступу до ресурсу: https://netmarketshare.com (дата звернення: 27.05.2024).
36. Mean Time To Failure (MTTF) [Електронний ресурс] – Режим доступу до ресурсу: https://www.clickmaint.com/maintenance-calculator/mttf-calculator (дата звернення: 28.05.2024).
37. What does MTTR mean? [Електронний ресурс] – Режим доступу до ресурсу: https://operations1.com/en/glossary/mttr (дата звернення: 28.05.2024).
38. NET_Framework [Електронний ресурс] – Режим доступу до ресурсу: https://www.wikiwand.com/en/.NET_Framework (дата звернення: 28.05.2024).
39. Mayo J. Microsoft Visual studio 2010: A beginner's guide. New York : McGraw-Hill, 2010. 426 p.
40. Principal Design Features of .NET Framework [Електронний ресурс] – Режим доступу до ресурсу: http://www.zabalnet.com/overview-highlight-principal-design-features.html (date of access: 28.05.2024).

41. Troelsen A., Japikse P. Pro C# 10 With . NET 6: Foundational Principles and Practices in Programming. Apress L. P., 2022.
42. Albahari B., Albahari J. C# 7.0 in a Nutshell: The Definitive Reference. O'Reilly Media, 2017. 1088 p.

Додаток А

Програмний код

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace FaultTreeAnalysis
{
    public partial class Form1 : Form
    {
        private Random random = new Random();
        private Graphics graphics;
        private bool showBrakeSystemTree = true;
        private bool showEngineControlSystemTree = false;
        private bool showCoolingSystemTree = false;
        private bool showElectricalSystemTree = false;
        private bool showTransmissionSystemTree = false;

        public Form1()
        {
            InitializeComponent();
            LoadTreeNames();
        }

        private void LoadTreeNames()
        {
            listBox1.Items.AddRange(new string[]
            {
        "Гальмівна система",
        "Система управління двигуном",
        "Система охолодження двигуна",
        "Електрична система",
        "Трансмісія"
            });
            listBox1.SelectedIndex = 0;
        }

        private void listBox1_SelectedIndexChanged(object
sender, EventArgs e)
        {
            switch (listBox1.SelectedItem.ToString())
            {
                case "Гальмівна система":
                    showBrakeSystemTree = true;
                    showEngineControlSystemTree = false;
                    showCoolingSystemTree = false;
                    showElectricalSystemTree = false;
                    showTransmissionSystemTree = false;
                    break;
```

```
                case "Система управління двигуном":
                    showBrakeSystemTree = false;
                    showEngineControlSystemTree = true;
                    showCoolingSystemTree = false;
                    showElectricalSystemTree = false;
                    showTransmissionSystemTree = false;
                    break;
                case "Система охолодження двигуна":
                    showBrakeSystemTree = false;
                    showEngineControlSystemTree = false;
                    showCoolingSystemTree = true;
                    showElectricalSystemTree = false;
                    showTransmissionSystemTree = false;
                    break;
                case "Електрична система":
                    showBrakeSystemTree = false;
                    showEngineControlSystemTree = false;
                    showCoolingSystemTree = false;
                    showElectricalSystemTree = true;
                    showTransmissionSystemTree = false;
                    break;
                case "Трансмісія":
                    showBrakeSystemTree = false;
                    showEngineControlSystemTree = false;
                    showCoolingSystemTree = false;
                    showElectricalSystemTree = false;
                    showTransmissionSystemTree = true;
                    break;
            }
            buttonDrawTree_Click(sender, e);
        }


        private void buttonDrawTree_Click(object sender,
EventArgs e)
        {
            // Малювання дерева на панелі
            graphics = panel1.CreateGraphics();
            if (showBrakeSystemTree)
            {
                DrawBrakeSystemTree(graphics);
            }
            else if (showEngineControlSystemTree)
            {
                DrawEngineControlSystemTree(graphics);
            }
            else if (showCoolingSystemTree)
            {
                DrawCoolingSystemTree(graphics);
            }
            else if (showElectricalSystemTree)
            {
```

```
            DrawElectricalSystemTree(graphics);
        }
        else if (showTransmissionSystemTree)
        {
            DrawTransmissionSystemTree(graphics);
        }
    }



    private void DrawBrakeSystemTree(Graphics g)
    {
        // Налаштування графіки
        g.Clear(Color.White);
        Pen pen = new Pen(Color.Black, 2);
        Font font = new Font("Times New Roman", 10);
        Brush brush = Brushes.Black;

        // Координати та розміри елементів
        int centerX = panel1.Width / 2;
        int startY = 20;
        int nodeWidth = 170;
        int nodeHeight = 50;
        int verticalSpacing = 80;
        int horizontalSpacing = 200;

        // Малювання вузлів
        Rectangle rootRect = new Rectangle(centerX -
nodeWidth / 2, startY, nodeWidth, nodeHeight);
        g.DrawRectangle(pen, rootRect);
        g.DrawString("Загальне зниження надійності
гальмівної системи", font, brush, rootRect);

        // Вузли другого рівня
        Rectangle nodeB = new Rectangle(centerX - nodeWidth
/ 2 - horizontalSpacing / 2, startY + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
        g.DrawRectangle(pen, nodeB);
        g.DrawString("Втрата ефективності ABS", font, brush,
nodeB);

        Rectangle nodeC = new Rectangle(centerX - nodeWidth
/ 2 + horizontalSpacing / 2, startY + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
        g.DrawRectangle(pen, nodeC);
        g.DrawString("Зниження гальмівного зусилля", font,
brush, nodeC);

        Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 2 + 100,
nodeWidth, nodeHeight);
        g.DrawRectangle(pen, nodeD);
```

```
            g.DrawString("Обрив або замикання електропроводки",
font, brush, nodeD);

            // Вузли третього рівня для nodeB
            Rectangle nodeE = new Rectangle(nodeB.X -
horizontalSpacing, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeE);
            g.DrawString("Відмова датчиків\nшвидкості колеса",
font, brush, nodeE);

            Rectangle nodeF = new Rectangle(nodeB.X +
horizontalSpacing, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeF);
            g.DrawString("Електронний збій\nв блоці управління
ABS", font, brush, nodeF);

