# Анотація

Магістерська кваліфікаційна робота присвячена розробці засобів проведення автоматизованого тестування програмного забезпечення вбудованих систем, розроблених у інтегрованому середовищі програмування uVision Keil.

Проект включає розробку алгоритму автоматизованого тестування програмного забезпечення вбудованих систем на основі проведеного аналізу існуючих методів, його програмну реалізацію та проведення дослідження розробленого засобу.

З врахуванням аналізу існуючих засобів автоматизованого тестування ПЗ вбудованих систем і факту про неможливість проведення тестування, за допомогою наявних засобі, програмного забезпечення вбудованої системи на апаратному забезпеченні без внесення змін у код, подано алгоритм автоматизованого тестування, для роботи якого не потрібно вносити додаткових змін у вихідний код вбудованої системи.

Проведено валідацію розробленого засобу тестування ПЗ вбудованої системи за допомогою функції затримки, основаній на апаратному таймері, та цифрового осцилографа.

Проведено дослідження на аналізі порівняння з засобом-аналогом, в основі якого лежить статичний аналізатор коду, яке показало, що розроблений засіб є точнішим за його аналог.

Основною метою магістерської кваліфікаційної роботи є розробка методу автоматизованого тестування тривалості виконання коду вбудованої системи, його програмна реалізація та проведення дослідження на реальних вбудованих системах.

Об’єкт дослідження – процес автоматизованого генерування тестових випадків та виконання тестування програмного забезпечення вбудованих систем.

Предмет дослідження – моделі та алгоритми автоматизованого тестування програмного забезпечення вбудованих систем.

Загальний обсяг роботи 80 сторінок.

# ABSTRACT

The master's qualification work is devoted to the research and development of tools for automated software testing of embedded systems developed in the integrated programming environment uVision Keil.

The project includes the development of an algorithm for automated software testing of embedded systems based on the analysis of existing methods; software implementation of the developed algorithm; conducting experiments of the developed tool.

Based on analysis of the existing tools of automated software testing of embedded systems and the fact that it is impossible to test the software of the embedded system on the hardware without modifying the code using existing software products, an automated testing algorithm is presented, for which no needs for additional changes to the source code of the embedded systems.

A developed software testing tool for the embedded system is validated with the delay function based on a hardware timer and a digital oscilloscope.

The research was based on the comparison analysis with the analogue device, which is based on static code analyzer. Benchmarking result has shown that the developed tool is more precise than its analogue.

The main purpose of master's qualification work is the research and development of the algorithm of automated software testing of embedded systems.

The object of the research is the process of automated test case generation and software testing of embedded systems.

Subject of research - models and algorithms of automated software testing of embedded systems

The total workload is 80 pages.

**ЗМІСТ**

[Анотація 4](#_Toc529380630)

[ABSTRACT 6](#_Toc529380631)

[ВСТУП 10](#_Toc529380632)

[РОЗДІЛ 1. АНАЛІЗ предметної області та суміжних досліджень. 12](#_Toc529380633)

[**1.1** **Великі дані, визначення та їх особливості** 12](#_Toc529380634)

[**1.2** **Поняття асоціативних правил** 12](#_Toc529380635)

[**1.3** **Аналіз існуючих досліджень** 12](#_Toc529380636)

[**1.4** **Мета, об’єкт та предмет дослідження** 12](#_Toc529380637)

[**1.5** **Висновок до розділу 1** 12](#_Toc529380638)

[РОЗДІЛ 2. Опис алгоритму apriori 12](#_Toc529380639)

[**2.1** **Алгоритм apriori** 12](#_Toc529380641)

[**2.2** **Видобування асоціативних правил** 12](#_Toc529380642)

[**2.3** **Опис відомих модифікацій** 12](#_Toc529380643)

[**2.4** **Опис можливих модифікацій** 12](#_Toc529380644)

[**2.5** **Висновок до розділу 2** 12](#_Toc529380645)

[Розділ 3. ПРОГРАМНА РЕАЛІЗАЦІя АЛГОРИТМУ apriori 13](#_Toc529380646)

[**3.1** **Загальна постановка задачі** 13](#_Toc529380648)

[**3.2** **Специфікація вимог** 13](#_Toc529380649)

[**3.3** **Використані технології** 13](#_Toc529380650)

[**3.4** **Висновок до розділу 3** 13](#_Toc529380651)

[3.2. Використані технології 13](#_Toc529380652)

[3.3. Реалізація інтерфейсу користувача програмного засобу для автоматизованого тестування програмного забезпечення вбудованих систем 14](#_Toc529380653)

[3.4. Розробка бази даних для зберігання результатів синтаксичного аналізу тексту програми вбудованої системи 17](#_Toc529380654)

[3.5. Розробка модуля взаємодії з програмним забезпеченням вбудованої системи, розробленої у Keil uVision 20](#_Toc529380655)

[3.6. Висновки до розділу 3 23](#_Toc529380656)

[Розділ 4. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ОТРИМАНИХ ДАНИХ 24](#_Toc529380657)

[4.1. Опис процесу дослідження 24](#_Toc529380658)

[4.2. Валідація розробленого засобу тестування програмного забезпечення вбудованих систем 25](#_Toc529380659)

[4.3. Проведення порівняльного аналізу з засобом-аналогом 30](#_Toc529380660)

[4.4. Висновки до розділу 4 33](#_Toc529380661)

[Розділ 5. ЕКОНОМІЧНА ХАРАКТЕРИСТИКА ПРОГРАМНОГО ПРОДУКТУ 35](#_Toc529380662)

[5.1. Економічна характеристика проектного рішення 35](#_Toc529380663)

[5.2. Розрахунок витрат на розробку та впровадження проектного рішення. 37](#_Toc529380664)

[5.3. Визначення комплексного показника якості 40](#_Toc529380665)

[5.4. Визначення експлуатаційних витрат 42](#_Toc529380666)

[5.5. Розрахунок ціни споживання проектного рішення 44](#_Toc529380667)

[6.6. Визначення показників економічної ефективності 46](#_Toc529380668)

[6.7. Висновки до розділу 5 47](#_Toc529380669)

[Висновки 49](#_Toc529380670)

[Перелік використаної літератури 51](#_Toc529380671)

[ДОДАТКИ 55](#_Toc529380672)

[Додаток А. План забезпечення якості програмного продукту 55](#_Toc529380673)

[Додаток Б. Інструкція користувача 66](#_Toc529380674)

[Додаток В. Сертифікат участі на конфереції «Computer Science and Engineering 2017» 68](#_Toc529380675)

[Додаток Д. Фрагменти коду 69](#_Toc529380676)

# ВСТУП

На сьогоднішній день тестування є одним з найбільш важливих засобів перевірки надійності програмного забезпечення. Тестування програмного забезпечення – це процес перевірки відповідності програмного засобу до поставлених вимог.

Здебільшого тестування програмного забезпечення проводиться ручним способом. Ручне тестування є трудомістким процесом, який потребує багато часу та фінансових ресурсів і в той самий час є менш ефективним, ніж автоматизоване тестування. Таким чином виникає потреба у автоматизації процесу тестування.

Тестування мікропрограмного забезпечення є складнішим, адже основна проблема полягає в специфіці вбудованих систем. На відміну від систем загального призначення, тести для мікропограмного забезпечення доводиться виконуватися в середовищі з специфічною апаратною підтримкою. Наприклад, у вбудованих систем мало пам'яті, і поставити засіб інтеграційного тестування всередину такого пристрою не представляється можливим.

У першому розділі даної роботи проведено аналіз методів тестування ПЗ вбудованих систем, що існують на сьогоднішній день. Проведений аналіз показав, що існуючі засоби не дозволяють однозначно та повністю виявити усі дефекти та встановити коректність функціонування програми що аналізується. Основним недоліком існуючих методів є неможливість проведення тестування без внесення змін у ПЗ вбудованої системи.

Тому у другому розділі цієї роботи подана архітектура засобу тестування програмного забезпечення вбудованих систем, новизною якого є можливість проведення процесу тестування без внесення змін у ПЗ вбудованої системи. В цьому розділі також описані алгоритми синтаксичного аналізу вихідних файлів компіляції проекту та алгоритм автоматизованого виконання тестів безпосередньо на вбудованій системі.

Третій розділ роботи містить специфікацію вимог до програмної реалізації алгоритмів представлених у розділі 2, інформацію про використані технології, графічний інтерфейс програмного засобу та певні особливості програмної реалізації засобу для проведення автоматизованого тестування ПЗ вбудованої систем.

У четвертому розділі показано дослідження розробленого засобу. Дослідження складається з двох частин: валідація засобу тестування, за допомогою «еталонної» функції, і цифрового осцилографа, та порівняльний аналіз з засобом-аналогом.

У останньому розділі даної роботи показано економічну характеристику програмного продукту, розробленого у розділі 3.

# РОЗДІЛ 1. АНАЛІЗ предметної області та суміжних досліджень.

* 1. **Великі дані, визначення та їх особливості**

Великі дані - це набори даних, настільки об'ємні та складні, що традиційне прикладне програмне забезпечення для обробки даних є недостатнім для їх вирішення. Великі задачі даних включають в себе захоплення даних, зберігання даних, аналіз даних, пошук, обмін, передачу, візуалізацію, запити, оновлення, конфіденційність інформації та джерела даних. Існує ряд понять, пов'язаних з великими даними: спочатку було 3 поняття обсягу, різноманітності, швидкості[1]. Іншими поняттями, які потім пояснюються великими даними, є достовірність (тобто, скільки шуму в даних) [2] та значення. Останнім часом термін "великі дані" має на увазі використання прогнозної аналітики, аналізу поведінки користувачів або деяких інших розширених методів аналізу даних, які витягують значення з даних, а рідко - до певного розміру набору даних. "Існує мало сумнівів у тому, що кількість даних, які зараз доступні, дійсно велика, але це не є найважливішою характеристикою цієї нової екосистеми даних". Аналіз наборів даних може знайти нові зв'язки з "точковими діловими тенденціями, запобігання захворюванням, боротьба з злочинністю та ін. "[3]

Вчені, керівники підприємств, практики медицини, реклами та уряди так само регулярно зустрічаються з труднощами з великими наборами даних у таких областях, як пошук в Інтернеті, технології, міська інформатика та бізнес-інформатика. Вчені стикаються з обмеженнями в роботі електронної науки, включаючи метеорологію, геноміку, сонеономію, складні фізичні симуляції, біологію та екологічні дослідження [4].

Цей термін використовується з 1990-х років, при цьому деякі приносять кредит Джону Маші за виготовлення або, принаймні, популярністю. [5] Великі дані зазвичай включають в себе набори даних, розміри яких не залежать від можливостей широко використовуваних програмних інструментів для захоплення, копіювання, керування та обробки даних протягом допустимого минулого часу [6]. Велика філософія даних охоплює неструктуровані, напівструктуровані та структуровані дані, однак основна увага приділяється неструктурованим даним.

Визначення 2016 р. Свідчить, що "великі дані являють собою інформаційні активи, які характеризуються такими великими обсягами, швидкістю та різноманітністю, що вимагають спеціальних технологій та аналітичних методів перетворення їх у вартість" [7].

Великі дані можна описати за такими характеристиками: [8]

**Обсяг** (англ. Volume) **–** кількість створених та збережених даних. Розмір даних визначає цінність та потенційне розуміння, і чи можна вважати великі дані чи ні.

**Різноманітність** (англ. Variety) **–** тип та характер даних. Це допомагає людям, які аналізують його, ефективно використовувати отримане розуміння. Великі дані витягуються з тексту, зображень, аудіо, відео; плюс він завершує відсутність частин через злиття даних.

**Швидкість** (англ. Velocity) **–** у цьому контексті швидкість, з якою дані створюються та обробляються, щоб задовольнити вимоги та виклики, які лежать на шляху зростання та розвитку. Великі дані часто доступні в режимі реального часу.

Дані треба обробляти за допомогою розширених інструментів (аналітики та алгоритмів) для виявлення значущої інформації. Наприклад, щоб керувати фабрикою, слід розглядати як видимі, так і невидимі проблеми з різними компонентами. Алгоритми формування інформації повинні виявляти та вирішувати невидимі проблеми, такі як деградація машини, знос компонентів тощо на заводі. [9]

* 1. **Поняття асоціативних правил**

Афінітивний аналіз (affinity analysis) — один з розповсюджених методів Data Mining. Його назва походить від англійського слова affinity, що у перекладі означає «близькість», «подібність». Ціль даного методу — дослідження взаємного зв'язку між подіями, які відбуваються спільно. Різновидом афінітивного аналізу є аналіз ринкового кошика (market basket analysis), ціль якого - виявити асоціації між різними подіями, тобто знайти правила для кількісного опису взаємного зв'язку між двома або більше подіями. Такі правила називаються асоціативними правилами (association rules).[10]

Базовим поняттям у теорії асоціативних правил є транзакція — деяка множина подій, що відбуваються спільно. Типова транзакція - покупка клієнтом товару в супермаркеті. У переважній більшості випадків клієнт купує не один товар, а набір товарів, що називається ринковим кошиком. При цьому виникає питання: чи є покупка одного товару в кошику наслідком або причиною покупки іншого товару, тобто, чи пов'язані дані події? Цей зв'язок і встановлюють асоціативні правила. Наприклад, може бути виявлене асоціативне правило, котре стверджує, що клієнт, який купив молоко, з імовірністю 75 % купить і хліб.

Наступне важливе поняття — предметний набір. Це непорожня множина предметів, що з'явилися в одній транзакції.

Аналіз ринкового кошика - це аналіз наборів даних для певної комбінації товарів, пов'язаних між собою. Іншими словами, виконується пошук товарів, присутність яких у транзакції впливає на ймовірність наявності інших товарів або комбінацій товарів.

Сучасні касові апарати в супермаркетах дозволяють збирати інформацію про покупки, що може зберігатися в базі даних. Потім накопичені дані можуть використовуватися для побудови систем пошуку асоціативних правил.

У табл. 1.1 представлений простий приклад, що містить дані про ринковий кошик. У кожному рядку вказується комбінація продуктів, придбаних за одну покупку. Хоча на практиці доводиться мати справу з мільйонами транзакцій, у яких беруть участь десятки й сотні різних продуктів, приклад обмежений 10 транзакціями, що містять 13 видів продуктів: аби проілюструвати методику виявлення асоціативних правил, цього досить.

Таблиця 1.1.

Приклад набору транзакцій

|  |  |
| --- | --- |
| **№** | **Транзакція** |
| 1 | Сливи, салат, помідори |
| 2 | Селера, цукерки |
| 3 | Цукерки |
| 4 | Яблука, морква, помідори, картопля, цукерки |
| 5 | Яблука, апельсини, салат, цукерки, помідори |
| 6 | Персики, апельсини, селера, помідори |
| 7 | Квасоля, салат, помідори |
| 8 | Апельсини, салат, морква, помідори, цукерки |
| 9 | Яблука, банани, сливи, морква, помідори, цибуля, цукерки |
| 10 | Яблука, картопля |

Візуальний аналіз прикладу показує, що всі чотири транзакції, у яких фігурує салат, також включають помідори, і що чотири із семи покупок, що містять помідори, також містять салат. Салат і помідори в більшості випадків купуються разом. Асоціативні правила дозволяють виявляти й кількісно описувати такі збіги.

Асоціативне правило складається із двох наборів предметів, що мають назву умова (antecedent) та наслідок (consequent), й записуються у вигляді . Таким чином, асоціативне правило формулюється у вигляді: «Якщо умова, то наслідок».

Умова може обмежуватися тільки одним предметом. Правила звичайно відображаються за допомогою стрілок, спрямованих від умови до наслідку, наприклад, помідори → салат.

Візуальний аналіз прикладу показує, що всі чотири транзакції, у яких фігурує цибулю, також включають картоплю, і що чотири із семи шести, що містять картоплю, також містять цибулю. Цибулю і картоплю в більшості випадків купуються разом. Асоціативні правила дозволяють виявляти й кількісно описувати такі збіги. Також правила можуть бути комбіновані {Цибуля, Картопля} => {Бургер}

Асоціативне правило складається із двох наборів предметів, що мають назву умова (antecedent) та наслідок (consequent), й записуються у вигляді . Таким чином, асоціативне правило формулюється у вигляді: «Якщо умова, то наслідок».

Умова може обмежуватися тільки одним предметом. Правила звичайно відображаються за допомогою стрілок, спрямованих від умови до наслідку, наприклад, цибуля → картопля.

Основними характеристиками, що описують асоціативне правило, є підтримка (support) і вірогідність (confidence).