            // Вузли третього рівня для nodeC
            Rectangle nodeG = new Rectangle(nodeC.X -
horizontalSpacing, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeG);
            g.DrawString("Протікання\nгальмівної рідини", font,
brush, nodeG);

            Rectangle nodeH = new Rectangle(nodeC.X +
horizontalSpacing, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeH);
            g.DrawString("Механічний знос\nголовного гальмівного
циліндра", font, brush, nodeH);

            // Логічні оператори
            int operatorWidth = 50; // Ширина області для
логічного оператора
            int operatorHeight = 20; // Висота області для
логічного оператора

            Rectangle or1 = new Rectangle(centerX -
operatorWidth / 2, rootRect.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or2 = new Rectangle(nodeB.X + nodeWidth /
2 - operatorWidth / 2, nodeB.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or3 = new Rectangle(nodeC.X + nodeWidth /
2 - operatorWidth / 2, nodeC.Y + nodeHeight + 10, operatorWidth,
operatorHeight);

            // Лінії з'єднань
```

```
            g.DrawLine(pen, new Point(rootRect.X + nodeWidth /
2, rootRect.Y + nodeHeight), new Point(centerX, or1.Y +
operatorHeight / 2));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeB.X + nodeWidth / 2,
nodeB.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeC.X + nodeWidth / 2,
nodeC.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeD.X + nodeWidth / 2,
nodeD.Y));

            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
nodeB.Y + nodeHeight), new Point(nodeB.X + nodeWidth / 2, or2.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeE.X + nodeWidth / 2,
nodeE.Y));
            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeF.X + nodeWidth / 2,
nodeF.Y));

            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
nodeC.Y + nodeHeight), new Point(nodeC.X + nodeWidth / 2, or3.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
or3.Y + operatorHeight / 2), new Point(nodeG.X + nodeWidth / 2,
nodeG.Y));
            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
or3.Y + operatorHeight / 2), new Point(nodeH.X + nodeWidth / 2,
nodeH.Y));


            Brush whiteBrush = Brushes.White;

            g.FillRectangle(whiteBrush, or1);
            g.DrawRectangle(pen, or1);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or1));

            g.FillRectangle(whiteBrush, or2);
            g.DrawRectangle(pen, or2);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or2));

            g.FillRectangle(whiteBrush, or3);
            g.DrawRectangle(pen, or3);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or3));
        }
```

```
        private void DrawEngineControlSystemTree(Graphics g)
        {
            // Налаштування графіки
            g.Clear(Color.White);
            Pen pen = new Pen(Color.Black, 2);
            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Black;

            // Координати та розміри елементів
            int centerX = panel1.Width / 2;
            int startY = 20;
            int nodeWidth = 170;
            int nodeHeight = 50;
            int verticalSpacing = 80;
            int horizontalSpacing = 200;

            // Малювання вузлів
            Rectangle rootRect = new Rectangle(centerX -
nodeWidth / 2, startY, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, rootRect);
            g.DrawString("Загальне зниження надійності системи
управління двигуном", font, brush, rootRect);

            // Вузли другого рівня
            Rectangle nodeB = new Rectangle(centerX - nodeWidth
/ 2 - horizontalSpacing / 2, startY + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeB);
            g.DrawString("Зниження потужності двигуна", font,
brush, nodeB);

            Rectangle nodeC = new Rectangle(centerX - nodeWidth
/ 2 + horizontalSpacing / 2, startY + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeC);
            g.DrawString("Втрата зв'язку з датчиками", font,
brush, nodeC);

            Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 2 + 100,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeD);
            g.DrawString("Збій блоку управління двигуном", font,
brush, nodeD);

            // Вузли третього рівня для nodeB
            Rectangle nodeE = new Rectangle(nodeB.X -
horizontalSpacing, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeE);
            g.DrawString("Відмова датчика тиску мастила", font,
brush, nodeE);
```

```
            Rectangle nodeF = new Rectangle(nodeB.X +
horizontalSpacing, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeF);
            g.DrawString("Відмова датчика температури", font,
brush, nodeF);

            Rectangle nodeI = new Rectangle(nodeE.X, nodeE.Y +
nodeHeight + verticalSpacing, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeI);
            g.DrawString("Недостатній рівень мастила", font,
brush, nodeI);

            // Вузли третього рівня для nodeC
            Rectangle nodeG = new Rectangle(nodeC.X -
horizontalSpacing, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeG);
            g.DrawString("Відмова датчика положення колінвала",
font, brush, nodeG);

            Rectangle nodeH = new Rectangle(nodeC.X +
horizontalSpacing, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeH);
            g.DrawString("Відмова датчика витрати палива", font,
brush, nodeH);

            Rectangle nodeJ = new Rectangle(nodeH.X, nodeH.Y +
nodeHeight + verticalSpacing, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeJ);
            g.DrawString("Закінчився бензин", font, brush,
nodeJ);

            // Логічні оператори
            int operatorWidth = 50; // Ширина області для
логічного оператора
            int operatorHeight = 20; // Висота області для
логічного оператора

            Rectangle or1 = new Rectangle(centerX -
operatorWidth / 2, rootRect.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or2 = new Rectangle(nodeB.X + nodeWidth /
2 - operatorWidth / 2, nodeB.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle and1 = new Rectangle(nodeE.X + nodeWidth /
2 - operatorWidth / 2, nodeE.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
```

```
            Rectangle not1 = new Rectangle(nodeJ.X + nodeWidth /
2 - operatorWidth / 2, nodeH.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or3 = new Rectangle(nodeC.X + nodeWidth /
2 - operatorWidth / 2, nodeC.Y + nodeHeight + 10, operatorWidth,
operatorHeight);

            // Лінії з'єднань
            g.DrawLine(pen, new Point(rootRect.X + nodeWidth /
2, rootRect.Y + nodeHeight), new Point(centerX, or1.Y +
operatorHeight / 2));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeB.X + nodeWidth / 2,
nodeB.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeC.X + nodeWidth / 2,
nodeC.Y));

            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeD.X + nodeWidth / 2,
nodeD.Y));

            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
nodeB.Y + nodeHeight), new Point(nodeB.X + nodeWidth / 2, or2.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeE.X + nodeWidth / 2,
nodeE.Y));
            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeF.X + nodeWidth / 2,
nodeF.Y));

            g.DrawLine(pen, new Point(nodeE.X + nodeWidth / 2,
nodeE.Y + nodeHeight), new Point(nodeE.X + nodeWidth / 2, and1.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeE.X + nodeWidth / 2,
and1.Y + operatorHeight / 2), new Point(nodeI.X + nodeWidth / 2,
nodeI.Y));

            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
nodeC.Y + nodeHeight), new Point(nodeC.X + nodeWidth / 2, or3.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
or3.Y + operatorHeight / 2), new Point(nodeG.X + nodeWidth / 2,
nodeG.Y));
            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
or3.Y + operatorHeight / 2), new Point(nodeH.X + nodeWidth / 2,
nodeH.Y));

            g.DrawLine(pen, new Point(nodeH.X + nodeWidth / 2,
nodeH.Y + nodeHeight), new Point(nodeH.X + nodeWidth / 2, not1.Y
+ operatorHeight / 2));
```

```
            g.DrawLine(pen, new Point(nodeH.X + nodeWidth / 2,
not1.Y + operatorHeight / 2), new Point(nodeJ.X + nodeWidth / 2,
nodeJ.Y));


            Brush whiteBrush = Brushes.White;

            g.FillRectangle(whiteBrush, or1);
            g.DrawRectangle(pen, or1);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or1));

            g.FillRectangle(whiteBrush, or2);
            g.DrawRectangle(pen, or2);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or2));

            g.FillRectangle(whiteBrush, and1);
            g.DrawRectangle(pen, and1);
            g.DrawString("AND", font, brush, CenterText("AND",
font, and1));

            g.FillRectangle(whiteBrush, not1);
            g.DrawRectangle(pen, not1);
            g.DrawString("NOT", font, brush, CenterText("NOT",
font, not1));

            g.FillRectangle(whiteBrush, or3);
            g.DrawRectangle(pen, or3);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or3));
        }

        private void DrawCoolingSystemTree(Graphics g)
        {
            // Налаштування графіки
            g.Clear(Color.White);
            Pen pen = new Pen(Color.Black, 2);
            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Black;

            // Координати та розміри елементів
            int centerX = panel1.Width / 2;
            int startY = 20;

            int nodeWidth = 150;
            int nodeHeight = 50;
            int verticalSpacing = 60;
            int horizontalSpacing = 160;


            // Малювання вузлів
```

```
            Rectangle rootRect = new Rectangle(centerX -
nodeWidth / 2, startY, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, rootRect);
            g.DrawString("Перегрів двигуна", font, brush,
rootRect);

            // Вузли другого рівня
            Rectangle nodeB = new Rectangle(centerX - nodeWidth
- horizontalSpacing / 2, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeB);
            g.DrawString("Відмова вентилятора", font, brush,
nodeB);

            Rectangle nodeC = new Rectangle(centerX +
horizontalSpacing / 2, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeC);
            g.DrawString("Утечка охолоджувальної рідини", font,
brush, nodeC);

            Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 2, nodeWidth,
nodeHeight);
            g.DrawRectangle(pen, nodeD);
            g.DrawString("Засорення радіатора", font, brush,
nodeD);

            // Вузли третього рівня для nodeD
            Rectangle nodeE = new Rectangle(nodeD.X -
horizontalSpacing / 2 - nodeWidth / 2, nodeD.Y + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeE);
            g.DrawString("Відмова термостата", font, brush,
nodeE);

            Rectangle nodeF = new Rectangle(nodeD.X +
horizontalSpacing / 2 - nodeWidth / 2, nodeD.Y + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeF);
            g.DrawString("Механічний збій помпи", font, brush,
nodeF);

            // Логічні оператори
            int operatorWidth = 50; // Ширина області для
логічного оператора
            int operatorHeight = 20; // Висота області для
логічного оператора

            Rectangle or1 = new Rectangle(centerX -
operatorWidth / 2, rootRect.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
```

```
            Rectangle or2 = new Rectangle(nodeD.X + nodeWidth /
2 - operatorWidth / 2, nodeD.Y + nodeHeight + 10, operatorWidth,
operatorHeight);

            // Лінії з'єднань
            g.DrawLine(pen, new Point(rootRect.X + nodeWidth /
2, rootRect.Y + nodeHeight), new Point(centerX, or1.Y +
operatorHeight / 2));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeB.X + nodeWidth / 2,
nodeB.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeC.X + nodeWidth / 2,
nodeC.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeD.X + nodeWidth / 2,
nodeD.Y));

            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
nodeD.Y + nodeHeight), new Point(nodeD.X + nodeWidth / 2, or2.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeE.X + nodeWidth / 2,
nodeE.Y));
            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeF.X + nodeWidth / 2,
nodeF.Y));

            // Отображение логических операторов на переднем
плане
            Brush whiteBrush = Brushes.White;

            g.FillRectangle(whiteBrush, or1);
            g.DrawRectangle(pen, or1);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or1));

            g.FillRectangle(whiteBrush, or2);
            g.DrawRectangle(pen, or2);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or2));
        }

        private void DrawElectricalSystemTree(Graphics g)
        {
            // Налаштування графіки
            g.Clear(Color.White);
            Pen pen = new Pen(Color.Black, 2);
            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Black;

            // Координати та розміри елементів
```

```
            int centerX = panel1.Width / 2;
            int startY = 20;
            int nodeWidth = 150;
            int nodeHeight = 50;
            int verticalSpacing = 60;
            int horizontalSpacing = 160;

            // Малювання вузлів
            Rectangle rootRect = new Rectangle(centerX -
nodeWidth / 2, startY, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, rootRect);
            g.DrawString("Повна відмова електричної системи",
font, brush, rootRect);

            // Вузли другого рівня
            Rectangle nodeB = new Rectangle(centerX - nodeWidth
- horizontalSpacing, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeB);
            g.DrawString("Відмова акумулятора", font, brush,
nodeB);

            Rectangle nodeC = new Rectangle(centerX +
horizontalSpacing, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeC);
            g.DrawString("Коротке замикання", font, brush,
nodeC);


            Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 3 + 30,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeD);
            g.DrawString("Пошкодження проводки", font, brush,
nodeD);

            // Вузли третього рівня для nodeB
            Rectangle nodeE = new Rectangle(nodeB.X -
horizontalSpacing / 2, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeE);
            g.DrawString("Розряд акумулятора", font, brush,
nodeE);

            Rectangle nodeF = new Rectangle(nodeB.X +
horizontalSpacing - 60, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight); // вправо на 40
            g.DrawRectangle(pen, nodeF);
            g.DrawString("Старий акумулятор", font, brush,
nodeF);
```

```
            // Вузли третього рівня для nodeC
            Rectangle nodeG = new Rectangle(nodeC.X -
horizontalSpacing + 60, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight); // влево на 40
            g.DrawRectangle(pen, nodeG);
            g.DrawString("Перегрів проводів", font, brush,
nodeG);

            Rectangle nodeH = new Rectangle(nodeC.X +
horizontalSpacing / 2, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeH);
            g.DrawString("Пошкодження ізоляції", font, brush,
nodeH);

            // Логічні оператори
            int operatorWidth = 50; // Ширина області для
логічного оператора
            int operatorHeight = 20; // Висота області для
логічного оператора

            Rectangle or1 = new Rectangle(centerX -
operatorWidth / 2, rootRect.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or2 = new Rectangle(nodeB.X + nodeWidth /
2 - operatorWidth / 2, nodeB.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or3 = new Rectangle(nodeC.X + nodeWidth /
2 - operatorWidth / 2, nodeC.Y + nodeHeight + 10, operatorWidth,
operatorHeight);

            // Лінії з'єднань
            g.DrawLine(pen, new Point(rootRect.X + nodeWidth /
2, rootRect.Y + nodeHeight), new Point(centerX, or1.Y +
operatorHeight / 2));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeB.X + nodeWidth / 2,
nodeB.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeC.X + nodeWidth / 2,
nodeC.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeD.X + nodeWidth / 2,
nodeD.Y));

            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
nodeB.Y + nodeHeight), new Point(nodeB.X + nodeWidth / 2, or2.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeE.X + nodeWidth / 2,
nodeE.Y));
```

```
            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeF.X + nodeWidth / 2,
nodeF.Y));

            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
nodeC.Y + nodeHeight), new Point(nodeC.X + nodeWidth / 2, or3.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
or3.Y + operatorHeight / 2), new Point(nodeG.X + nodeWidth / 2,
nodeG.Y));
            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
or3.Y + operatorHeight / 2), new Point(nodeH.X + nodeWidth / 2,
nodeH.Y));

            Brush whiteBrush = Brushes.White;

            g.FillRectangle(whiteBrush, or1);
            g.DrawRectangle(pen, or1);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or1));

            g.FillRectangle(whiteBrush, or2);
            g.DrawRectangle(pen, or2);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or2));

            g.FillRectangle(whiteBrush, or3);
            g.DrawRectangle(pen, or3);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or3));
        }

        private void DrawTransmissionSystemTree(Graphics g)
        {
            // Налаштування графіки
            g.Clear(Color.White);
            Pen pen = new Pen(Color.Black, 2);
            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Black;

            // Координати та розміри елементів
            int centerX = panel1.Width / 2;
            int startY = 20;
            int nodeWidth = 150;
            int nodeHeight = 50;
            int verticalSpacing = 60;
            int horizontalSpacing = 160;

            // Малювання вузлів
            Rectangle rootRect = new Rectangle(centerX -
nodeWidth / 2, startY, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, rootRect);
```

```
            g.DrawString("Відмова трансмісії", font, brush,
rootRect);

            // Вузли другого рівня
            Rectangle nodeB = new Rectangle(centerX - nodeWidth
- horizontalSpacing, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeB);
            g.DrawString("Знос зчеплення", font, brush, nodeB);

            Rectangle nodeC = new Rectangle(centerX +
horizontalSpacing, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeC);
            g.DrawString("Поломка коробки передач", font, brush,
nodeC);

            Rectangle nodeD = new Rectangle(centerX - nodeWidth
- horizontalSpacing / 2 + 155, startY + (nodeHeight +
verticalSpacing) * 2 - 53, nodeWidth, nodeHeight); // вправо на
30, вверх на 30
            g.DrawRectangle(pen, nodeD);
            g.DrawString("Низький рівень трансмісійної рідини",
font, brush, nodeD);

            Rectangle nodeE = new Rectangle(centerX +
horizontalSpacing / 2 + 80, startY + (nodeHeight +
verticalSpacing) * 2-200, nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeE);
            g.DrawString("Пошкодження карданного вала", font,
brush, nodeE);

            // Вузли третього рівня для nodeB
            Rectangle nodeF = new Rectangle(nodeB.X -
horizontalSpacing / 2, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeF);
            g.DrawString("Невідповідне зчеплення", font, brush,
nodeF);

            Rectangle nodeG = new Rectangle(nodeB.X +
horizontalSpacing / 2, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeG);
            g.DrawString("Низька якість зчеплення", font, brush,
nodeG);

            // Вузли третього рівня для nodeC
            Rectangle nodeH = new Rectangle(nodeC.X -
horizontalSpacing / 2, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeH);
```

```
            g.DrawString("Знос шестерень", font, brush, nodeH);

            Rectangle nodeI = new Rectangle(nodeC.X +
horizontalSpacing / 2, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeI);
            g.DrawString("Поломка валів", font, brush, nodeI);

            // Вузли третього рівня для nodeD
            Rectangle nodeJ = new Rectangle(nodeD.X -
horizontalSpacing / 2, nodeD.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeJ);
            g.DrawString("Протікання мастила", font, brush,
nodeJ);

            Rectangle nodeK = new Rectangle(nodeD.X +
horizontalSpacing / 2, nodeD.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            g.DrawRectangle(pen, nodeK);
            g.DrawString("Неправильний рівень мастила", font,
brush, nodeK);

            // Логічні оператори
            int operatorWidth = 50;
            int operatorHeight = 20;