Якщо позначити базу даних транзакцій через , а число транзакцій у цій базі , то кожна транзакція  являє собою певний набір предметів. Позначимо підтримку правила через , а вірогідність – через .

Підтримка асоціативного правила — це число транзакцій, які містять як умову, так і наслідок. Наприклад, для асоціації  можна записати



Вірогідність асоціативного правила  являє собою міру точності правила й визначається як відношення кількості транзакцій, що містять умову і наслідок, до кількості транзакцій, що містять тільки умову:



Якщо підтримка й вірогідність досить високі, можна з великою ймовірністю стверджувати, що будь-яка майбутня транзакції, що включає умову, буде містити й наслідок.

Для прикладу розглянемо правило салат → помідори, що напрошується з попередніх спостережень . Для нього

;

.

Дане правило, що зустрічається в 40% транзакцій, є для вихідних даних абсолютно вірним – у всіх випадках, коли клієнт купує салат, він разом з тим купує й помідори. Це легко пояснити логічно - обидва продукти використовуються для готування овочевих блюд і дійсно часто купуються разом.

Тепер розглянемо асоціацію , у якій містяться слабко сумісні в гастрономічному плані продукти.

*помідори*

*цукерки*



Підтримка даної асоціації S = 4/10 = 0,4 – та ж, що в попереднього правила, а вірогідність С = 4/6 = 0,67. Таким чином, порівняно невисока вірогідність даної асоціації дає привід засумніватися в тому, що вона є правилом.

Число 0,67 здається не таким вже й малим. Чому ж ми говоримо про «незначну вірогідність» цієї асоціації? Справа в тому, що помідори зустрічаються в 7 чеках з 10 ( ). Прийнято, що всі правила з вірогідністю меншою, ніж проста ймовірність випадання наслідку не повинні розглядатися, адже вони, по суті, випадкові. Для прийняття асоціативного правила його вірогідність повинна бути не меншою ніж імовірність наслідку.

Останнє зауваження при досить великій номенклатурі товарів призводить до необхідності подвоювати кількість розрахунків. На практиці аналітики можуть віддавати перевагу правилам, які мають високу підтримку (вище певного рівня, наприклад, 0,3) або високу вірогідність (не менше 0,8-0,85). Висування одночасних вимог щодо підтримки й вірогідності дозволяють значно позм'якшити критерії (підтримку до 0,1-0,15, вірогідність – до 0,67-0,75). Правила, для яких значення підтримки й вірогідності перевищують певні, задані користувачем пороги, називають сильними правилами (strong rules). Всі наведені вище числові значення - емпіричні. Наприклад, у задачах виявлення шахрайських операцій значення підтримки може знижуватися й до 1%, оскільки із шахрайством пов'язане порівняно невелике число транзакцій.

Крім об'єктивних оцінок (підтримки й вірогідності) кожного зі згенерованих правил, різні джерела радять використовувати деякі суб'єктивні оцінки. Всі вони, так чи інакше, базуються на об'єктивних.

Ліфт (від interest lift – підвищення інтересу) обчислюється в такий спосіб



Ліфт - це відношення частоти появи умови в транзакціях, які містять й умову, і наслідок, до частоти появи наслідку в цілому. Чим більше значення ліфта, тим частіше наслідок визначається умовою в порівнянні з випадками, коли умова відсутня. Якщо ліфт дорівнює 1, зв'язок відсутній, близькі ж до нуля значення свідчать про сильну зворотну залежність.

Для нашого прикладу в таблиці 1.1. візьмемо два правила з однаковою вірогідністю:

 4/10 = 0,4. ,4/7 = 0,57.

*салат) =*

*C(помідори*



 6/10 = 0,6. 4/7 = 0,57.

*цукерки) =*

*C(помідори*



Здавалося б, правила однаково достовірні. Після розрахунку ліфта все стає на свої місця:

0,57/0,4 = 1,425;

*салат) =*

*L(помідори*



0,57/0,6 = 0,95.

*цукерки) =*

*L(помідори*



Не варто вважати ліфт універсальним мірилом адекватності. Справа в тому, що правило з меншою підтримкою й більшим ліфтом може бути менш значимим, ніж альтернативне правило з більшою підтримкою й меншим ліфтом. Це пов'язане з тим, що останнє застосовується для більшого числа покупців.

Левередж (leverage – важіль, плече) – це різниця між спостережуваною частотою, з якої умова й наслідок з'являються спільно, та добутком частот появи умови й наслідку окремо



Левередж дозволяє впоратися із ситуаціями, коли й підтримка, й ліфт у правил ідентичні, але їх значимість явно відрізняється. Наприклад, у нашому овочевому магазині в правила 

;  0,7;  1/0,7 = 1,43.

Морква, як показує таблиця 1.1, також продається тільки з помідорами, і також зустрічається чотири рази, тому й у правила 

;  0,7;  1/0,7 = 1,43.

А от левередж у цих правил відрізняється на 30%:

 0,3 – 0,3 \* 0,7 = 0,09.

 0,4 – 0,4 \* 0,7 = 0,12.

Таким чином, значимість другої асоціації більша, ніж першої.

Альтернативою левередж є поліпшення.

Поліпшення (improvement) – це відношення частоти спостережуваних виконань правила до добутку частот появи умови й наслідку окремо.



фактично, поліпшення показує, у скільки разів розглянуте правило забезпечує правильний прогноз краще, ніж випадкове вгадування. Всі правила  не є значимими.

Такі міри, як ліфт, левередж і поліпшення, можуть використовуватися для обмеження набору розглянутих асоціацій шляхом встановлення граничних значень значимості, нижче яких асоціації відкидаються.

* 1. **Аналіз існуючих досліджень**

На сьогоднішній день існує величезна кількість досліджень та модифікацій для алгоритму Apriori.

У [11] проаналізовано алгоритм Apriori як метод пошуку асоціативних правил у структурованих та неструктурованих даних з погляду кількості знайдених правил, швидкодії та потреб в обчислювальних ресурсах. Неструктуровані дані тісно пов’язані з терміном Big Data. Актуальним завданням інженерії даних є виявлення ефективних засобів опрацювання неструктурованої інформації. Для проведення обчислювальних експериментів розроблено програмну систему, що опрацьовує дані алгоритмом Apriori, предметною областю якої вибрано торгівлю. Така система може бути прототипом реальної рекомендаційної системи.

Отже було досліджено що,

1. Втрати у швидкодії для неструктурованої інформації порівняно зі структурованою інформацією незначні зі зростанням обсягів інформації, і зокрема, для випадку обсягу даних 2,5 Гб зростання часу роботи становить близько 12 %. А на порівняно невеликих обсягах даних для структурованих даних алгоритм працює утричі швидше порівняно із неструктурованими.

2. Кількість знайдених правил для неструктурованої інформації на близько 25 % менша, ніж для структурованої інформації.

3. Для коефіцієнтів підтримки та достовірності, вищих від середнього, для неструктурованих даних не знайдено жодного правила, а для структурованих результат – від 2 до 5 правил.

У [12] виконувався аналіз надзвичайних ситуацій у залізничному транспорті для пошуку асоціативних правил із використанням алгоритму Apriori. У цьому дослідженні запропоновано модифікацію алгоритму: розбитя множини всіх можливих характеристик на дві підмножини:

1. A – характеристики, сукупність яких могла стати причиною виникнення надзвичайних ситуацій.
2. B – характеристики, скупність яких описує надзвичайну ситуацію що виникла.

В результаті застосування цієї модифікації збільшується швидкість роботи

алгоритму.

З отриманих результатів дослідження видно що алгоритм є доволі дослідженим та схильном до модифікацій проте, його можна досліджувати для навіструктурованих даних або ввести номі модифікації для більш точної обробки інформації.

* 1. **Мета, об’єкт та предмет дослідження**

Мета роботи полягає в порівняльному аналізі роботи алгоритму Apriori на великих обсягах даних. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Об’єкт дослідження – алгоритм apriori та його модифікації.

Предмет дослідження – час і точність пошуку асоціативних правил та пошук можливих модифікацій.

* 1. **Висновок до розділу 1**

Накопичення інформації є актуальною темою сьогодення. Для аналізу цієї інформації використовують різноманітні технології та алгоритму. Після огляду декількох із них було вирішено провести дослідження для алгоритму Apriori.

Отже, у магістерській кваліфікаційній роботі необхідно розглянути алгоритм Apriori з точки зору пошуку асоціативних правил у великій кількості даних та проаналізувати затрачені обчислювальні ресурси. Мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій на великих обсягах даних. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних.

Після аналізу існуючих систем, що займаються пошуком закономірностей у даних, огляду існуючих підходів та алгоритмів для опрацювання великих даних було виявлено потребу у дослідженні роботи алгоритму для напіструктурованих даних, так як досліджень аналогів не було виявлено та виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

# РОЗДІЛ 2. Опис алгоритму apriori

1. 1. **Алгоритм apriori**

На даний момент існує декілька основних алгоритмів пошуку асоціативних правил в загальному вигляді, що не зав’язані на конкретизовану предметну область. Розглянемо найбільш поширені з них.

Алгоритм Apriori [14] призначений для знаходження всіх частих наборів елементів множини транзакцій, що аналізується. Він працює порівнево, використовуючи стратегію пошуку в ширину і виконується зверху вниз. Алгоритм Apriori має просту структуру, завдяки якій в неї можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних для вирішення задачі пошуку асоціативних правил.

Алгоритм Eclat [15] побудований на основі пошуку в глибину використовуючи перетин наборів елементів для знаходження частих наборів елементів. В загальному випадку алгоритм працює повільніше ніж алгоритм Apriori, однак в ситуації коли кількість різних елементів транзакцій є невеликою, він працює швидше.

Алгоритм FP-growth [16] (frequent pattern growth, зростання частих патернів) використовує розширене префіксне дерево для збереження бази даних у стислому вигляді. Він застосовує метод «розділяй і володарюй» для декомпозиції і видобутку знань із бази даних. В процесі роботи алгоритм використовує метод зростаючих патернів для уникнення ресурсоємкого процесу генерації частих кандидатів і їх тестування, що використовується в алгоритмі Apriori. Однак через це ускладнюється процес оптимізації алгоритму для використання з специфічними наборами даних, де покращення швидкості роботи можна досягти іншими методами.

Отже, враховуючи переваги та недоліки розглянутих алгоритмів пошуку асоціативних правил, було обрано алгоритм Apriori так як він є простим у розумінні та можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних.[23-29]

Таблиця 2.1.

Приклад набору транзакцій

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Transaction ID** | **Цибуля** | **Картопля** | **Бургер** | **Молоко** | **Пиво** |
| t_1 | 1 | 1 | 1 | 0 | 0 |
| t_2 | 0 | 1 | 1 | 1 | 0 |
| t_3 | 0 | 0 | 0 | 1 | 1 |
| t_4 | 1 | 1 | 0 | 1 | 0 |
| t_5 | 1 | 1 | 1 | 0 | 1 |
| t_6 | 1 | 1 | 1 | 1 | 1 |

Ключовим поняттям в алгоритмі Апріорі є антимонотонність міри підтримки. Це передбачає що:

* Всі підмножини частої Набір повинні бути частими
* Подібним чином, для будь-якого незвичного набору, всі його суперсеті також повинні бути нечастими

Давайте розглянемо інтуїтивно зрозуміле пояснення алгоритму за допомогою наведеного вище прикладу(табл. 2). Перед початком процесу, встановіть порогової підтримки до 50%, тобто лише ті елементи значні, для яких підтримка більше 50%.

**Крок 1**: Створити таблицю частот для всіх елементів, що виникають у всіх транзакціях. Для нашого випадку:

Таблиця 2.2.

Таблиця частот

|  |  |
| --- | --- |
| **Одиниця** | **Частота** |
| Цибуля(O) | 4 |
| Картопля(P) | 5 |
| Бургер(B) | 4 |
| Молоко(M) | 4 |
| Пиво(Be) | 2 |

**Крок 2**: Ми знаємо, що важливі лише ті елементи, для яких підтримка більше або дорівнює пороговій підтримці. Тут границя підтримки становить 50%, отже, лише ті елементи значущі, що трапляються у більш ніж трьох транзакціях, а такими є - Цибуля (O), Картопля (P), Бургер (B) і Молоко (M). Тому ми отримуємо наступну таблицю:

Таблиця 2.3.

Таблиця частот

|  |  |
| --- | --- |
| **Одиниця** | **Частота** |
| Цибуля(O) | 4 |
| Картопля(P) | 5 |
| Бургер(B) | 4 |
| Молоко(M) | 4 |

**Крок 3**: Наступним кроком є зробити всі можливі пари значущих елементів, маючи на увазі, що порядок не має значення, тобто АB такий самий, як BА. Щоб зробити це, візьміть перший предмет і з'єднайте його з усіма іншими, такими як OP, OB, OM. Аналогічним чином розглянемо другий елемент і об'єднає його з попередніми пунктами, тобто PB, PM. Ми розглядаємо лише попередні пункти, оскільки PO (як OP) вже існує. Отже, всі пари в нашому прикладі - OP, OB, OM, PB, PM, BM.

**Крок 4:** Тепер ми розглянемо випадки кожної пари у всіх транзакціях.

Таблиця 2.4.

Таблиця частот

|  |  |
| --- | --- |
| **Набір** | **Частота** |
| OP | 4 |
| OB | 3 |
| OM | 2 |
| PB | 4 |
| PM | 3 |
| BM | 2 |

**Крок 5.** Знову тільки ті набори наборів є значними, які перетинають порогову підтримку, а такі - OP, OB, PB та PM.

**Крок 6.** Тепер скажімо, ми хотіли б шукати набір з трьох елементів, які купуються разом. Ми будемо використовувати пункти, знайдені на кроці 5, і створимо набір з 3 елементів.

Щоб створити набір з 3 елементів, потрібно інше правило, яке називається самоз'єднання. Він говорить, що з пар OP, OB, PB та PM ми шукаємо дві пари з однаковою першою буквою, і об’єднуємо їх:

* OP та OB, це дає OPB
* PB і PM, це дає PBM

Далі ми знайдемо частоту для цих двох наборів.

Таблиця 2.5.

Таблиця частот

|  |  |
| --- | --- |
| **Набір** | **Частота** |
| OPB | 4 |
| PBM | 3 |

Знову застосувавши правило порогу, ми виявили, що OPB є єдиним значним набором елементів.

Отже, набір з 3 предметів, який був придбаний найчастіше, - OPB.

Приклад, який ми розглянули, був досить простим, і видобуток частих предметів у товаристві зупинено на 3 пунктах, але на практиці існує кілька десятків предметів, і цей процес може тривати багато предметів. Припустимо, що ми отримали значні набори з трьома елементами як OPQ, OPR, OQR, OQS та PQR, і тепер ми хочемо створити набір з 4 елементів. Для цього ми розглянемо набори, які мають перші два алфавіти, тобто загалом

* OPQ та OPR дають OPQR
* OQR та OQS дають OQRS

Взагалі, ми повинні шукати набори, які тільки відрізняються в їхній останній букві / предметі.

Тепер, коли ми розглянули приклад функціональності алгоритму Апріорі, давайте сформулювати загальний процес.

Весь алгоритм можна розділити на два етапи:

Крок 1: застосуйте мінімальну підтримку, щоб знайти всі часто використовувані набори з елементами k у базі даних.

Крок 2. Використовуйте правило самопідключення, щоб знайти часті множини з елементами k + 1 за допомогою часто використовуваних k-елементів. Повторіть цей процес з k = 1 до моменту, коли нам не вдається застосувати правило самопідключення.

Такий підхід до розповсюдження частих елементів, що належить до одного, називається підходом "знизу вверх".

Cкладність цього алгоритму залежить від мінімального значення підтримки. Припустимо, що кількість вхідних транзакцій N, а кількість унікальних елементів R. Складність для генерації набору розміру i – О(Ri), де і – час для обчислення підтримки для кожного набору може бути O(N), у разі використання HashMap.

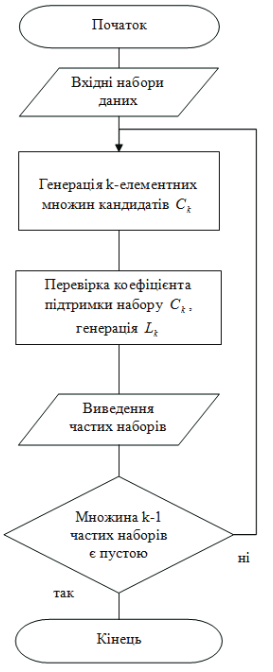


Рис. 2.1. Приклад роботи алгоритму у вигляді блок схеми

* 1. **Видобування асоціативних правил**

До цих пір ми розглянули алгоритм Апріорі по відношенню до частого покоління пакетів. Існує ще одне завдання, за допомогою якого ми можемо використовувати цей алгоритм, тобто ефективно знаходити правила об'єднання. [26-27]

Щоб знайти правила об'єднання, нам потрібно знайти всі правила, що мають підтримку, більшу за порогову підтримку та довіру, більшу за порогову довіру.

Але як ми знаходимо це? Одним із можливих шляхів є груба сила, тобто перелік всіх можливих правил асоціації та обчислення підтримки та впевненості для кожного правила. Потім скасуйте правила, які не підтримують порогову підтримку та довіру. Але це обчислювально дуже важкий і заборонений, оскільки кількість всіх можливих правил асоціації зростає експоненціально з кількістю предметів.

Враховуючи, що в наборі I є n пунктів, загальна кількість можливих правил асоціації становить 3 ^ n - 2 ^ (n +1) + 1.

Ми також можемо використовувати інший спосіб, який називається двоетапним підходом, для пошуку ефективних правил асоціації.

Двоступеневий підхід:

Крок 1: генерація частотних наборів елементів: знайдіть всі набори предметів, для яких підтримка перевищує порогову підтримку відповідно до процесу, який ми вже бачили раніше в цій статті.

Крок 2: генерація правил: створюйте правила з кожної частинки, використовуючи бінарний розділ часто використовуваних предметів і шукайте тих, що мають високу впевненість. Ці правила називаються правилами кандидатів.

Давайте розглянемо наш попередній приклад, щоб отримати ефективне правило об'єднання. Ми виявили, що OPB є частим комплектом. Тому для цієї проблеми, крок 1 вже зроблено. Отже, давайте розглянемо крок 2. Усі можливі правила, що використовують OPB, є:

OP\longrightarrowB, OB\longrightarrowP, PB\longrightarrowO, B\longrightarrow OP, P\longrightarrowOB, O\longrightarrowPB

Якщо X являє собою частий набір елементів з k-елементами, то існують правила асоціації кандидатів 2 ^ k-2.

Плюси алгоритму:

Легкий у реалізації та розумінні.

Може бути використаний для величезних даних.

Мінуси алгоритму:

Іноді, потрібно знайти величезну кільуість правил кандидатів, що є обчислювано затратно.

Підрахуном підтримки також затратний по ресурсах, адже потрібно пройтись по всій базі.

* 1. **Опис відомих модифікацій**

Проаналізувавши суміжні дослідження було виявлено що вже існує ряд модифікацій для алгоритму apriori, які покращують його роботу.

PredictiveApriori - клас реалізації інтелектуального алгоритму Apriori для видобування асоціативних правил. Він виконує пошук зі збільшенням порогу підтримки для кращих 'N' правил, що стосуються скоригованого значення достовірності на основі підтримки [13].

Правило додається, якщо: очікувана прогностична точність цього правила є одним з 'N' найкращих і не поглинена правилом, принаймні тієї ж очікуваної точності прогнозу. Якщо включені асоціативні правила видобуваються замість (загальних) асоціативних правил.

Якщо встановлено значення -1, останній атрибут буде взятий в якості атрибута класу та існує кількість правил для знаходження. Можливі пропущені значення, бінарні атрибути, порожні номінальні атрибути та номінальний клас.

Алгоритм AprioriTid є різновидом алгоритму Apriori. Відмінною рисою даного алгоритму є підрахунок значення підтримки кандидатів не при скануванні безлічі D, а за допомогою безлічі Ck, які є множиною кандидатів (k- елементних наборів) потенційно частих, у відповідність яким ставиться ідентифікатор TID транзакцій, в яких вони містяться.

Кожен член безлічі Ck є парою виду , де кожен Fk є потенційно частим k - елементним набором, представленим в транзакції з ідентифікатором TID. Безліч C1 = D відповідає безлічі транзакцій, хоча кожен об'єкт в транзакції відповідає однооб'єктному набору в безлічі C1, що містить цей об'єкт. Для k> 1 безліч Ck генерується відповідно до алгоритму, описаним нижче.

Член безлічі Ck, відповідний транзакції Т, є парою такого вигляду:



Підмножина наборів в Ck з однаковими TID (містяться в одній і тій же транзакції) називається записом. Якщо транзакція не містить ні одного k - елементного кандидата, то Ck не матиме записи для цієї транзакції Тобто кількість записів в Ck може бути менше, ніж в D, особливо для великих значень k. Крім того, для великих значень k кожна запис може бути менше, ніж відповідна їй транзакція, т. К. В транзакції буде міститися мало кандидатів. Однак для малих знань k кожен запис може бути більше, ніж відповідна транзакція, тобто Ck включає всіх кандидатів k-елементних наборів, що містяться в транзакції.

Апріорі-TID використовує генерацію кандидата і апріорну функцію для визначення кандидата для наборів перед початком проходу. Основна відмінність від Apriori в тому, що він не використовує базу даних, для підрахунку підтримки після першого проходу. Швидше, він використовує 41 кодування кандидата наборів елементів, який використовується в попередному прохододі і позначається Ck. У Apriori-TID, кандидат в Ck наборів зберігаються в масиві індексується TID в Ck наборів. Кожен Ck зберігається в послідовної структури. У проході k-го, Апріорі-TID потребує обсяг пам'яті для Lk-1 і Ck при генерації кандидатів. Було також встановлено, що Апріорі-TID Перевищує Апріорі, коли є менше число наборів Ck, яке може поміститися в пам'яті і розподіл з великих наборів має довгий набір . Це означає, що розподіл записів у великих наборів високий на ранній стадії [13].

Іншим різновидом алгоритму Apriori є алгоритм MSAP (Mining Sequential Alarm Patterns), спеціально розроблений для виконання секвенціального аналізу збоїв телекомунікаційної мережі. Він використовує наступне властивість підтримки послідовностей: для будь-якій послідовності Lk її підтримка буде менше, ніж підтримка послідовностей з безлічі Lk1. Алгоритм MSAP для пошуку подій, що слідують один за одним, використовує поняття "термінового вікна" (Urgent Window). Це дозволяє виявляти не просто однакові послідовності подій, а наступні один за одним. В іншому даний алгоритм працює за тим же принципом, що і Apriori.

* 1. **Опис можливих модифікацій**

Отже, переглянувши декілька відомих модифікацій, проаналізувавши предметну область та зрозумівши принцип роботи алгоритму та пошуку асоціативних правил було запропоновано модифікації для алгоритму.

Відомо, що алгоритм на кожному кроці визначає:

* підтримку (англ. support) - скільки разів у всьому масиві використано елементи даних, що складаються з X та Y
* достовірність (англ. confidence) - який відсоток від всіх одиниць, що містять X, містить також і Y

Асоціативні правила мають також характеристику під назвою Ліфт (англ. lift) – це відношення частоти появи умови в транзакціях, які містять й умову, і наслідок, до частоти появи наслідку в цілому. Чим більше значення ліфта, тим частіше наслідок визначається умовою в порівнянні з випадками, коли умова відсутня. Якщо ліфт дорівнює 1, зв'язок відсутній, близькі ж до нуля значення свідчать про сильну зворотну залежність.

Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсіканання абсурдних або випадкових груп правил, оттже швидкість алгоритму повинна зрости адже не потрібно здійснюватинаступні кроки по вже відкинутих правилах.

Цей підхід можна спробувати для вище перелічених методів, а також спробувати даний алгоритм при різних значеннях коефіцієнтів підтримки та достовірності.

Ще одним із можливих варіантів модифікації є розділ множини факторів I на дві підмножини:

1) фактори, які мали місце до виникнення асоціації Si;

2) фактори, що стали результатом виникнення асоціації Di.

Асоціативні правила, що містять лише підмножину факторів із множини Si , не містять\* корисної інформації для виконання подальшого аналізу, завдяки чому можна прискорити алгоритм пошуку асоціативних правил. Для цього необхідно змінити процес генерації наборів кандидатів Ck , в якому відсікати набори, що не містять факторів із множини Di. В результаті значно зменшиться кількість наборів кандидатів Ck , що генеруватиметься на кожній ітерації алгоритму, а також це впливатиме на генерацію частих наборів Lk. Час обробки множини наборів Ck та генерації Lk лінійно залежить від потужності даних множин і загальної потужності множини І, яка є сталою в процесі роботи алгоритму. Тому завдяки зменшенню кількості елементів множини Ck зменшиться загальний час роботи алгоритму.

* 1. **Висновок до розділу 2**

Здійснено обгрунтування та порівняльний аналіз алгоритму, визначено що алгоритм є простий у розумінні та схильний до модифікацій.

Запропоновано покращення базового алгоритму, за допомогою застосування характиристики Ліфту на кожному кроці алгоритму. Даний підхід зменшить витрати часу при знаходженні асоціативних правил. Також є можливим впровадження даного підходу що уже відомих модифікацій, а також спробувати даний алгоритм при різних значеннях коефіцієнтів підтримки та достовірності.

Також запропоновано ще один варіант у якому вхідна множина розділяється на дві підмножини, перша множина це множина із факторами які мали місце до виникнення асоціацій і друга – множина із факторами що стали результатом виникнення асоціацій, на кожному кроці відсікається перша множина, адже вона не містить корисної інформаці.

# Розділ 3. ПРОГРАМНА РЕАЛІЗАЦІя АЛГОРИТМУ apriori

1. 1. **Загальна постановка задачі**

Реалізувати програмний засіб, за допомогою якого можна здійснювати інтелектуальний аналіз даних, а саме пошук асоціативних правил. Даний програмний засіб повинен надавати можливість для проведення аналізу вхідних даних та виведення результатів аналізу (знайдені асоціативні правила, їх кількість та швидкодію). Після виконання програма повинна формувати звітність на основі результатів аналізу представлених даних. Результати будуть представлені безпосередньо на екрані із можливістю збереження їх у файл.

Провести дослідження ефективності алгоритму. На основі отриманих даних із експериментів визначити можливість видозміни алгоритму для збільшення ефективності його роботи. При позитивних результатах виконати модифікацію алгоритму.

* 1. **Специфікація вимог**

**3.2.1. Вступ**

Призначення даного продукту – автоматичний аналіз великих наборів даних та пошук в них асоціативних правил, формування звіту.

Мета – реалізувати класичний алгоритм apriori та модифіковані його версії та порівняти результати роботи у вигляді програмного продуку.

**3.2.2. Загальний опис**

**3.2.2.1. Характеристики продукту**

Функції, що будуть реалізовані у програмі:

* Зчитування даних з файлу;
* Аналіз даних;
* Виведення результатів у вигляді таблиць та діаграм;
* Формування звіту про результати аналізу;

**3.2.2.2. Класи користувачів та їх характеристики**

В даній програмі передбачено лише один клас користувачів, тому функціонал не буде мати обмеження.

Програмний засіб призначений для бізнес аналітиків, щоб ефективно провести інтелектуальний аналіз даних та сформувати звіт замовнику.

**3.2.2.3. Середовище функціонування**

Апаратні вимоги:

* Частота процесора 1 GHz;
* Оперативна пам’ять 512 Mb;
* 10 Mb вільного простору на диску.

Системні вимоги:

* Операційна система Microsoft Windows XP або вище;
* .Net Framework версії 4.0 і вище;

**3.2.3. Характеристики системи**

**3.2.3.1. Пошук асоціативних правил для введених даних**

3.2.3.1.1. Опис і пріоритет

Аналіз введених даних, пошук залежностей. Виведення результатів тестування у зручному для користувача вигляді, у вигляді таблиць та діаграм, із можливістю збереження на локальний диск.

Пріоритет – високий.

3.2.3.1.2. Послідовності дія/відгук

Користувач вибирає текстовий документ із даними, аналіз яких потрібно провести. Натискає на кнопку «Почати аналіз» і у низу вікна статус бар відображатиме прогрес.

3.2.3.1.3. Функціональні вимоги

REQ-1.1: Зчитування даних із .txt формата;

REQ-1.2: Проведення аналізу;

REQ-1.3: Пошук асоціативних правил;

REQ-1.4: Виведення повідомлення про закінчення проведення аналізу;

REQ-1.5: Виведення результатів;

**3.2.3.2. Формування звіту результату**

3.2.3.2.1. Опис і пріоритет

Створення звіту, який буде відображати інформацію про результат пошуку залежностей. Звіт можна зберегти на диск у форматі txt.

3.2.3.2.2. Послідовності дія/відгук

Для збереження звіту потрібно натиснути кнопку «Зберегти звіт», після чого появиться діалогове вікно для вибору місця збереження файлу.

3.2.3.2.3. Функціональні вимоги

REQ-2.1: Формування звіту;

REQ-2.2: Збереження на диску у форматі txt;

**3.2.4. Вимоги зовнішніх інтерфейсів**

**3.2.4.1. Користувацькі інтерфейси**

Інтерфейс програми повинен бути простим для розуміння, забезпечувати швидкодію відповідно до вимог та дозволяти зручно виконувати необхідні операції.

**3.2.4.2. Програмні інтерфейси**

* Бібліотека Microsoft.Office.Interop.Excel;
* Бібліотека Xml.Serializer;

**3.2.5. Інші нефункційні вимоги**

**3.2.5.1. Вимоги продуктивності**

Швидкість проведення аналізу буде прямо залежати від кількості введених даних, кількості галужень та від потужності процесора.

**3.2.5.2. Вимоги безпеки**

Немає вимог безпеки.

**3.2.5.3. Атрибути якості програмного продукту**

Програма реалізована із мінімальним основним набором функції, саме тому вона зрозуміла та зручна для використання користувачам, які з нею ніколи не працювали;

Ніяким чином не впливає на роботу інших програм.

**3.2.5.4. Інші вимоги**

Необхідна наявність встановленого пакету Microsoft Excel та .Net Framework версії 4.0 і вище.

* 1. **Використані технології**

Для написання продукту буде використовуватись архітектура клієнт-сервер.

Оскільки основним завданням проекту буде аналіз роботи алгоритму то було обрано основну архітектуру як клієнт-сервер, із основною логікою на сервері, клієнт лише приймає дані та показує результати.

Для створення програмного продукту на серверній частині було обрано технологію .Net, середовище розробки Microsoft Visual Studio[19], мова C#[18] та фреймворк ASP.NET Web API[17]. Також для обробки великих даних було обрано технологію Hadoop[28].

На клієнтській було обрано такі технології як: HTML5\CSS3, JavaScript, Bootstrap, фреймворк AngularJS та середовище розробки Web Storm[20-21]

 .NET Framework є основою програмного забезпечення, розроблена корпорацією Майкрософт, яка працює в основному на Microsoft Windows. Вона містить велику бібліотеку класів під назвою Framework Class Library (FCL) і забезпечує мовну сумісність (кожна мова може використовувати код, написаний на інших мовах) на кілька мов програмування. Програми, написані для .NET Framework виконуються в програмному середовищі (на відміну від апаратної середовища) під назвою Common Language Runtime (CLR), віртуальна машина додаток, яке надає такі послуги, як безпека, управління пам'яттю і обробки винятків.

C # — є об’єктно-орієнтованою мовою програмування яка охоплює строгу типізацію, імперативність, декларативність, функціональні, загальні, об'єктно-орієнтовані (на основі класів), компонент-орієнтоване програмування дисциплін. C # є одним з мов програмування, призначених для Common Language Infrastructure.

JavaScript (JS) — це високорівнева, не типізована та динамічна мова програмування. Він був стандартизований в специфікації мови ECMAScript. Поряд з HTML і CSS, JavaScript є одним з трьох основних технологій світового виробництва Wide Web контенту; більшість сайтів використовують його, і всі сучасні веб-браузери підтримують його без необхідності плагінів. JavaScript є прототипом основи з функціями першого класу, що робить його мову мульти-парадигму, підтримка об'єктно-орієнтовані, імперативні і функціональні стилі програмування.

Apache Hadoop — вільна програмна платформа і каркас для організації розподіленого зберігання і обробки наборів великих даних з використанням моделі програмування MapReduce, при якій завдання ділиться на багато дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі кластера, що складається з серійних комп'ютерів. Всі модулі в Hadoop спроектовані з врахуванням припущення, що злам апаратного забезпечення трапляється часто і це повинно автоматично враховуватись фреймворком.

Ядро системи Apache Hadoop складається з розподіленої файлової системи Hadoop Distributed Filesystem (HDFS), та системи обчислень на основі моделі програмування MapReduce. Hadoop розділяє файли на великі блоки і розподіляє їх між вузлами кластера.