            Rectangle or1 = new Rectangle(centerX -
operatorWidth / 2, rootRect.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or2 = new Rectangle(nodeB.X + nodeWidth /
2 - operatorWidth / 2, nodeB.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or3 = new Rectangle(nodeC.X + nodeWidth /
2 - operatorWidth / 2, nodeC.Y + nodeHeight + 10, operatorWidth,
operatorHeight);
            Rectangle or4 = new Rectangle(nodeD.X + nodeWidth /
2 - operatorWidth / 2, nodeD.Y + nodeHeight + 10, operatorWidth,
operatorHeight);

            // Лінії з'єднань
            g.DrawLine(pen, new Point(rootRect.X + nodeWidth /
2, rootRect.Y + nodeHeight), new Point(centerX, or1.Y +
operatorHeight / 2));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeB.X + nodeWidth / 2,
nodeB.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeC.X + nodeWidth / 2,
nodeC.Y));
```

```
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeD.X + nodeWidth / 2,
nodeD.Y));
            g.DrawLine(pen, new Point(centerX, or1.Y +
operatorHeight / 2), new Point(nodeE.X + nodeWidth / 2,
nodeE.Y));

            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
nodeB.Y + nodeHeight), new Point(nodeB.X + nodeWidth / 2, or2.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeF.X + nodeWidth / 2,
nodeF.Y));
            g.DrawLine(pen, new Point(nodeB.X + nodeWidth / 2,
or2.Y + operatorHeight / 2), new Point(nodeG.X + nodeWidth / 2,
nodeG.Y));

            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
nodeC.Y + nodeHeight), new Point(nodeC.X + nodeWidth / 2, or3.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
or3.Y + operatorHeight / 2), new Point(nodeH.X + nodeWidth / 2,
nodeH.Y));
            g.DrawLine(pen, new Point(nodeC.X + nodeWidth / 2,
or3.Y + operatorHeight / 2), new Point(nodeI.X + nodeWidth / 2,
nodeI.Y));

            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
nodeD.Y + nodeHeight), new Point(nodeD.X + nodeWidth / 2, or4.Y
+ operatorHeight / 2));
            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
or4.Y + operatorHeight / 2), new Point(nodeJ.X + nodeWidth / 2,
nodeJ.Y));
            g.DrawLine(pen, new Point(nodeD.X + nodeWidth / 2,
or4.Y + operatorHeight / 2), new Point(nodeK.X + nodeWidth / 2,
nodeK.Y));

            // Отображение логических операторов на переднем
плане
            Brush whiteBrush = Brushes.White;

            g.FillRectangle(whiteBrush, or1);
            g.DrawRectangle(pen, or1);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or1));

            g.FillRectangle(whiteBrush, or2);
            g.DrawRectangle(pen, or2);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or2));

            g.FillRectangle(whiteBrush, or3);
```

```
            g.DrawRectangle(pen, or3);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or3));

            g.FillRectangle(whiteBrush, or4);
            g.DrawRectangle(pen, or4);
            g.DrawString("OR", font, brush, CenterText("OR",
font, or4));
        }

        private PointF CenterText(string text, Font font,
Rectangle rect)
        {
            SizeF textSize = graphics.MeasureString(text, font);
            float x = rect.X + (rect.Width - textSize.Width) /
2;
            float y = rect.Y + (rect.Height - textSize.Height) /
2;
            return new PointF(x, y);
        }

        private void MonteCarloSimulation(int iterations)
        {
            double failureProbability = 0;
            for (int i = 0; i < iterations; i++)
            {
                if (SimulateFailure())
                {
                    failureProbability += 1.0 / iterations;
                }
            }
            textBoxMonteCarloSimulation.Text =
$"{failureProbability:P2}";
        }

        private bool SimulateFailure()
        {
            if (showBrakeSystemTree)
            {
                return SimulateBrakeSystemFailure();
            }
            else if (showEngineControlSystemTree)
            {
                return SimulateEngineControlSystemFailure();
            }
            else if (showCoolingSystemTree)
            {
                return SimulateCoolingSystemFailure();
            }
            else if (showElectricalSystemTree)
            {
                return SimulateElectricalSystemFailure();
```

```
            }
            else if (showTransmissionSystemTree)
            {
                return SimulateTransmissionSystemFailure();
            }
            return false;
        }


        private bool SimulateBrakeSystemFailure()
        {
            // Ймовірності відмов
            double pE = 0.015; // Відмова датчиків швидкості
колеса
            double pG = 0.008; // Протікання гальмівної рідини
            double pF = 0.002; // Електронний збій в блоці
управління ABS
            double pH = 0.004; // Механічний знос головного
гальмівного циліндра
            double pD = 0.005; // Обрив або замикання
електропроводки

            // Генерація випадкових сценаріїв відмов
            bool failureE = random.NextDouble() < pE;
            bool failureG = random.NextDouble() < pG;
            bool failureF = random.NextDouble() < pF;
            bool failureH = random.NextDouble() < pH;
            bool failureD = random.NextDouble() < pD;

            // Логічні оператори
            bool lossOfABS = failureE || failureF;
            bool reductionInBrakingForce = failureG || failureH;
            bool overallFailure = lossOfABS ||
reductionInBrakingForce || failureD;

            return overallFailure;
        }

        private bool SimulateEngineControlSystemFailure()
        {
            // Ймовірності відмов
            double pE = 0.010; // Відмова датчика тиску мастила
            double pG = 0.012; // Відмова датчика положення
колінвала
            double pF = 0.003; // Відмова датчика температури
            double pH = 0.007; // Відмова датчика витрати палива
            double pI = 0.005; // Недостатній рівень мастила
            double pJ = 0.006; // Закінчився бензин
            double pD = 0.002; // Збій блоку управління двигуном

            // Генерація випадкових сценаріїв відмов
            bool failureE = random.NextDouble() < pE;
```

```
            bool failureG = random.NextDouble() < pG;
            bool failureF = random.NextDouble() < pF;
            bool failureH = random.NextDouble() < pH;
            bool failureI = random.NextDouble() < pI;
            bool failureJ = random.NextDouble() < pJ;
            bool failureD = random.NextDouble() < pD;