* 1. **Висновок до розділу 3**

Ключовим завданням постає розробка програмного засобу для проведення аналізу даних алгоритмом apriori. По суті програма буде працювати у певній логічній послідовності, отже варіантів дій користувача не так і багато, що робить користування просте та зрозуміле. Користувачу представляється лише вибір даних, який слід аналізувати і результати для подальшого використання.

Так як ціль проекту включає в собі інтелектуальний аналіз даних то технологія Microsoft .NET чудово підходить для написання прикладної програми. Для побудови інтерфейсу використовуватиметься технологія AngularJs, яка дозволить без проблем створити цікавий вигляд програми, проект буде розроблено за клієнт-серверною архітектурою. Також буде використовуватись Hadoop для фільтрування та групування даних, які потрібні для роботи алгоритму.

## **Використані технології**

Оскільки для вирішення поставленої задачі потрібна можливість отримати та аналізувати файли системи контролю версії. Однією з найбільш поширених систем контролю версій є SVN. Для роботи з даною системою буде використано .NET збірку SharpSvn. При використанні імпортування даної збірки і додаванні посилання на нього в Solutation Explorer (References), нам зразу ж стає доступним простір імен об’єкта. Основної уваги заслуговують об’єкти, які безпосередньо необхідні для отримання інформації про версію проекту, перелік змінених файлів, конкретні зміни певного файлу.

Для комунікації з мікроконтролером буде використана бібліотека UVSOCK. Дана бібліотека написана мовою програмування С. Готові бібліотеки рішень, написані на мові програмування С легко інтегруються у C# проект. За допомогою спеціальної конструкції можна з легкістю викликати бібліотечні функції.

Реалізація графічного інтерфейсу проводилась на основі технології WPF (Windows Presentation Foundation). Дана технологія призначена для організації взаємодії WIN-програм з користувачем, тобто WPF забезпечує інтерфейс користувача [24]. Головне новаторство технології полягає в тому, що вона використовує вектори для малювання своїх компонентів. На основі цих векторів можна малювати як прості елементи (елементи управління та двовимірну графіку), так і трьохвимірну графіку, анімацію та інші складні частини програми.

Мовою розробки було обрано C#. Версія платформи – .NET Framework 4.0, яка включає в себе усі використані технології. Синтаксис мови програмування C# виглядає дуже схожим на синтаксис мови Java [25]. Однак називати C# клоном Java неправильно. В дійсності і C#, і Java являються членами сімейства мов програмування, які базуються на C (куди також входять C, Objective C, C++), і тому вони розділяють подібний синтаксис.

Внаслідок того, що C# представляє собою гібрид із декількох мов, він є таким же синтаксично чистим, як і Java, майже таким ж простим, як VB і практично таким ж потужним та гнучким, як C++.

## Реалізація інтерфейсу користувача програмного засобу для автоматизованого тестування програмного забезпечення вбудованих систем

Розроблений інтерфейс користувача включає в себе мінімальний набір компонентів, що робить його дуже простим та зручним у користуванні, в той самий час, даний інтерфейс дає змогу користувачу в повній мірі застосувати весь функціонал розробленого засобу. При розробці графічного інтерфейсу користувача не слід забувати про 10 евристичних правил Якоба Нільсена [26], дотримання яких допоможе створити максимально дружній до користувача інтрерфейс.

Інтерфейс користувача складається з основного та додаткового вікон (див. Рис. 8 та Рис. 9). Основне вікно застосунку дозволяє користувачу вказати шлях до проекту, а також ввести назву конкретної функцію, тестування якої слід провести. Під час введення шляху до проекту користувач має можливість натиснути на допоміжний елемент (знак пошуку блакитного кольору), для виклику пошукового вікна, з допомогою якого користувач легко може переміщатися по файловій системі комп’ютера. Також, на головному вікні користувач має можливість почати процес тестування(аналіз тестованої функції та виконання тестових випадків), та зберегти результати тестування(характеристики функції, час виконання та граф інцидентності даної функції). Перед початком процесу тестування користувач може налаштувати значення глобальних змінних у програмному забезпеченні вбудованої системи. Для цього користувачу слід натиснути кнопку «Global Vars», після чого буде викликано додаткове вікно програми.

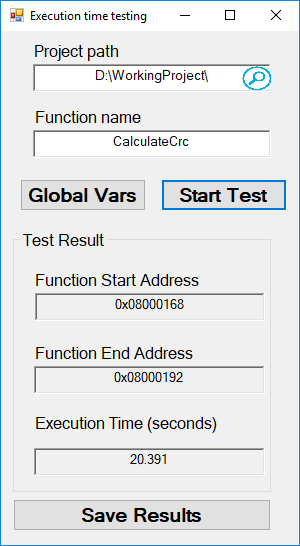


Рис. 8. Користувацький інтерфейс основного вікна програми

Додаткове вікно програми відповідає за налаштування глобальних змінних програмного забезпечення. Дане вікно складається з поля вводу кількості змінних, які потрібно налаштувати. Після введення значення числа змінних формується таблиця з відповідною кількістю рядків. Перший стовбець таблиці містить інформацію про назву змінної. Користувач не може ввести неправильну назву змінної, так як він має можливість вибрати лише існуючі значення з випадаючого списку(Рис. 10).

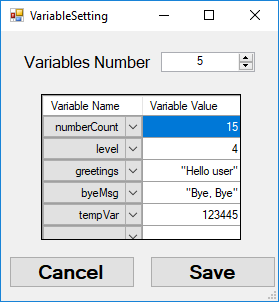


Рис. 9. Користувацький інтерфейс налаштування глобальних змінних

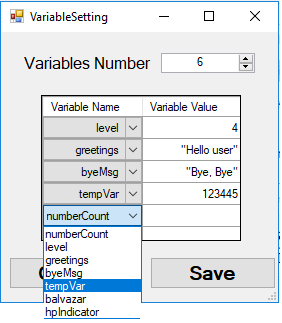


Рис. 10. Користувацький інтерфейс вибору існуючих глобальних змінних

Після завершення роботи з вікном налаштування глобальних змінних користувач може зберегти, або відмінити введенні зміни, натиснувши відповідно кнопку «Cancel» чи «Save», відповідно.

Якщо користувач вводить неправильне значення змінної (наприклад, для змінної типу int – стрічку), програма не дасть змоги зберегти зміни і попередить про це користувача(Рис. 11).

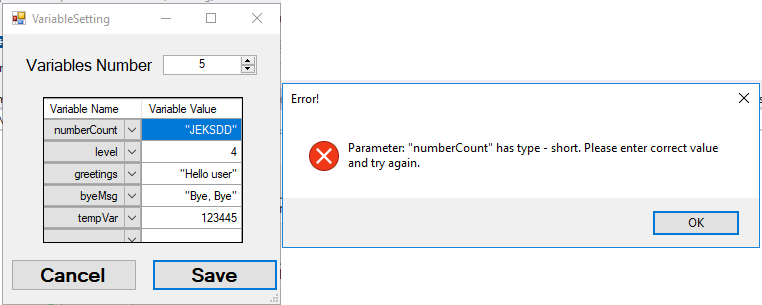


Рис. 11. Інтерфейс користувача, при неправильному введені даних.

## Розробка бази даних для зберігання результатів синтаксичного аналізу тексту програми вбудованої системи

Створення бази даних слід починати з її проектування. У результаті проектування визначена структура бази, тобто склад таблиць, їхня структура та логічні зв'язки. Структура реляційної таблиці визначається складом стовпців, їхньою послідовністю, типом даних кожного стовпця та їхнім розміром, а також ключем таблиці. Процес проектування можна здійснювати двома підходами. За першого підходу спочатку визначають основні задачі, для розв'язання яких створюється база, та потреби цих задач у даних. За другого підходу визначають предметну область (сферу), здійснюють аналіз її даних і встановлюють типові об'єкти предметної області. При розробці бази даних для засобу автоматизованого тестування програмного забезпечення вбудованих систем використана комбінація з обох підходів. Провівши аналіз предметної області[27] та визначивши ряд основних задач, розроблено базу даних(Рис. 11).



Рис. 11. Фізична модель бази даних для зберігання результатів синтаксичного аналізу тексту програми вбудованої системи

Основною сутністю спроектованої бази даних є Функція, дана сутність включає в собі наступні поля:

* FunctionId – ключове поле;
* FunctionName – назва функції;
* ExecutionTime – час виконання тестованої функції;
* isTested – прапорець, який вказує чи дана функція вже пройшла процес тестування;
* ReturnType – зовнішній ключ на сутність Тип даних. Даний зв’язок поєднує функцію з типом даних, який дана функція повертає як результат виконання;
* File – зовнішній ключ, на сутність Файл, даний зв’язок поєднує функцію з файлом, в якому вона знаходиться;
* Project – зовнішній ключ на сутність Проекту, даний зв’язок поєднує функцію з проектом, в якому вона знаходиться;
* Address – зовнішній ключ на сутність Адреса, даний зв’язок поєднує функцію з набором адрес, який включає в собі початкову адресу функції та кінцеву адресу;
* Matrix – зовнішній ключ на сутність Матриця (Інцидентності), даний зв’язок поєднує функцію з її відповідною матрицею інцидентності;
* Parameters – зовнішній ключ на сутність Набір Параметрів, даний зв’язок поєднує функцію з набором вхідних параметрів.

Допоміжними сутностями розробленої бази даних є:

* Файл – містить назву та повний шлях до файлів проекту;
* Проект – містить інформацію про повний шлях до проекту та версію проекту в системі контролю версій SVN[28];
* Набір адрес – інформація про початкову та кінцеву адреси функції;
* Тип даних – містить інформацію про типи даних програмного забезпечення вбудованої системи;
* Параметр – інформація про назву параметра та його тип;
* Набір параметрів – допоміжна таблиця, яка слугує для з’єднання сутності Функція та Параметр з відношенням багато до багатьох;
* Вершина – містить інформацію про вершину матриці інцидентності;
* Ребро(пара) – містить інформацію про зв’язок між двома вершинами матриці;
* Матриця – допоміжна таблиця, яка містить інформацію про всі ребра матриці.

Дана база даних була програмно реалізована мовою програмування C#, за допомогою простору імен System.Xml.Serialization та створеного статичного класу ProjectXMLSerializer, у вигляді XML файлу. Створений клас містить дві функції: SaveProjectBD та ReadProjectDB, які слугують для зберігання об’єкту у базу даних та читання інформації з бази даних у об’єкт відповідно.

Приклад збереженої бази даних наведений на Рис. 12.

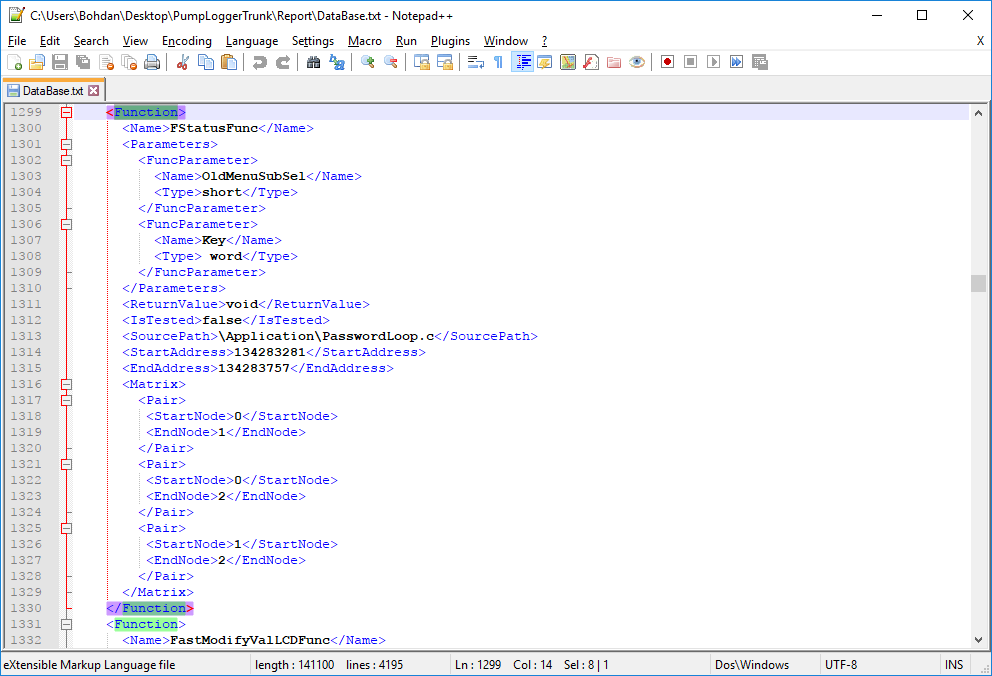


Рис. 12. Фрагмент тексту бази даних для зберігання результатів синтаксичного аналізу тексту програми вбудованої системи

## Розробка модуля взаємодії з програмним забезпеченням вбудованої системи, розробленої у Keil uVision

В основі даного модуля лежить інтерфейс uVision Socket. uVision Socket Interface (UVSOCK) – інтерфейс, що дозволяє керувати і відстежувати стан Keil uVision IDE, через сторонні застосунки. Обмін інформацією між застосунком та середовищем розробки Keil uVision, реалізований через стек протоколів TCP/IP, що дає можливість використання локально на одному ПК, де запущено Keil uVision IDE, так і віддалено через мережу. Інтерфейс UVSOCK може використовуватись двома способами:

- Клієнтський інтерфейс (UVSC) – передбачає контроль Keil uVision IDE через легкий для розуміння інтерфейс (API), що написаний мовою програмування C.

- Прямий інтерфейс (UVSOCK) – передбачає контроль Keil uVision IDE безпосередньо через стек TCP/IP. Цей підхід рекомендується використовувати у випадку, коли система на якій запущений клієнт, працює не під управлінням ОС Microsoft Windows.

Даний інтерфейс реалізований як динамічно приєднувальна бібліотека(dll)[29], написана мовою програмування С.

Основною проблемою при реалізації даного модуля була інтеграція бібліотеки в проект, написаний мовою програмування C#.  Перш за все, слід перенести всі структури даних, які використовуватимуться, з бібліотеки у проект. У більшості бібліотек чи програм реалізованих мовою програмування С використовується вирівнювання даних, або пакування. Даний підхід дозволяє зекономити пам’ять та ефективніше з нею працювати[30]. Щоб перенести вирівняні структури даних у C# проекті слід використати наступний атрибут для структур – StructLayout (LayoutKind.Explicit), а також явно зазначити позицію кожного поля структури за допомогою атрибуту FieldOffset(кількість байтів). Отже, для вирішення проблеми з вирівнюванням даних, структури, перенесені з бібліотеки, у С# проект мають наступний вигляд:

[StructLayout(LayoutKind.Explicit)]

public struct UVSOCK\_CMD\_RESPONSE

{

[FieldOffset(0)]

public UV\_OPERATION cmd;

[FieldOffset(4)]

public UV\_STATUS status;

}

Щоб мати можливість використовувати функції з динамічної бібліотеки, слід додати їх прототипи з наступним атрибутом – [DllImport("Назва бібліотеки", CallingConvention = CallingConvention.Cdecl)] з модифікатором extern. Даний мотифікатор вказує компілятору на те, що реалізація функції знаходиться поза межами даного проекту[31]. Отже, для використання функції з динамічної бібліотеки, її слід оголошувати наступним чином:

[DllImport("UVSC64.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern UVSC\_STATUS UVSC\_DBG\_ENTER(short iConnHandle);

Для повноцінної роботи модуля взаємодії з програмним забезпеченням вбудованої системи, розробленої у Keil uVision винесено наступні функції:

* UVSC\_Init – даний метод слугує для ініціалізації інтерфейсу uVision Socket Interface. Параметрами даного метода є діапазон портів, через які встановлюватиметься з’єднання з середовищем Keil uVision;
* UVSC\_OpenConnection – метод, за допомогою якого відкривається з’єднання з сесією середовища програмування Keil uVision. Вхідними параметрами даного метода є ім’я з’єднання, шлях до клієнта середовища програмування, режим роботи, функція зворотнього виклику для асинхронних подій, ім’я файлу логу, функція зворотнього виклику для логу подій. Вихідними параметрами є номер з’єднання та порт підключення;
* UVSC\_DBG\_ENTER – переведення режиму роботи Keil uVision у відлагодження. Вхідним параметром є номер з’єднання;
* UVSC\_DBG\_CREATE\_BP – створення точки зупинки у програмі. Вхідними параметрами є номер з’єднання, а також налаштування точки зупинки (адреса зупинки, тип зупинки і т.д.);
* UVSC\_DBG\_START\_EXECUTION – метод, за допомогою якого здійснюється початок виконання відлагодження програмного забезпечення. Вхідним параметром є номер з’єднання;
* UVSC\_DBG\_EXIT – вихід з режиму відлагодження. Вхідним параметром є номер з’єднання;
* UVSC\_CloseConnection – закриття з’єднання з середовищем програмування Keil uVision. Вхідним параметром є номер з’єднання та прапорець, яки вказує чи слід завершити роботу Keil uVision;

UVSC\_UnInit – деініціалізація інтерфейсу uVision Socket Interface. Вхідним параметром є номер з’єднання;

## Висновки до розділу 3

Описано програмну реалізацію засобу для автоматизації процесу проведення тестування програмного забезпечення вбудованої системи, розробленої у середовищі програмування uVision Keil.

Програмна реалізація даного алгоритму розроблена мовою програмування C#, cинтаксиc якої є доволі простим, а сама мова надає потужні можливості для розробки.

Розроблений засіб дозволяє проводити тестування своєчасності виконання заданих функцій вбудованої системи реального часу.

Програмний засіб працює у певній логічній послідовності, тому інтерфейс користувача є доволі простим. Інтерфейс користувача розроблений згідно 10-ти евристичних характеристик Нільсена, що робить його для користувача максимально легким в користуванні та зрозумілим.

Вимогою для функціонування програми є наявність встановленого пакету Microsoft .Net версії 4.0 та вище.

# Розділ 4. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ОТРИМАНИХ ДАНИХ

## 4.1. Опис процесу дослідження

Поданий алгоритм тестування програмного забезпечення вбудованої системи базується на визначенні часу виконання певного модуля програми. Для якісної оцінки запропонованого засобу, процес дослідження складається з двох частин. Перш за все, перед тим, як порівнювати даний засіб з аналогом, потрібно валідувати роботу алгоритму відносно «еталонної» функції. Під терміном «еталонна» функція передбачається функція, час виконання якої є сталим та відомим. Другим етапом дослідження є порівняння роботи розробленого засобу з аналогом для двох випадків:

* для простої функції, без зв’язку з периферійними пристроями;
* складніша функція, яка містить взаємодію з декількома периферійними пристроями.

Вибраний засіб-аналог є Bound-T, розроблений компанією Tidorum Ltd. Вибраний засіб використовується для обчислення часу виконання програмного забезпечення вбудованих систем на основі статичного аналізу. У основі Bound-T запропоновано метод розрахунку тривалості з урахуванням архітектури конвеєру та кеш-пам’яті мікроконтролера. Однак, навіть у випадку використання такого методу розрахунку, отримані показники тривалості виконання будуть завищеними, оскільки не враховується швидкість обміну даними між внутрішніми периферійними пристроями, та швидкість обміну даними через інтерфейсу зв’язку. Орім того, використання цих методів є недоцільними при аналізі тривалості виконання програмного коду, що виконується на мікроконтролерних вбудованих системах, адже більшість мікроконтролерів мають просту архітектуру конвеєру, а також у них відсутня кеш пам’ять. Таким чином, використання вищезгаданої вибірки дасть змогу отримати достатньо повні результати порівняльного аналізу розробленого засобу та засобу на основі статистичного аналізу.

## 4.2. Валідація розробленого засобу тестування програмного забезпечення вбудованих систем

Розглянемо декілька функцій, які можливо використати для валідації розробленого засобу:

1. Відомі алгоритми.

Складність полягає у виборі функції, тривалість якої наперед відома. Добре дослідженні функції легко змінюють тривалість виконання для проведення повноцінних експериментальних досліджень. Для прикладу, функція сортування даних на основі алгоритму сортування бульбашки не підходить, адже тривалість виконання залежить від типу вхідних даних, отже є не прогнозованою.

1. Стандартні функції затримки.

Використання бібліотечних функцій затримки, що входять до складу операційної системи не підходить з причини не прогнозованості тривалості виконання, адже такі функції зав’язані на планувальнику. Отже, як результат функція може виконуватися як за коротший, так і за довший період часу. Такий ефект зумовлений алгоритмом планувальника, який може передати управління готовому до виконання потоку з вищим пріоритетом;

1. Функція затримки, написана власноруч.

Даний підхід передбачає написання методу, який буде заповнений викликами функцій \_\_nop()[32]. Дана функція є еквівалентом асемблерної інструкції NOP(скорочено від “No OPeration”), яка наказує процесору нічого не робити. Одна команда NOP виконується за один такт. Таким чином, якщо відомо тактову частоту на якій працює мікроконтролер (кількість тактів за 1 секунду часу), можна легко розрахувати необхідну кількість NOP команд для створення односекундної затримки. Отже, розрахунок можна здійснити за формулою 4.1.

*NOP\_count = delay\_time \* clock\_frequency*, (4.1)

де NOP\_count – кількість NOP команд;

delay\_time – час затримки;

clock\_frequency – тактова частота вбудованої системи.

Вигляд функції односекундної затримки, основаної на NOP командах матиме наступний вигляд:

*OneSecondDelay()*

*{*

*while( -- NOP\_count ) { NOP(); }*

*}*

На перший погляд даний підхід є простим та оптимальним, адже тривалість виконання залежить від тактової частоти мікроконтролера і кількості інструкцій без операції(NOP). Насправді розрахунок є складнішим і розрахувати коректний час виконання NOP операції є практично неможливою задачею. Це зумовлено наступними чинниками, які слід враховувати при обчислені часу виконання:

* час вичитування інструкції з пам'яті програми;
* наявність кешу команд ;
* наявність конвеєру команд;
* точність даних, щодо мікроконтролера, адже в даташітах надають наближені дані для загального розуміння роботи мікроконтролера.

Також, беручи до уваги вищеописану функцію OneSecondDelay(), варто враховувати, що для декременту змінної NOP\_count буде використано ще один такт, ще два такти будуть використані для перевірки чи дана змінна не дорівнює нулю, а також, декілька тактів для збереження та відновлення станів регістру перед та після циклу. Під час виконання написаної функції може трапитися переривання, обробник якого призупинить виконання функції. Відповідно, час виконання написаної функції затримки збільшиться на час виконання обробника переривання.

Таким чином, даний підхід не дозволить валідувати розроблений алгоритм тестування програмного забезпечення вбудованої системи.

1. Функція затримки на основі апаратного таймера.

Апаратний таймер – це пристрій, який генерує переривання таймера через певний проміжок часу. Даний таймер є частиною ядра Cortex-M, відповідно бібліотека CMSIS містить опис цього таймера. Таймер є 24-х розрядним (приймає значення від 0 до ) і відраховує від заданого значення до нуля, після чого перевантажується і генерує переривання. Апаратний таймер підраховує кількість тактових сигналів мікроконтролера. Якщо кількість тактових сигналів досягла певної цифри (попередньо вказаної, під час налаштувань таймера), таймер спрацьовує і передає керування програми функцїі обробнику таймера. Саме це дозволяє нівелювати чинниками, описаними у попередньому підході.

Для створення функції затримки на основі апаратного таймера спершу слід налаштувати таймер на певну частоту спрацьовування, для прикладу таймер спрацьовуватиме кожну мілісекунду. Наступним кроком слід створити глобальну змінну, яка вказуватиме на кількість мілісекунд та налаштувати функцію обробник таймера, яка буде збільшувати на одиницю значення глобальної змінної. Дана функція має наступний вигляд:

*void SysTick\_Handler(void) {*

*ticks\_delay++;*

*}*

Сама функція виглядатиме так:

*void delay(uint32\_t milliseconds) {*

*uint32\_t start = ticks\_delay;*

*while((ticks\_delay - start) < milliseconds);*

*}*

Даний підхід, з використанням функції затримки на основі апаратного таймера, є найбільш оптимальним та точним. Використання такого підходу дозволить валідувати розроблений засіб тестування програмного забезпечення вбудованої системи. Для кращого результату буде здійснено вимірювання за допомогою цифрового осцилографу.

Для проведення дослідження додатково створено функцію blinkDiods, яка засвічує світлодіод, чекає 1 секунду часу ( реалізовано за допомогою описаної вище функції delay), гасить світлодіод, чекає 1 секунду та закінчує виконання. Операції засвічення та гасіння світлодіоду займають 2 такти, тому вони не несуть вагомого впливу на загальний час виконання функції. Код функції представлений нижче:

*void blinkDiods(void)*

*{*

*GPIOD->ODR = 0x9000;*

*delay(1000);*

*GPIOD->ODR = 0x0000;*

*delay(1000);*

*}*

Результати застосування розробленого засобу для тестування програмного забезпечення вбудованої системи надані в Таблиці 4.1.

*Таблиця 4.1.*

Результати виконання розробленого засобу

|  |  |
| --- | --- |
| Спроба № | Час виконання, мс |
| 1 | 2181 |
| 2 | 2181 |
| 3 | 2166 |
| 4 | 2169 |
| 5 | 2192 |
| 6 | 2179 |
| 7 | 2183 |
| 8 | 2182 |

*Продовження таблиці 4.1.*

|  |  |
| --- | --- |
| 9 | 2150 |
| 10 | 2183 |

Таким чином, середній час виконання функції blinkDiods, виміряний розробленим засобом становить 2176.6мс.

Також, було здійснено замір за допомогою цифрового осцилографа. Результат представлений на рис. 13.

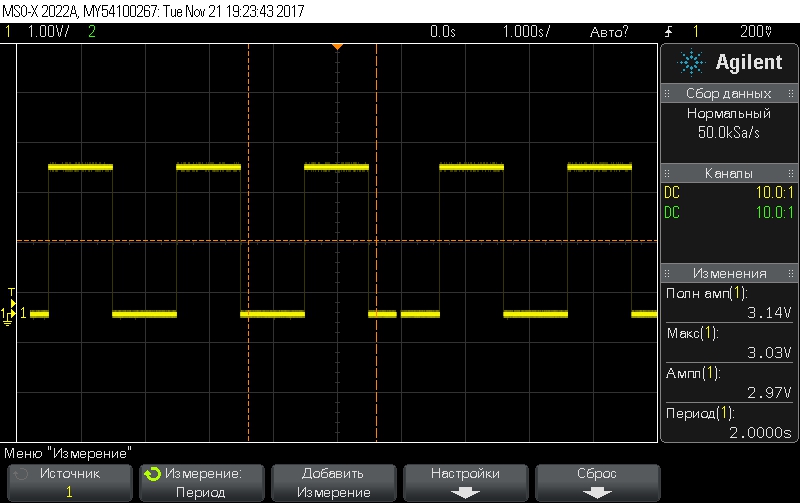


Рис. 13. Знімок екрану цифрового осцилографу.

Як показано на рис. 13 час виконання функції blinkDiods становить рівно 2 секунди. Вимір, здійснений за допомогою осцилографа можна вважати «еталонним».

Таким чином, вимірювання часу виконання фукнції за допомогою розробленого засобу має похибку рівну 176,6 мс. Дана похибка зумовлена обміном трьох пакетів даних між розробленим засобом та вбудованою системою. Дані пакети – це команда старту виконання програми, підтвердження початку виконання та інформація про те (відповідно і старт відліку часу), що програма досягнула другої точки зупинки, що означає закінчення виконання (відповідно закінчення відліку часу). Величина похибки напряму залежить від машини, з якої проводиться тестування, та її обчислювальних характеристик. Таким чином, можна вважати, що дана похибка є сталою величиною для даної обчислювальної машини.

## Проведення порівняльного аналізу з засобом-аналогом

Як було зазначено в підрозділі 4.1., програмою-аналогом вибрано засіб Bound-T, який базується на статичному аналізаторі. Модулем, який підлягатиме тестуванню, виступить лінійна функція, яка містить в собі світіння світлодіодами та декілька затримок. Код функції представлений нижче:

*void testingFunction(void){*

*uint8\_t i;*

*for(i = 0; i < 10; i++) {*

*GPIOD->ODR = 0x9000;*

*delay(500);*

*GPIOD->ODR = 0x0000;*

*delay(500);*

*GPIOD->ODR = 0x6000;*

*delay(5000);*

*GPIOD->ODR = 0x0000;*

*delay(500);*

*} }*

Результати подані у таблиці 4.2.

*Таблиця 4.2.*

Результати тестування функції testingFunction

|  |  |  |
| --- | --- | --- |
| Спроба № | Час виконання розробленого засобу з врахуванням похибки, мс | Час виконання засобу-аналогу, мс |
| 1 | 65122 | 63865 |
| 2 | 65169 | 63865 |

*Продовження таблиці 4.2.*

|  |  |  |
| --- | --- | --- |
| 3 | 65141 | 63865 |
| 4 | 65169 | 63865 |
| 5 | 65146 | 63865 |
| 6 | 65120 | 63865 |
| 7 | 65105 | 63865 |
| 8 | 65123 | 63865 |
| 9 | 65106 | 63865 |
| 10 | 65107 | 63865 |

Найгірший час виконання функції, виміряний за допомогою розробленого засобу, становить 65169 мс. Час, виміряний засобом Bound-T рівний 63865 мс. Таким чином, можна сказати, що розроблений засіб для тестування програмного забезпечення вбудованої системи точніше виміряє час виконання простих модулей, без взаємодії з периферійними пристроями. Точність виміру Bound-T є меншою на 2%.

Наступним модулем, результати тестування якого порівнюватимуться, буде складний модуль, який включає в собі два підмодуля:

* модуль взаємодії з периферійними пристроями. Даний підмодуль приймає дані з датчиків температури та тиску, включає взаємодію з зовнішньою пам’яттю;
* модуль обробки даних з периферійних пристроїв. Даний підмодуль включає декілька алгоритмів обробки даних.

Для такого модуля буде проведено 3 вимірювання: вимірювання часу виконання модуля в цілому, вимірювання часу виконання кожного з підмодулей окремо. Результати тестування подані у таблицях 4.3-4.5.

*Таблиця 4.3.*

Результати тестування модуля acquisitionTask

|  |  |  |
| --- | --- | --- |
| Спроба № | Час виконання розробленого засобу з врахуванням похибки, мс | Час виконання засобу-аналогу, мс |
| 1 | 185 | 155 |
| 2 | 184 | 155 |
| 3 | 186 | 155 |
| 4 | 183 | 155 |
| 5 | 184 | 155 |

*Таблиця 4.4.*

Результати тестування підмодуля взаємодії з периферійними пристроями

|  |  |  |
| --- | --- | --- |
| Спроба № | Час виконання розробленого засобу з врахуванням похибки, мс | Час виконання засобу-аналогу, мс |
| 1 | 131 | 102 |
| 2 | 132 | 102 |
| 3 | 128 | 102 |
| 4 | 131 | 102 |
| 5 | 130 | 102 |

*Таблиця 4.5.*

Результати тестування підмодуля обробки даних

|  |  |  |
| --- | --- | --- |
| Спроба № | Час виконання розробленого засобу з врахуванням похибки, мс | Час виконання засобу-аналогу, мс |
| 1 | 55 | 53 |

*Продовження таблиці 4.5.*

|  |  |  |
| --- | --- | --- |
| 2 | 53 | 53 |
| 3 | 53 | 53 |
| 4 | 54 | 53 |
| 5 | 56 | 53 |

Найгірший час виконання функції, виміряний за допомогою розробленого засобу, становить 132мс. Час, виміряний засобом Bound-T, рівний 102мс. Точність виміру Bound-T є меншою на 23%. Судячи з вимірів, дана неточність зумовлена неможливістю статичного аналізатора визначити швидкість обміну даними через інтерфейс зв’язку з периферійними пристроями, адже найбільша недочність Bound-T допущена у розрахунку підмодуля роботи з периферією. В той самий час, результати вимірювання підмодуля обробки отриманих даних близькі до реального. Таким чином, можна сказати, що розроблений засіб для тестування програмного забезпечення вбудованої системи краще вимірює час виконання складних модулей, які містять в собі взаємодію з периферійними пристроями.

## Висновки до розділу 4

Для валідації створеного метода обрано функція затримки на основі апаратного таймера, яка має сталий та відомий час виконання. Додатково за джерело оцінки часу виконання функції обраний цифровий осцилограф.

Валідуючи програмний засіб за допомогою функції затримки та цифрового осцилографа виявлено, що розроблений метод має похибку у 176,6 мс, яка зумовлена передачею трьох пакетів між розробленим засобом та вбудованою системою. Похибка відрізняється, в залежності від ПК, на якому виконується розроблений засіб, що зумовлено різними обчислювальними можливостями комп’ютерів.