            // Логічні оператори
            bool oilPressureFailure = failureE && failureI;
            bool powerReduction = oilPressureFailure ||
failureF;
            bool fuelSensorFailure = failureH && !failureJ;
            bool sensorFailure = failureG || fuelSensorFailure;
            bool overallFailure = powerReduction ||
sensorFailure || failureD;

            return overallFailure;
        }

        private bool SimulateCoolingSystemFailure()
        {
            // Ймовірності відмов
            double pB = 0.010; // Відмова вентилятора
            double pC = 0.012; // Утечка охолоджувальної рідини
            double pD = 0.003; // Засорення радіатора
            double pE = 0.005; // Відмова термостата
            double pF = 0.007; // Механічний збій помпи

            // Генерація випадкових сценаріїв відмов
            bool failureB = random.NextDouble() < pB;
            bool failureC = random.NextDouble() < pC;
            bool failureD = random.NextDouble() < pD;
            bool failureE = random.NextDouble() < pE;
            bool failureF = random.NextDouble() < pF;

            // Логічні оператори
            bool radiatorClogging = failureE || failureF; //
            bool overallFailure = failureB || failureC ||
failureD || radiatorClogging;

            return overallFailure;
        }

        private bool SimulateElectricalSystemFailure()
        {
            // Ймовірності відмов
            double pB = 0.010; // Відмова акумулятора
            double pC = 0.012; // Коротке замикання
            double pD = 0.003; // Пошкодження проводки
            double pE = 0.005; // Розряд акумулятора
            double pF = 0.007; // Старий акумулятор
            double pG = 0.006; // Перегрів проводів
```

```
            double pH = 0.004; // Пошкодження ізоляції

            // Генерація випадкових сценаріїв відмов
            bool failureB = random.NextDouble() < pB;
            bool failureC = random.NextDouble() < pC;
            bool failureD = random.NextDouble() < pD;
            bool failureE = random.NextDouble() < pE;
            bool failureF = random.NextDouble() < pF;
            bool failureG = random.NextDouble() < pG;
            bool failureH = random.NextDouble() < pH;

            // Логічні оператори
            bool batteryFailure = failureE || failureF; //
Використовуємо вентиль OR для розряду та старого акумулятора
            bool shortCircuit = failureG || failureH; //
Використовуємо вентиль OR для перегріву проводів та пошкодження
ізоляції
            bool overallFailure = failureB || failureC ||
failureD || batteryFailure || shortCircuit;

            return overallFailure;
        }

        private bool SimulateTransmissionSystemFailure()
        {
            // Ймовірності відмов
            double pB = 0.010; // Знос зчеплення
            double pC = 0.012; // Поломка коробки передач
            double pD = 0.003; // Низький рівень трансмісійної
рідини
            double pE = 0.005; // Пошкодження карданного вала
            double pF = 0.004; // Невідповідне зчеплення
            double pG = 0.006; // Низька якість зчеплення
            double pH = 0.007; // Знос шестерень
            double pI = 0.008; // Поломка валів
            double pJ = 0.009; // Протікання мастила
            double pK = 0.002; // Неправильний рівень мастила

            // Генерація випадкових сценаріїв відмов
            bool failureB = random.NextDouble() < pB;
            bool failureC = random.NextDouble() < pC;
            bool failureD = random.NextDouble() < pD;
            bool failureE = random.NextDouble() < pE;
            bool failureF = random.NextDouble() < pF;
            bool failureG = random.NextDouble() < pG;
            bool failureH = random.NextDouble() < pH;
            bool failureI = random.NextDouble() < pI;
            bool failureJ = random.NextDouble() < pJ;
            bool failureK = random.NextDouble() < pK;

            // Логічні оператори
```

```
            bool clutchFailure = failureF || failureG; //
Використовуємо вентиль OR
            bool gearboxFailure = failureH || failureI; //
Використовуємо вентиль OR
            bool oilLevelFailure = failureJ || failureK; //
Використовуємо вентиль OR
            bool overallFailure = clutchFailure ||
gearboxFailure || oilLevelFailure || failureB || failureC ||
failureD || failureE;

            return overallFailure;
        }


        private void buttonMonteCarlo_Click(object sender,
EventArgs e)
        {

MonteCarloSimulation(Convert.ToInt32(textBox1.Text));
        }

        private void buttonShowProbabilities_Click(object
sender, EventArgs e)
        {
            if (showBrakeSystemTree)
            {
                ShowBrakeSystemProbabilities(graphics);
            }
            else if (showEngineControlSystemTree)
            {
                ShowEngineControlSystemProbabilities(graphics);
            }
            else if (showCoolingSystemTree)
            {
                ShowCoolingSystemProbabilities(graphics);
            }
            else if (showElectricalSystemTree)
            {
                ShowElectricalSystemProbabilities(graphics);
            }
            else if (showTransmissionSystemTree)
            {
                ShowTransmissionSystemProbabilities(graphics);
            }
        }


        private void ShowBrakeSystemProbabilities(Graphics g)
        {
            if (g == null)
            {
                return;
```

```
            }

            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Red;

            // Ймовірності відмов
            double pE = 0.015;
            double pG = 0.008;
            double pF = 0.002;
            double pH = 0.004;
            double pD = 0.005;

            // Вузли другого рівня
            int centerX = panel1.Width / 2;
            int startY = 20;
            int nodeWidth = 170;
            int nodeHeight = 50;
            int verticalSpacing = 80;
            int horizontalSpacing = 200;

            Rectangle nodeB = new Rectangle(centerX - nodeWidth
/ 2 - horizontalSpacing / 2, startY + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            Rectangle nodeC = new Rectangle(centerX - nodeWidth
/ 2 + horizontalSpacing / 2, startY + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 2 + 100,
nodeWidth, nodeHeight);

            // Вузли третього рівня для nodeB
            Rectangle nodeE = new Rectangle(nodeB.X -
horizontalSpacing, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeF = new Rectangle(nodeB.X +
horizontalSpacing, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);

            // Вузли третього рівня для nodeC
            Rectangle nodeG = new Rectangle(nodeC.X -
horizontalSpacing, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeH = new Rectangle(nodeC.X +
horizontalSpacing, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);

            // Малювання ймовірностей
            g.DrawString($"{pE:P1}", font, brush, nodeE.X +
nodeWidth / 2 - 20, nodeE.Y - 20);
            g.DrawString($"{pG:P1}", font, brush, nodeG.X +
nodeWidth / 2 - 20, nodeG.Y - 20);
```