Проведено порівняльний аналіз створеного засобу з засобом-аналогом Bound-T, в основі якого лежить синтаксичний аналізатор, показав, що вимірювання часу виконання за допомогою синтаксичного аналізатора дає дещо завищені результати. Якщо модуль, який тестується є доволі тривіальним та не містить в собі взаємодії з периферійними пристроями, то Bound-T показує доволі близькі результати, похибка яких становить 2%. Якщо модуль, який тестується є складним та містить взаємодію з периферійними пристроями, то засоби, основані на синтаксичному аналізу, дають неточні результати.

Провівши дослідження, можна сказати, що розроблений програмний засіб для проведення автоматизованого тестування програмного забезпечення вбудованої системи є ефективнішим, ніж аналоги.

# Розділ 5. ЕКОНОМІЧНА ХАРАКТЕРИСТИКА ПРОГРАМНОГО ПРОДУКТУ

## 5.1. Економічна характеристика проектного рішення

Метою дипломного проекту є розробка засобів для автоматизованого тестування програмного забезпечення вбудованих систем, які розроблені у середовищі програмування Keil uVision. Кінцевим продуктом даного проекту, є алгоритм тестування та його програмна реалізація. Призначенням цієї системи є допомога розробникам та тестувальникам програмного забезпечення вбудованих систем у проведенні тестування. Тестування є обов’язковим етапом життєвого циклу програмного забезпечення.

Розроблена система створена як застосунок для комп’ютера. Основними ідеями для підвищення економічної ефективності є нівелювання недоліками ручного тестування. Дана система володіє наступним рядом переваг:

* зменшення часу для проведення тестування;
* проведення тестування комп’ютером, тобто, унеможливлення «людського фактору»;

Новизною даного проекту є те, що розроблений засіб дозволяє проводити тестування на вбудованій системі без внесення змін у програмне забезпечення. Таким чином ПЗ не є спотвореним і результати проведеного тестування є об’єктивними. Такий підхід робить дану систему особливою, аналогів якій на ринку немає.

Існуючи засоби для проведення автоматизованого тестування програмного забезпечення вбудованих систем:

1. KLEE - засіб для автоматизованого тестування програмного забезпечення. Тестові приклади згенеровані даним засобом забезпечили високий рівень покриття вихідного коду. Незважаючи на ряд переваг, представлений інструмент має і ряд недоліків: використання символьного виконання передбачає неможливість виконання тестових прикладів безпосередньо на апаратному забезпеченні, необхідність удосконалення даного інструменту для роботи з змінними з плаваючою крапкою, можливість роботи лише на операційній системі Linux;
2. CTGEN – платформа для автоматизованого генерування тестових прикладів, та їх символьного виконання. Вхідними даними для роботи даного модуля є вихідний код програми, що використовується для побудови графу потоку керування, який у свою чергу є вхідними даними для символьного генератору тестових прикладів. Розглянута платформа (модель) підходить для тестування функцій, або невеликих модулів програми і не може бути застосовано для тестування системи через великий простір станів, та тестування функцій для роботи з реальними периферійними пристроями.
3. Статистичні аналізатори коду (Lint, Parasoft, CppCheck і т.д.) – засоби, які проводять аналіз ПЗ, без реального виконання програми. Дані засоби розроблені для проведення формальних перевірок коду та виявлення місць, які можливо містять помилки. Статичний аналіз коду, і формальні методи верифікації мають проблеми з гнучкістю і не можуть бути застосовані для різноманітних вбудованих систем.

Розроблена система є актуальною, оскільки володіє рядом переваг відносно існуючих засобів автоматизованого тестування програмного забезпечення вбудованих систем, що дозволяє провести тестування більш якісно.

В даному розділі дипломного проекту проводиться економічне обґрунтування проектування та розробки засобу для проведення автоматизованого тестування ПЗ вбудованої системи. Зокрема розраховується комплексний показник якості проектного рішення, котрий відображає його переваги та недоліки.

В економічній частині дипломного проекту відображено доцільність проектного рішення через відповідні економічні розрахунки, проведено техніко-економічне обґрунтування розробки.

## 5.2. Розрахунок витрат на розробку та впровадження проектного рішення.

1) Витрати на розробку і впровадження програмного засобу (К) визначаються як:

, (5.1)

де – витрати на розробку програмного засобу, грн.;

 – витрати на відлагодження і дослідну експлуатацію програмного засобу на ЕОМ, грн.

Витрати на розробку програмного засобу включають в себе:

1. витрати на оплату праці розробників ();
2. єдиний соціальний внесок ();
3. вартість додаткових виробів, що закуповуються ();
4. транспортно-заготівельні витрати ();
5. витрати на придбання спецобладнання ();
6. накладні витрати ();
7. інші витрати ().

Для проведення розрахунків витрат на оплату праці необхідно визначити категорії працівників, які приймають участь в процесі проектування, їх чисельність, середньоденну заробітну плату спеціаліста відповідної категорії та трудомісткість робіт у людино-днях (людино-годинах).

До цієї статті належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам та іншим працівникам, безпосередньо зайнятим розробкою програмного забезпечення за конкретною темою (технічним завданням), обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з діючими в організації системами оплати праці, включаючи будь-які види грошових і матеріальних доплат.

Над даним проектом працюють: програміст з місячною заробітною платою 10010 грн, тестувальник з заробітною платою 6006 грн в місяць та керівник проекту – 12012 грн.

Середньоденна заробітна плата і-го розробника () обчислюється за формулою:

, (5.2)

де – основна місячна заробітна плата розробника і-ої спеціальності, грн.;

 – місячний фонд робочого часу, днів (22 дні – листопад 2017 року).

Таким чином – денна заробітня плата програміста:

ЗПд1 = 10010 / 22 = 455 грн;

Денна заробітна плата тестувальника:

ЗПд2 = 6006 / 22 = 273 грн;

Денна заробітна плата керівника проекту:

ЗПд3 = 12012 / 22 = 546 грн

Розрахунок витрат на оплату праці усіх розробників проекту обчислюємо за формулою:

, (5.3)

де – чисельність розробників проекту і-ої спеціальності, осіб;

– час, витрачений на розробку проекту працівником і-ої спеціальності, днів;

– денна заробітна плата розробника і-ої спеціальності, грн.

Розрахунок витрат на оплату праці розробників зводиться у таблицю 5.1.

*Таблиця 5.1.*

Розрахунок витрат на оплату праці

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Спеціальність  розробника | Кількість осіб. | Час роботи, дні | Денна заробітна плата розробника, грн. | Витрати на оплату праці, грн. |
|
| Програміст | 1 | 32 | 455 | 14560 |
| Тестувальник | 1 | 17 | 273 | 4641 |
| Керівник | 1 | 32 | 546 | 17472 |
| Всього | 3 | - | - | 36673 |

2) Витрати на оплату праці працівникам призводять до виникнення зобов'язань підприємства за єдиним соціальним внеском (2-й клас ризику -22%)

Вф = 36673\*0,22 = 8068,06 грн.

3) Витрати на додаткові вироби, що закуповуються (Вд) (папір, накопичувачі, тощо) визначаються за їхніми фактичними цінами з врахуванням найменування, номенклатури та необхідної їх кількості в проекті. Вихідні дані та результати розрахунків оформляються у таблицю 5.2. Транспортно-заготівельні витрати (*)* становлять 15% суми витрат на додаткові вироби, що закуповуються.

*Таблиця 5.2.*

Розрахунок витрат на закуплені вироби

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Найменування купованих виробів | Марка, тип | Кількість штук | Ціна за одиницю, грн | Сума витрат, грн. | Сума витрат з урахуванням транспортно-заготівельних витрат, грн. |
| Папір | MAESTRO, А4, , 500арк, клас C | 2 | 95,04 | 190,08 | 218,6 |
| Ручка | Ручка кулькова Economix BOLIDE | 10 | 5,04 | 50,4 | 57.96 |
| Тонер | 1 | 1 | 150 | 150 | 172,5 |
| Всього | - | - | - | 390.48 | 449,1 |

Отже, витрати на додаткові вироби, що закуповуються (*Вд*) дорівнюють 449,10 грн.

4) Накладні витрати (Вн) проектних організацій включають три групи видатків: витрати на управління; загальногосподарські; невиробничі витрати. Вони розраховуються за встановленими 24% до витрат на оплату праці.

Вн = 36673 ⋅ 0,24 = 8801,52 грн

5) Інші витрати (Він) складають видатки, які не враховані в попередніх статтях витрат. Вони розраховуються за встановленими 9% до витрат на оплату праці.

Він = 36673 ⋅0,09 = 3300,57 грн

6) Витрати на розробку проектного рішення обчислюємо за формулою:

K = BОП + ВФ + ВД + Вн + Він ; (5.4)

Підставивши значення у формулу 5.4, отримаємо:

*К* = 36673 + 8068,06 + 449,1 + 8801,52 + 3300,57 = 57292,25 (грн.).

Результати обрахунків зведені в табл. 5.3

*Таблиця 5.3.*

Кошторис витрат на розробку проектного рішення

|  |  |
| --- | --- |
| Назва елементів витрат | Сума витрат, грн |
| Витрати на розроблення проектного рішення, зокрема: |  |
| * витрати на оплату праці | 36673 |
| * відрахування у спеціальні державні фонди | 8068,06 |
| * відрахування на додаткові вироби, що закуповуються | 449,10 |
| * накладні витрати | 8801,52 |
| * інші витрати | 3300,57 |
| Разом | 57292,25 |

## 5.3. Визначення комплексного показника якості

Комплексний показник якості *(ПЯ)* визначається шляхом порівняння показників якості проектованої системи і вибраного аналогу.

Аналогом ми обрали програмний продукт «Parasoft», котрий відповідає розробленому рішенню (проектованій системи) за сферою застосування та функціональним призначенням і є представлений на ринку автоматизованого тестування програмного забезпечення вбудованих систем.

Для визначення *Пя* використовуємо систему показників технічного рівня і якості, яка містить такі групи, причому в кожній групі вказана в дужках мінімальна кількість показників:

1. показники призначення;
2. показники надійності;
3. зручність використання;
4. супроводжуваність;
5. інші показники.

Комплексний показник якості проектованої системи визначаємо методом арифметичного середньозваженого з формули:

*Пя= ,* (5.5)

де *m* – кількість одиничних показників (параметрів), прийнятних для оцінки якості проектованої системи;

*qi* – коефіцієнт вагомості кожного з параметрів щодо їхнього впливу на технічний рівень та якість проектованої системи (встановлюється експериментальним шляхом), причому:

=1,0 (5.6)

*Сі* – часткові показники якості, визначені порівнянням числових значень одиничних показників проектованої системи і аналога за формулами:

Сі= або Сі =  (5.7)

де *Пп*рі та *П* - кількісні значення і-го одиничного показника якості відповідно проектованої системи і аналога.

З попередніх двох формул вибираємо ту, в якій збільшення відповідає покращенню показника якості проектованої системи. Результати розрахунку зведені у таблицю 5.4.

*Таблиця 5.4*

Показники якості проектованої системи та аналога

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Найменування показника | | Числові значення | | Відносний показник (С) | Коефіцієнт вагомості (q) | C\*q |
| Аналог | Проектована система |
| Функціональна повнота | | 8 | 9 | 1,125 | 0,15 | 0,16875 |
| Правильність | | 8 | 8 | 1 | 0,1 | 0,1 |
| Сумісність | | 6 | 6 | 1 | 0,05 | 0,05 |
| Захищеність | | 9 | 7 | 1,286 | 0,05 | 0,0643 |
| Безвідмовність | | 10 | 9 | 1,11 | 0,1 | 0,111 |
| Відновлюваність | | 6 | 9 | 1,5 | 0,1 | 0,15 |
| Зрозумілість | | 6 | 9 | 1,5 | 0,1 | 0,15 |
| Оперативність | | 9 | 9 | 1 | 0,05 | 0,05 |
| Аналізованість | | 7 | 10 | 1,42 | 0,05 | 0,071 |
| Актуальність | | 7 | 8 | 1,14 | 0,1 | 0,114 |
| Ступінь новизни | | 6 | 10 | 1,67 | 0,15 | 0,2505 |
| Разом | - | | - | - | 1 | 1,28 |

Отже, комплексний показник якості (Пя) рівний 1,28.

## 5.4. Визначення експлуатаційних витрат

При порівнянні програмних засобів в експлуатаційні річні витрати включають вартість підготовки даних (E1) і вартість годин роботи ПК (E2) і визначаються за формулою:

*ЕП(А)=Е1П(А)+Е2П(А)* , (5.8)

де *ЕП(А) -* одноразові експлуатаційні витрати на проектне рішення (аналог), грн.;

*E1П(А) -* вартість підготовки даних для експлуатації проектного рішення (аналогу), грн.;

*E2П(A) -* вартість машино-годин роботи ПК для проектного рішення (аналогу), грн.

Річні експлуатаційні витрати визначаються за формулою:

*В(е)П(А)= ЕП(А)⋅NП(А)* , (5.9)

де *В(е)П(А)* - експлуатаційні річні витрати проектного рішення;

*NП(А)* - періодичність експлуатації проектного рішення (аналогу), раз/рік.

Вартість підготовки даних для роботи на ПК (E1) визначаються за формулою:

 (5.10)

де *і* - номери категорій персоналу, які беруть участь у підготовці даних;

*nr-* кількість співробітників і-ї категорії, чол.;

*ti* - трудомісткість роботи співробітників *і*-ї категорії, дні;

*-* середньогодиннаставка робітника *і*-ї категорії з врахуванням відрахувань ЄСВ, грн./год.

Середньогодиннаставка оператора визначається за формулою:

 (5.11)

де ЗПгi *-* основна місячна зарплата працівника *і*-ї категорії, грн.;

*b* - коефіцієнт, який враховує ЄСВ (b=0,22);

*Фг -* місячний фонд робочого часу, год.

Отже, для проектного рішення середньогодиннаставка становить:

ЗПг1 = 12012 *⋅*(1+0,22) / (22*⋅*8) = 83,27 грн.

ЗПг2 = 10010 *⋅*(1+0,22) / (22*⋅*8) = 69,39 грн.

ЗПг3 = 6006 *⋅*(1+0,22) / (22*⋅*8) = 41,63 грн.

Тоді одноразові експлуатаційні витрати на проектне рішення становлять:

*EП =*(1*⋅*32*⋅* 83,27) +(1*⋅*32*⋅* 69,66) + (1*⋅*17*⋅* 41,63) = 5601,47 грн.

A річні експлуатаційні витрати з урахуванням того, що періодичність експлуатації проектного рішення дорівнює 12 разів/рік, становлять:

*В(е)П* = 5601,47*⋅*12 = 67217,64 грн.

Над проектом-аналогом працює 1 керівник проекту, 4 розробники та 2 тестери. Їхні місячні заробітні ставки відповідно 9000 грн., 7000 грн. та 5000 грн. Тоді середньогодинна ставка для кожного з них становить:

ЗПг1 = 9000⋅ (1+0,22) / (22⋅ 8)= 62,39 грн.

ЗПг2 = 7000⋅ (1+0,22) / (22⋅ 8) = 48,52 грн.

ЗПг3 = 5000⋅ (1+0,22) / (22⋅ 8) = 34,65 грн.

Тоді одноразові експлуатаційні витрати на аналог становлять:

*EА* =(1*⋅*32*⋅*62,39)+(4*⋅*32*⋅*48,52)+(2*⋅*17⋅34,65) = 9385,14 грн.

A річні експлуатаційні витрати з урахуванням того, що періодичність експлуатації аналогу дорівнює 12 раз/рік, становлять:

*B(е)А*=9385,14 *⋅*12 = 112621,68 грн.

Вихідні дані та результати розрахунків витрат на підготовку даних для експлуатації на EOM зводяться у табл. 5.5.

*Таблиця 5.5*

Розрахунок витрат на підготовку даних для роботи на EOM

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Категорія  персоналу | Чисельність співробітників і-ої категорії, люд., | | Час роботи співробітників і-ої категорії, год. | | | Середньогодинна ЗП співробітника і-ої категорії, грн. | | Витрати на підготовку даних, грн. |
| 1 | 2 | | 3 | | | 4 | | 5 |
| Проектне рішення | | | | | | | | |
| Керівник проекту | 1 | | 4 | | | 83,27 | | 333,08 |
| Розробник | 1 | | 6 | | | 69,39 | | 416,34 |
| Тестувальник | 1 | | 4 | | | 41,63 | | 166,52 |
| Всього | - | | - | | | - | | 915,94 |
| Аналог | | | | | | | | |
| Керівник проекту | | 1 | | 4 | 62,39 | | 249,56 | |
| Розробник | | 4 | | 24 | 48,52 | | 1164,48 | |
| Тестувальник | | 2 | | 8 | 34,65 | | 277,2 | |
| Всього | | - | | - | - | | 1691,24 | |

Витрати на підготовку даних для розроблюваного програмного продукту є суттєво меншими ніж витрати на підготовку даних для програми аналога.