```
            g.DrawString($"{pF:P1}", font, brush, nodeF.X +
nodeWidth / 2 - 20, nodeF.Y - 20);
            g.DrawString($"{pH:P1}", font, brush, nodeH.X +
nodeWidth / 2 - 20, nodeH.Y - 20);
            g.DrawString($"{pD:P1}", font, brush, nodeD.X +
nodeWidth / 2 - 20, nodeD.Y - 20);
        }

        private void
ShowEngineControlSystemProbabilities(Graphics g)
        {
            if (g == null)
            {
                return;
            }

            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Red;

            // Ймовірності відмов
            double pE = 0.010;
            double pG = 0.012;
            double pF = 0.003;
            double pH = 0.007;
            double pI = 0.005;
            double pJ = 0.006;
            double pD = 0.002;

            // Вузли другого рівня
            int centerX = panel1.Width / 2;
            int startY = 20;
            int nodeWidth = 170;
            int nodeHeight = 50;
            int verticalSpacing = 80;
            int horizontalSpacing = 200;

            Rectangle nodeB = new Rectangle(centerX - nodeWidth
/ 2 - horizontalSpacing / 2, startY + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            Rectangle nodeC = new Rectangle(centerX - nodeWidth
/ 2 + horizontalSpacing / 2, startY + nodeHeight +
verticalSpacing, nodeWidth, nodeHeight);
            Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 2 + 100,
nodeWidth, nodeHeight);

            // Вузли третього рівня для nodeB
            Rectangle nodeE = new Rectangle(nodeB.X -
horizontalSpacing, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
```

```
            Rectangle nodeF = new Rectangle(nodeB.X +
horizontalSpacing, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeI = new Rectangle(nodeE.X, nodeE.Y +
nodeHeight + verticalSpacing, nodeWidth, nodeHeight);

            // Вузли третього рівня для nodeC
            Rectangle nodeG = new Rectangle(nodeC.X -
horizontalSpacing, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeH = new Rectangle(nodeC.X +
horizontalSpacing, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeJ = new Rectangle(nodeH.X, nodeH.Y +
nodeHeight + verticalSpacing, nodeWidth, nodeHeight);

            // Малювання ймовірностей
            g.DrawString($"{pE:P1}", font, brush, nodeE.X +
nodeWidth / 2 - 20, nodeE.Y - 20);
            g.DrawString($"{pG:P1}", font, brush, nodeG.X +
nodeWidth / 2 - 20, nodeG.Y - 20);
            g.DrawString($"{pF:P1}", font, brush, nodeF.X +
nodeWidth / 2 - 20, nodeF.Y - 20);
            g.DrawString($"{pH:P1}", font, brush, nodeH.X +
nodeWidth / 2 - 20, nodeH.Y - 20);
            g.DrawString($"{pI:P1}", font, brush, nodeI.X +
nodeWidth / 2 - 20, nodeI.Y - 20);
            g.DrawString($"{pJ:P1}", font, brush, nodeJ.X +
nodeWidth / 2 - 20, nodeJ.Y - 20);
            g.DrawString($"{pD:P1}", font, brush, nodeD.X +
nodeWidth / 2 - 20, nodeD.Y - 20);
        }

        private void ShowCoolingSystemProbabilities(Graphics g)
        {
            if (g == null)
            {
                return;
            }

            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Red;

            // Ймовірності відмов
            double pB = 0.010;
            double pC = 0.012;
            double pD = 0.003;
            double pE = 0.005;
            double pF = 0.007;

            // Вузли другого рівня
            int centerX = panel1.Width / 2;
```

```
            int startY = 20;
            int nodeWidth = 150;
            int nodeHeight = 50;
            int verticalSpacing = 60;
            int horizontalSpacing = 160;

            Rectangle nodeB = new Rectangle(centerX - nodeWidth
- horizontalSpacing / 2, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeC = new Rectangle(centerX +
horizontalSpacing / 2, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 2, nodeWidth,
nodeHeight);
            Rectangle nodeE = new Rectangle(centerX - nodeWidth
- horizontalSpacing / 2, startY + (nodeHeight + verticalSpacing)
* 3, nodeWidth, nodeHeight);
            Rectangle nodeF = new Rectangle(centerX +
horizontalSpacing / 2, startY + (nodeHeight + verticalSpacing) *
3, nodeWidth, nodeHeight);

            // Малювання ймовірностей
            g.DrawString($"{pB:P1}", font, brush, nodeB.X +
nodeWidth / 2 - 20, nodeB.Y - 20);
            g.DrawString($"{pC:P1}", font, brush, nodeC.X +
nodeWidth / 2 - 20, nodeC.Y - 20);
            g.DrawString($"{pD:P1}", font, brush, nodeD.X +
nodeWidth / 2 - 20, nodeD.Y - 20);
            g.DrawString($"{pE:P1}", font, brush, nodeE.X +
nodeWidth / 2 - 20, nodeE.Y - 20);
            g.DrawString($"{pF:P1}", font, brush, nodeF.X +
nodeWidth / 2 - 20, nodeF.Y - 20);
        }

        private void ShowElectricalSystemProbabilities(Graphics
g)
        {
            if (g == null)
            {
                return;
            }

            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Red;

            // Ймовірності відмов
            double pB = 0.010;
            double pC = 0.012;
            double pD = 0.003;
            double pE = 0.005;
            double pF = 0.007;
```

```
            double pG = 0.006;
            double pH = 0.004;

            // Вузли другого рівня
            int centerX = panel1.Width / 2;
            int startY = 20;
            int nodeWidth = 150;
            int nodeHeight = 50;
            int verticalSpacing = 60;
            int horizontalSpacing = 160;

            Rectangle nodeB = new Rectangle(centerX - nodeWidth
- horizontalSpacing / 2, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeC = new Rectangle(centerX +
horizontalSpacing / 2, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeD = new Rectangle(centerX - nodeWidth
/ 2, startY + (nodeHeight + verticalSpacing) * 2, nodeWidth,
nodeHeight);
            Rectangle nodeE = new Rectangle(nodeB.X -
horizontalSpacing / 2, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeF = new Rectangle(nodeB.X +
horizontalSpacing / 2, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeG = new Rectangle(nodeC.X -
horizontalSpacing / 2, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeH = new Rectangle(nodeC.X +
horizontalSpacing / 2, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);

            // Малювання ймовірностей
            g.DrawString($"{pB:P1}", font, brush, nodeB.X +
nodeWidth / 2 - 20, nodeB.Y - 20);
            g.DrawString($"{pC:P1}", font, brush, nodeC.X +
nodeWidth / 2 - 20, nodeC.Y - 20);
            g.DrawString($"{pD:P1}", font, brush, nodeD.X +
nodeWidth / 2 - 20, nodeD.Y - 20);
            g.DrawString($"{pE:P1}", font, brush, nodeE.X +
nodeWidth / 2 - 20, nodeE.Y - 20);
            g.DrawString($"{pF:P1}", font, brush, nodeF.X +
nodeWidth / 2 - 20, nodeF.Y - 20);
            g.DrawString($"{pG:P1}", font, brush, nodeG.X +
nodeWidth / 2 - 20, nodeG.Y - 20);
            g.DrawString($"{pH:P1}", font, brush, nodeH.X +
nodeWidth / 2 - 20, nodeH.Y - 20);
        }

        private void
ShowTransmissionSystemProbabilities(Graphics g)
```

```
        {
            if (g == null)
            {
                return;
            }

            Font font = new Font("Times New Roman", 10);
            Brush brush = Brushes.Red;

            // Ймовірності відмов
            double pB = 0.010;
            double pC = 0.012;
            double pD = 0.003;
            double pE = 0.005;
            double pF = 0.004;
            double pG = 0.006;
            double pH = 0.007;
            double pI = 0.008;
            double pJ = 0.009;
            double pK = 0.002;

            // Вузли другого рівня
            int centerX = panel1.Width / 2;
            int startY = 20;
            int nodeWidth = 150;
            int nodeHeight = 50;
            int verticalSpacing = 60;
            int horizontalSpacing = 160;

            Rectangle nodeB = new Rectangle(centerX - nodeWidth
- horizontalSpacing, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeC = new Rectangle(centerX +
horizontalSpacing, startY + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeD = new Rectangle(centerX - nodeWidth
- horizontalSpacing / 2 + 50, startY + (nodeHeight +
verticalSpacing) * 2 + 100, nodeWidth, nodeHeight);
            Rectangle nodeE = new Rectangle(centerX +
horizontalSpacing / 2, startY + (nodeHeight + verticalSpacing) *
2, nodeWidth, nodeHeight);

            Rectangle nodeF = new Rectangle(nodeB.X -
horizontalSpacing / 2, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeG = new Rectangle(nodeB.X +
horizontalSpacing / 2, nodeB.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);

            Rectangle nodeH = new Rectangle(nodeC.X -
horizontalSpacing / 2, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
```

```
            Rectangle nodeI = new Rectangle(nodeC.X +
horizontalSpacing / 2, nodeC.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);

            Rectangle nodeJ = new Rectangle(nodeD.X -
horizontalSpacing / 2, nodeD.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);
            Rectangle nodeK = new Rectangle(nodeD.X +
horizontalSpacing / 2, nodeD.Y + nodeHeight + verticalSpacing,
nodeWidth, nodeHeight);

            // Малювання ймовірностей
            g.DrawString($"{pB:P1}", font, brush, nodeB.X +
nodeWidth / 2 - 20, nodeB.Y - 20);
            g.DrawString($"{pC:P1}", font, brush, nodeC.X +
nodeWidth / 2 - 20, nodeC.Y - 20);
            g.DrawString($"{pD:P1}", font, brush, nodeD.X +
nodeWidth / 2 - 20, nodeD.Y - 20);
            g.DrawString($"{pE:P1}", font, brush, nodeE.X +
nodeWidth / 2 - 20, nodeE.Y - 20);
            g.DrawString($"{pF:P1}", font, brush, nodeF.X +
nodeWidth / 2 - 20, nodeF.Y - 20);
            g.DrawString($"{pG:P1}", font, brush, nodeG.X +
nodeWidth / 2 - 20, nodeG.Y - 20);
            g.DrawString($"{pH:P1}", font, brush, nodeH.X +
nodeWidth / 2 - 20, nodeH.Y - 20);
            g.DrawString($"{pI:P1}", font, brush, nodeI.X +
nodeWidth / 2 - 20, nodeI.Y - 20);
            g.DrawString($"{pJ:P1}", font, brush, nodeJ.X +
nodeWidth / 2 - 20, nodeJ.Y - 20);
            g.DrawString($"{pK:P1}", font, brush, nodeK.X +
nodeWidth / 2 - 20, nodeK.Y - 20);
        }



        private void buttonSwitchTree_Click(object sender,
EventArgs e)
        {
            if (showBrakeSystemTree)
            {
                showBrakeSystemTree = false;
                showEngineControlSystemTree = true;
                showCoolingSystemTree = false;
                showElectricalSystemTree = false;
                showTransmissionSystemTree = false;
            }
            else if (showEngineControlSystemTree)
            {
                showEngineControlSystemTree = false;
                showCoolingSystemTree = true;
                showBrakeSystemTree = false;
```

```
                showElectricalSystemTree = false;
                showTransmissionSystemTree = false;
            }
            else if (showCoolingSystemTree)
            {
                showCoolingSystemTree = false;
                showElectricalSystemTree = true;
                showBrakeSystemTree = false;
                showEngineControlSystemTree = false;
                showTransmissionSystemTree = false;
            }
            else if (showElectricalSystemTree)
            {
                showElectricalSystemTree = false;
                showTransmissionSystemTree = true;
                showBrakeSystemTree = false;
                showEngineControlSystemTree = false;
                showCoolingSystemTree = false;
            }
            else if (showTransmissionSystemTree)
            {
                showTransmissionSystemTree = false;
                showBrakeSystemTree = true;
                showEngineControlSystemTree = false;
                showCoolingSystemTree = false;
                showElectricalSystemTree = false;
            }

            buttonDrawTree_Click(sender, e);
        }

        private void buttonExit_Click(object sender, EventArgs
e)
        {
            Close();
        }
    }
}
```