## 5.5. Розрахунок ціни споживання проектного рішення

Ціна споживання *(Цс)* – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

*ЦC(П) = ЦП*+*B(E)NPV* (5.12)

де *ЦП* – ціна придбання проектного рішення, грн.;

*B(E)NPV*  – теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

*ЦП* (5.13)

де *Пр -* норматив рентабельності (приймаємо 25%);

*K0 -* витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті, грн.,

*K0=* 1500 грн.;

*Кк -* витрати на доукомплектування технічних засобів на об'єкті, грн.,

*Кк* = 0 грн.;

*СПДВ -*ставка податку на додану вартість (20 %).

*ЦП* = 57292,25 *⋅* (1 + 25/100)*⋅* (1 + 0,2) + 1500 + 0 = 87438,375 грн.

Згідно ринкових цін, ціна аналога становить ЦА=132000,00 грн.

Теперішня вартість витрат на експлуатацію проектного рішення розраховується за формулою:

*B(E)NPV* =  (5.14)

де *B(E)Пt -* річні експлуатаційні витрати в *t*-омуроці, грн.; *T-* період експлуатації проектного рішення, 3 роки; *R* - річна ставка проценту банків (15%).

*B(E)NPV=* 67217,64 / (1 + 0,15)1 + 67217,64 / (1 + 0,15)2 + 67217,64 / (1+ 0,15)3 = 153473,00 грн

Таким чином ціна споживання проектного рішення становить:

*ЦС(П)* = 87438,375 + 153473,00= 240911,375 грн

Аналогічно визначається ціна споживання для аналогу. Визначимо теперішню вартість витрат на експлуатацію аналогу. Термін експлуатації аналогу становить 3 років, тоді за формулою 5.15:

*B(E)NPV=* 132000/ (1 + 0,15)1 + 132000/ (1 + 0,15)2 + 132000/ (1 + 0,15)3 = 301385,72 грн

Таким чином ціна споживання аналогу становить:

*ЦС(А)* = 132000,00 + 301385,72 = 433385,72грн.

## 6.6. Визначення показників економічної ефективності

1) Показник конкурентоспроможності:

 (5.15)

ККС =433385,72⋅1,28 / 240911,375= 2,30

2) Економічний ефект в сфері експлуатації:

ЕЕКС = В(Е)А - B(Е)П (5.16)

ЕЕКС= 112621,68 – 67217,64= 45404,04 грн.

3) Економічний ефект в сфері проектування:

*EПР*= *ЦА - ЦП*  (5.17)

EПР = 132000,00 - 87438,375 = 44561,625 грн.

1. Додатковий економічний ефект в сфері експлуатації (3 роки):

(5.18)

ЕЕКС Д*=* 45404,04⋅ (1,150+1,151+1,152) = 157665,53 грн*.*

1. Додатковий економічний ефект в сфері проектування:

 (5.19)

*EПР Д* =44561,625⋅ 1,52 = 67733,67 грн

1. Термін окупності витрат на проектування рішення:

*ТОК =* (5.20)

ТОК = 57292,25/ 45404,04 = 1,26 року aбo 1 рік і 3 місяці.

Результуючі показники економічної ефективності зводяться у табл. 5.6.

*Таблиця 5.6*

Показники економічної ефективності проектного рішення

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування показників | Одиниці  вимірювання | Значення показників | | |
| Аналог | | Проектне  рішення |
| 1 | 2 | 3 | | 4 |
| 1. Капітальні вкладення | грн. | | - | 57292,25 |
| 2. Ціна придбання | грн. | | 132000 | 87438,375 |
| 3. Річні експлуатаційні витрати | грн. | | 112621,68 | 67217,64 |
| 4. Ціна споживання | грн. | | 433385,72 | 240911,375 |
| 5. Економічний ефект в сфері експлуатації | грн. | | - | 45404,04 |
| 6. Додатковий економічний ефект в сфері експлуатації | грн. | | - | 157665,53 |
| 7. Економічний ефект в сфері проектування | грн. | | - | 44561,625 |
| 8. Додатковий економічний ефект в сфері проектування | грн. | | - | 67733,67 |
| 9. Термін окупності витрат на проектування рішення | місяці | | - | 1,26 (1 рік, 3 місяці) |
| 10. Коефіцієнт  конкурентоспроможності | - | | - | 2,30 |

## 6.7. Висновки до розділу 5

У цьому розділі дипломного проекту було проведено економічну оцінку проектного програмного продукту та його аналогу «Parasoft». Після аналізу отриманих результатів можна зробити висновок, що дана система є конкурентоспроможною, оскільки коефіцієнт конкуренто-спроможності становить 2,30.

Отже, сума витрат на розробку проектного рішення рівна 57292,25 грн. Ціна розробленого продукту буде становити 87438,375 грн, що являється меншим, ніж ціна аналогу - 132000,00 грн. Відповідно ціна споживання продукту аналогу, - 433385,72 грн набагато перевищує ціну споживання розробленої системи - 240911,375грн.

Отож, згідно проведених розрахунків можна стверджувати, що розроблювані засоби для проведення автоматизованого тестування програмного забезпечення вбудованих систем є кращою від її аналогу за економічними показниками.

Даний проект враховує підтримку актуальності свого функціоналу протягом 3 років, його підтримку та подальше розширення. Малий термін окупності витрат на проектування, який становить всього 1 рік та 3 місяці, також прогнозує економічну вигідність продукту.

# Висновки

В магістерській кваліфікаційній роботі розглянуто існуючі засоби автоматизованого тестування програмного забезпечення вбудованої системи, запропоновано алгоритм для проведення автоматизованого тестування програмного забезпечення вбудованої системи, а також проведено аналіз основних вимог до програмного продукту. Для програмної реалізації створеного алгоритму використано інтеграційне середовище розробки Visual Studio 2015.

Проведено валідацію розробленого програмного засобу за допомогою цифрового осцилографа та функції затримки на основі апаратного таймера. Процес валідації виявив сталу похибку, рівній 176,6 мс, яка зумовлена передачею трьох пакетів між розробленим засобом та вбудованою системою.

В рамках дослідження розробленого засобу проведений порівняльний аналіз з продуктом-аналогом Bound-T, в основі якого лежить статичний аналізатор, який показав, що вимірювання часу виконання за допомогою розробленого засобу дає точніші результати, ніж при застосуванні програмних засобів на основі синтаксичного аналізатора.

Науковою новизною даного дослідження є розроблений алгоритм, який дозволяє проводити тестування своєчасності виконання безпосередньо на апаратному забезпечені, без внесення змін у вихідний код програми вбудованої системи.

Практична цінність отриманих результатів полягає у тому, що реалізоване програмне забезпечення дозволяє проводити автоматизоване тестування ПЗ вбудованих систем, розроблених у Keil uVision.

За результатами дослідження опубліковано матеріали доповіді 14-ої міжнародної конференції CADSM[33], а також апробовано у VІІ Міжнародному форумі молодих вчених «Litteris et Artibus», на конференції «Computer Science and Engineering 2017» з доповіддю на тему «The model of software execution time remote testing».

Проведено економічну характеристику розробленого програмного продукту. Згідно проведених розрахунків можна стверджувати, що розроблювані засоби для проведення автоматизованого тестування програмного забезпечення вбудованих систем є кращою від її аналогу за економічними показниками. Термін окупності витрат на проектування становить всього 1 рік та 3 місяці.

# Перелік використаної літератури

1. Jack Ganssle, Michael Barr Embedded Systems Dictionary – San Francisco, CA CMP Books. – 305 p. – ISBN 1578201209.
2. Embedded Systems Glossary [Electronic resource] – Web page: https://barrgroup.com/Embedded-Systems/Glossary-A (viewed on May 5, 2017).
3. Бибило П.Н. Основы языка VHDL: Учебное пособие. Изд. 5-е. — М.:Книжный дом «ЛИБРОКОМ», 2012. — 328 с. — ISBN 978-5- 397-02584- 3.
4. Michael Barr, Anthony Massa Programming Embedded Systems, Second Edition with C and GNU Development Tools – New York, O'Reilly Media – 336 p.
5. Kaner Cem A Tutorial in Exploratory Testing – . Florida Institute of Technology, 2014 – 180 p.
6. Дідковська М.В., Тимошенко Ю.О. «Тестування. Основні визначення, аксіоми та принципи. Текст лекцій. Частина І»: навчальний посібник – КПІ – 62 с.
7. Тамре Л. Введение в тестирование программного обеспечения: Пер. с англ. – М.: Изд. дом “Вильямс”, 2003 – 368с.: ил
8. Kolawa, Adam; Huizinga, Dorota Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press, 2007 – 74 p. – ISBN 0-470-04212-5.
9. Roger S. Pressman Software Engineering: A Practitioner's Approach – New York, McGraw-Hill – 888 p. – ISBN 978-0078022128
10. Information technology : ISO 14598 [valid from 04-1999] – 45 p.
11. Shibu K V. Introduction to Embedded Systems - New Delhi : Tata McGraw - Hill Education, 2009 – 718 p.
12. CMSIS - Cortex Microcontroller Software Interface Standard [Electronic resource] – Web page: http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php (viewed on May 5, 2017).
13. STM32 Standard Peripheral Libraries [Electronic resource] – Web page: http://www.st.com/st-web-ui/active/en/catalog/tools/FM147/CL1794/SC961/SS1743/LN1939 (viewed on May 5, 2017).
14. Description of STM32F4xx HAL drivers [Electronic resource] –Web page: http://www.st.com/st-web-ui/static/active/jp/resource/technical/document/user\_manual/DM00105879.pdf (viewed on May 5, 2017).
15. Advantages of the µVision IDE for ARM. [Electronic resource] – Web page: http://www.keil.com/arm/idebenefits.asp (viewed on Sep 14 2017).
16. Cadar C. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs / C. Cadar, D. Dunbar, E. Dawson. // Proceedings of the 8th USENIX conference on Operating systems design and implementation. – Stanford, 2008. – С. 209–224.
17. Mangels T. Ctgen - a unit test generator for C / T. Mangels, J. Peleska. // 7th Systems Software Verification Conference. – Sydney, 2012. – С. 88–102.
18. Arcuri A. Black-Box System Testing of Real-Time Embedded Systems Using Random and Search-Based Testing / A. Arcuri, M. Z. Iqbal, L. Briand. // Proc. 22nd IFIP Int. Conf. Testing Software and Systems. – Natal, 2010. – С. 95–110.
19. Six issues in testing event-triggered real-time systems / [B. Lindström, R. Nilsson, A. Ericsson та ін.]. // School of Humanities and Informatics Tech. Rep. – Skvde, 2007.
20. Test Case Generation for Testing of Timeliness / [R. Nilsson, J. Offutt, J. Mellin та ін.]. // Proc. of the 2 Workshop on Model Based Testing. – Vienna, 2006.
21. Engblom J. Structured Testing of Worst-Case Execution Time Analysis Methods / J. Engblom, A. Ermedahl, F. Stappert. // 21st IEEE Real-Time Systems Symposium. – Orlando, 2000.
22. Michael B. Programming Embedded Systems: With C and GNU Development Tools / B. Michael, M. Anthony. – Sebastopol, California: O'Reilly Media, 2006 – 12p.
23. Xu H. X. High performance code compression architecture for the embedded ARM/THUMB processor / H. X. Xu, T. C. Clarke, R. S. Jones. // Сonference on Computing frontiers. – 2004. – №1.
24. WPF - опис. – www.paveldev.blogspot.com [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: www.paveldev.blogspot.com/2010/12/wpf-opisanie.html (дата звернення: 22.06.2017).
25. Троелсен Эндрю. Язык програмирования C# 5.0 и платформа .NET 4.5, 6-е изд. / Эндрю Троелсен / Пер. з англ. – М.: ООО «И.Д. Вильямс», 2013. – 47 с.
26. 10 Usability Heuristics for User Interface Design [Electronic resource] – Web page: https://www.nngroup.com/articles/ten-usability-heuristics/ (viewed on May 5, 2016).
27. Книш Б. Р. Засоби автоматизованого тестування програмного забезпечення вбудованих систем : 8.05010302 / Книш Богдан Романович – Львів, 2017. – 39 с.
28. Filippo L. Collaboration Tools for Global Software Engineering / L.Filippo, E. Christof, P. Rafael, V. Aurora., IEEE Software. – 52 с.
29. Dynamic-Link Libraries. [Electronic resource] – Web page: https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682589(v=vs.85).aspx (viewed on Sep 13.2017).
30. The Lost Art of C Structure Packing. [Electronic resource] – Web page: http://www.catb.org/esr/structure-packing/#\_alignment\_requirements (viewed on Sep 13 2017).
31. extern (довідник по C#). [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/extern (дата звернення: 13.09.2017).
32. Compiler User Guide: \_\_nop intrinsic. [Electronic resource] – Web page: http://www.keil.com/support/man/docs/armcc/armcc\_chr1359124998347.htm (viewed on Oct 23 2017).
33. Fedasyuk D. Architecture of a Tool for Automated Testing the Worst-Case Execution Time of Real-Time Embedded Systems’ firmware / D. Fedasyuk, R. Chopey, B. Knysh. // Experience of Designing and Application of Cad Systems in Microelectronics. – 2017. – №14.

# ДОДАТКИ

## Додаток А. План забезпечення якості програмного продукту

1. **Мета**

Цей план забезпечення якості встановлює процеси, методи, стандарти і процедури які будуть використовуватись для встановлення та забезпечення якості програмного продукту - «Засоби для автоматизованого тестування програмного забезпечення вбудованих систем розроблених у Keil uVision».

Для розробки проекту буде використана швидка (RAD) модель розробки програмного забезпечення. Особливістю такої моделі є розробка, яка здійснюється ітераційним шляхом її еволюційного розвитку з отриманням деякого варіанта системи–прототипу, на якому перевіряється реалізація вимог. Дана модель використовується для розробки систем, де головною вимогою є реалізація функцій системи. В разі використання даної моделі система послідовно розробляється з блоків конструкцій. В основу швидкої моделі розробки програмного забезпечення покладена спіральна модель.

Недоліки цієї моделі такі:

* Використання моделі може виявитися невдалим у випадку, якщо відсутні придатні для повторного використання компоненти;
* Виникає потреба в системі, яка може бути змодельована коректним чином;
* Існує ризик, що роботу над проектом не буде завершено;

Мета плану забезпечення якості полягає в забезпеченні відповідності процесу розробки вимогам до програмного продукту.

1. **Управління**
2. 1. **Організація**

Кожен учасник команди відповідає за якість виконання своєї роботи. Для виконання проекту призначається відповідальний за якість програмного проекту, в даному випадку це студент магістр. Відповідальний за якість керує всіма питаннями, пов'язаними із забезпеченням якості в проекті. Зокрема за вчасне виконання поставлених завдань, дотримування вимог до проекту, відповідність стандартам, ефективність процесу розробки та ефективність реалізації вимог до програмного забезпечення, відповідність вимогам та стандартам якості.

* 1. **Завдання**

Завдання з контролю якості включають в себе:

* Документування;
* Оглядові наради (експертизи, інспекції);
* Верифікацію;
* Валідацію;
* Види діяльності, спрямовані на поліпшення самого процесу забезпечення якості.

Ці завдання деталізовані в даному документі.

* 1. **Ролі та обов'язки**

Оскільки команда розробників складається з однієї людини, то роль інженера по якості програмного забезпечення буде виконувати розробник. Дана особа повинна мати чітке уявлення як продукт має бути реалізований. У зв’язку з невеликою кількість людей у команді, було вирішено поділити ролі таким чином, щоб не довелося залучати в команду додаткову особу, яка б виконувала роль тільки інженера з якості програмного забезпечення.

Також на проекті присутня роль власника продукту, який відповідає за формулювання вимог до програмного продукту та керівника проекту. Всі питання, що виникають власник продукту обговорює з розробником, чия робота напряму спрямована на результат та відповідність плану якості.

До обов’язків інженера з забезпечення якості належать:

* створення та підтримка плану забезпечення якості;
* виконання положень плану забезпечення якості, визначених документом, для перевірки дотримання якості та відповідності вимогам програмного продукту;
* надання необхідних пояснень, щодо цього документу, процесу забезпечення якості на проекті;
* перевірка дотримування плану якості;
* при необхідності оновлювання плану якості програмного забезпечення.

До обов’язків керівника проекту належать:

* затвердження положень та змін до процесу забезпечення якості;
* контроль за дотриманням виконання обов’язків розробника.

1. **Документація**
2. 1. **Мета**

У даному розділі визначається документація, що описує забезпечення якості і буде використана в рамка виконання проекту «Сервіс аналізу взаємозв’язків в соціальних мережах».

* 1. **Мінімальні вимоги до документації**

Повинні бути створені такі документи:

* SQAP: План забезпечення якості (даний документ).
* SRD: Специфікація вимог до програмного забезпечення.
* SDD: Дизайн програмного забезпечення.
* STD: Документація з тестування програмного забезпечення.
* Плани проведення верифікації та валідації.
* План по виявленню та класифікації ризиків.
* План супроводу.
  + 1. **Опис вимог до програмного забезпечення (SRD)**

SRD визначає вимоги для програмного продукту, які виконують певні функції в конкретному середовищі. Опис вимог повинен визначати вимоги для конкретного програмного продукту. У ньому документі слід розглянути основні питання функціональності, зовнішні інтерфейси, продуктивність, атрибутів та обмежень на реалізацію. Кожна вимога повинна бути однозначно ідентифікована та визначена так, що її задоволення здатне бути об'єктивно перевіреним і підтвердженим.

Опис вимог складається з метою формування спільного бачення роботи системи всім учасниками проекту та надає вичерпний перелік вимог, розділених за їх класами. Документ формується на початкових етапах проекту та може доповнюватись по мірі необхідності.

Кроки для підтвердження відповідності опису вимог:

* При внесенні зміни в опис вимог документ провіряється на відповідність:
  + Внутрішні практикам та узгодження;
  + Міжнародним стандартам;
* Кожна нова\змінена вимоги повинні верифікується керівником продукту;
* Критерії відповіднсоті для опису вимог:
* Відповідність установленим практикам та домовленостям;
* Відповідність стандартам;
* Затвердження власником продукту.
  + 1. **Опис дизайну програмного забезпечення (SDD)**

SDD описує, як програмне забезпечення буде будуватися відповідно до вимог SRD. Документ повинен описувати компоненти й підкомпоненти розробки програмного забезпечення, включаючи бази даних і внутрішні інтерфейси. Він може бути підготовлений як архітектура продукту.

Кроки для підтвердження опису дизайну ПЗ:

* При внесенні змін в опис дизайну ПЗ документ перевіряється на відповідність:
  + функціональним та нефункціональним вимогам до продукту згідно опису вимог;
  + установленим практикам та стандартам.
* Потім документ презентується керівнику продукту (керівнику магістерської кваліфікаційної роботи);
* У випадку несхвалення документ потрібно переопрацювати з урахування зауважень керівника;
* При необхідності збереження поточних змін в документі опису дизайну ініціюється зміна до опису вимог, що обговорюється з власником продукту.

Критерії для підтвердження опису дизайну ПЗ:

* Інформованість керівника продукту щодо документу;
* Відповідність стандартам та визначеним практикам;
* Задоволення опису вимог до ПЗ.
  + 1. **Плани верифікації та валідації**

Верифікація та валідація використовується для визначення того, чи розроблений програмний продукт відповідає вимогам, і чи є програмне забезпечення відповідним до призначення і очікуванням користувачів. План включає в себе: аналіз, контроль, оцінку і тестування програмного продукту і процесів, які стосуються випуску продукту. Крім того, тестування програмного забезпечення, атестація і перевірка процесів застосовуються при інтеграції зовнішніх частин розроблюваного продукту.

План визначає завдання верифікації та валідації, входів і виходів, необхідних для підтримки потрібного рівня цілісності програмного забезпечення. Він також надає засоби тестування (інтеграційного та системного рівня) та розробки вимог SRD, щодо розробки, як описано в SDD, і щодо тестування, як описано в проектній документації.

Кроки для підтвердження плану верифікації та валідації:

* При внесенні змін в план верифікації та валідації документ провіряється на відповідність установленим практикам та стандартам;

Далі документ презентується власника продукту;

* У випадку несхвалення документ потрібно переопрацювати з урахування зауважень керівника;
* Критерії для підтвердження плану варифікації та валідації:
* Інформованість керівника продукту щодо документу;
* Відповідність стандартам та визначеним практикам.
  + 1. **Користувацька документація**

Користувацька документація містить опис встановлення, експлуатації, управління та обслуговування програмного продукту.

Документація описує контроль вихідних даних, вхідних послідовностей, параметри програми, обмеження, і всі інші необхідні дані для програмного продукту. Всі повідомлення про помилки визначені і описані. Описані всі коригувальні дії, направлені на виправлення помилок, описаних в повідомленнях про помилку. Документація поширюється на будь-яку частину вбудованого програмного забезпечення, з якими користувач взаємодіє безпосередньо.

Кроки для підтвердження користувацької документації:

* При внесенні змін в користувацьку документацію потрібно інформувати власника продукту;
* У випадку несхвалення потрібно врахувати зауваження та внести потрібні поправки.
* Критерії для підтвердження користувацької документації:
* Достовірність;
* Повнота;
* Несуперечливість;
* Чіткість та лаконічність.
  + 1. **Інші документи**

Керівник проекту, керівник продукту, чи інженер з забезпечення якості можуть включати в документ нову документацію, відносно якої відбуватиметься контроль забезпечення якості.

1. **Стандарти, практики, конвенцій, і метрики**

Даний розділ описує стандарти, практики, угоди і метрики, які використовуються в проекті. Ці матеріали покликані не тільки забезпечити якість проекту, але й отримати кількісні дані про сам процес контролю якості. Отримані дані повинні бути використані розробником для підвищення рівня самого програмного продукту.

*Стандарти.*

Для ведення документації використовуються стандарти IEEE з відповідними модифікаціями.

*Практики.*

* Оскільки відкладати забезпечення якості недоцільно, розробник (студент магістр) застосовує процедури забезпечення якості безпосередньо під час роботи.
* Для зменшення кількості помилок перед початком написання коду розробляються необхідні для нього тести;
* Всі артефакти проекту підлягають інспектуванню і всі вони доступні після випуску. Це досягається шляхом переміщенням артефактів в систему управління конфігураціями, що забезпечує доступ до вмісту в будь-який час.
* Для всіх процесів має бути проведений огляд можливості поліпшення хоча б раз, і результати цього огляду документуються окремо у письмовій формі.

*Угоди.*

Правила та стиль написання документів повинні відповідати стандартам.

Для полегшення супроводу при написанні коду застосовуються стандарти мови програмування C, а також дотримання стандарту написання коду MISRA C: 2004.

*Метрики.*

* Для кожного процесу і кожного документа повинні вимірюватися принаймні три показника.
* Час, витрачений розробником, на виконання кожної під задачі.
* Інтегральний показник якості за шкалою від 1 до 10 на основі самооцінки та оцінки керівника.
* Кількість дефектів на одиницю об'єму (наприклад, на тисячу рядків коду).
* Кількість виправлених дефектів.

1. **Тестування**

В процесі реалізації програмного продукту здійснюватиметься його тестування. Передбачається використання наступних видів тестування:

* **Функціональне тестування –** перевіряє чи реалізовані функціональні вимоги, тобто можливості ПЗ в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить продукт, які завдання вирішує.
* **Навантажувальне тестування –** проводиться для того, щоб оцінити поведінку програми із заданим очікуваним навантаженням.
* **Стресове тестування –** застосовується для встановлення границь пропускної здатності програми. Цей тип тестування проводиться для визначення надійності системи під час екстремальних або непропорційних навантаженнях і відповідає на питання про достатню продуктивність системи у випадку, якщо поточне навантаження значно перевищить очікуваний максимум.
* **Тестування швидкодії –** Проводиться з метою встановлення, як швидко працює система або її частина, під певним навантаженням. Оскільки буде опрацьовуватись графові дані великого обсягу.
* **Тестування зручності використання –** Виконується з метою визначення зручності використання ПЗ для його подальшого застосування

Проектування та опис тестових випадків здійснюється згідно стандарту ISO 839. Модульні тести не документуються. Процес тестування, повинен відбуватись за планом функціонального тестування. Цей план тестування створюється керівником проекту та узгоджується з замовником. Модульне тестування повинно проводитися в повному обсязі після внесення в проект будь-яких змін.

План функціонального тестування не повинен включати тестові випадки для перевірки функцій, тестування яких було здійснене за допомогою модульних тестів.

1. **Звіти про помилки та коригувальні дії**

Для звітності буде використовуватись система відслідковування дефектів, що буде оформлена як дошка на веб-сервісі kanbanize. Опис програмних дефектів здійснюється згідно стандарту ISO 839.

Керівник проекту повинен визначити процедуру звітності про проблеми в забезпеченні якості. Вимоги до процедури звітності:

* При виявленні дефекту необхідно описати його відповідно до вимог та повідомити про нього керівника проекту;
* Всі дефекти вносяться в електронну систему відслідковування дефектів.
* Звіт про проблему повинен містити:
* Відомості про частину системи, в якій виник дефект;
* Час винекнення ефекту;
* Дії, що призвели до винекнення дефекту, в разі наявності такої інформації.

1. **Засоби, методи і метрики**

Керівник проекту повинен вимірювати здатність команди передбачати та долати ризики, здавати проект у заплановані сроки та вести статистику порушень якості продукту відносно порушених вимог, програмних модулів, порушників тощо.

Метрикою продуктивності є відношення виконаних завдань до запланованих завдань.

Метриками якості є кількість відкритих дефектів та кількість закритих дефектів.

1. **Контроль середовища**

Для визначення середовища для кожного проміжного результату та документації, у тому числі процесу копіювання і відновлення буде застосовуватись додаткове програмне забезпечення, а саме: система контролю версій SVN та веб-сервіс TortoiseSVN для управління нею, який надає можливість резервного копіювання проекту.

Проект буде знаходитись на сервері з обмеженим доступом, що забезпечить його від пошкодження на всіх етапах життєвого циклу програмного забезпечення.

1. **Контроль розробника**

Для забезпечення контролю якості після кожної внесеної зміни буде проводитися ручне модульне тестування для перевірки наявності дефектів в роботі програмного забезпечення.

1. **Збір коментарів, їх підтримка і збереження**

В процесі забезпечення якості продукту керівник проекту може збирати таку осовну інформацію про стан якості на проекті:

* Звіти про порушення якості
* Вимоги до системи
* Стандарти
* Плани тестувань

Ця інформація повинна бути надана замовнику по його вимозі.

1. **Навчання**

Для навчання і покращення програмного коду буде використовуватися методика Code Review, що дозволить покращити якість програмного продукту.

1. **Управління ризиками**

Для кожного ризику визначається його тип та вплив. Ці параметри використовуються для подальшої його оцінки керівником проекту. Ризики та їх параметри можуть змінюватися впродовж виконання проекту. Управління ризиками здійснюють керівник проекту та замовник.

Основними ризиками є:

1. Технологічні:
   1. Несумісність програмного забезпечення з апаратним (середній пріорітет);
   2. Непередбачувані технологічні складності (середній пріоритет);
   3. Наявність дефектів у підключених бібліотеках (низький пріоритет);
   4. Несумісність версій використовуваних технологій і фреймворків (середній пріорітет);
2. Організаційні:
   1. Зміни в вимогах (середній пріоритет);
   2. Не правильне планування (середній пріорітет);
   3. Недостатньо часу для реалізації програмного продукту;
3. Зовнішні ризики:
   1. Зміни в бізнес-середовищі (високий пріоритет);
   2. Паралельний розвиток конкурентоспроможних продуктів (високий пріоритет);

Зміни законодавчої бази держави (високий пріоритет);

## Додаток Б. Інструкція користувача

1. **Компоненти ПЗ**

Програма написана на мові програмування C# у середовищі програмування Microsoft Visual Studio .Net 2015 і може бути використана лише під управлінням операційної системи сімейства Windows.

Для експлуатації програмного засобу потрібно встановити збірку класів .Net Framework версії 4.0 та вище. Частота процесора повинна бути не менше 1 GHz, а оперативна пам’ять 512 Mb або більше. Перелік файлів, які необхідні для повноцінної роботи програми наведено у таблиці Б.1.

*Таблиця Б.1.*

Набір файлів для коректної роботи системи

|  |  |  |
| --- | --- | --- |
| № | Файл | Призначення |
| 1 | AutomativeTest.exe | Виконавчий файл |
| 2 | Microsoft.Office.Interop.Word.dll | Бібліотека для роботи з документами MS Office |
| 3 | SharpSvn.dll | Бібліотека для роботи з системою котролю версії SVN |
| 4 | UVSC64.dll | Бібліотека для віддаленого керування середовищем програмування Keil uVision |

Для повноцінної роботи ПЗ слід підключити до ПК вбудовану систему через програматор. Проект вбудованої системи повинен бути скомпільованим у інтеграційному середовищі розробки Keil uVision. Також під час роботи ПЗ повинне бути запущено середовище розробки Keil uVision з проектом, який підлягає тестуванню.

1. **Встановлення ПЗ**

Дане програмне забезпечення не потребує встановлення. Для її виконання достатньо запустити на виконання файл AutomativeTest.exe.

1. **Базові функції програмного забезпечення**

Програмний засіб дозволяє провести автоматизоване тестування часу виконання програмного забезпечення вбудованої системи. Даний засіб дозволяє визначити час виконання вибраних модулів ПЗ вбудованої системи.

Для початку тестування слід виконати такі дії у головному вікні програми:

1. Натиснути на поле «Project path» та вибрати шлях до проекту, який слід протестувати;
2. Вибрати функцію, час якої потрібно виміряти;
3. При потребі встановлення значень глобальних змінних, слід натиснути кнопку «Global Vars»
   1. В полі «Variables Number» встановити число глобальних змінних, значення яких потрібно змінити;
   2. З випадаючого списку вибрати потрібну змінну та у полі «Variable Value» встановити її значення
   3. Для збереження налаштувань глобальних змінних слід натиснути кнопку «Save». Для скасування налаштувань – кнопку «Cancel».
4. Для початку тестування вибраної функції слід натиснути кнопку «Start Test»; Після закінчення тестування вибраної функції, на головному вікні програми будуть відображені наступні значення:

* Початкова адреса функції;
* Кінцева адреса функції;
* Час виконання фукнції;

1. Для збереження результату виконання слід натиснути кнопку «Save Results» та вибрати шлях для створеного звіту тестування.
2. **Аналіз помилок та можливих проблем**

У разі виникнення помилок в роботі програмного засобу зверніться, будь-ласка, до розробника за допомогою електронної пошти. Адрес електронної скриньки knysh.bohdan@gmail.com

## Додаток В. Сертифікат участі на конфереції «Computer Science and Engineering 2017»



## Додаток Д. Фрагменти коду

Д.1. Вихідний код функції знаходження кінцевої адреси функції

/\*Parsing assembly code to get end address shift of function\*/

private int GetFunctionEndAddressShift(string filePath, string functionName)

{

int endAddressShift = 0;

List<string> functionBody = new List<string>();

using (StreamReader sr = new StreamReader(filePath))

{

string fileLine = "";

bool startFinded = false;

while ((fileLine = sr.ReadLine()) != null)

{

if(fileLine.Contains(functionName + " PROC"))

{

startFinded = true;

}

/\* Storing to local variable only assembly code of function \*/

if(startFinded && !fileLine.Contains(";;;"))

{

functionBody.Add(fileLine);

if (fileLine.Contains("ENDP"))

break;

}

}

}

/\*If function is very small or empty at all\*/

if(functionBody.Count >= 3)

{

string endAddressHex = "";

foreach (string line in functionBody)

{

if (line.Contains("POP"))

{

endAddressHex = line.Substring(0, line.IndexOf(' '));

return int.Parse(endAddressHex, System.Globalization.NumberStyles.HexNumber);

}

}

if (endAddressHex == "")

{

endAddressHex = functionBody[functionBody.Count - 2].Substring(0, functionBody[functionBody.Count - 2].IndexOf(' '));

return int.Parse(endAddressHex, System.Globalization.NumberStyles.HexNumber);

}

}

return 0;}

Д.2. Вихідний код функції взаємодії з вбудованою системою

public static TimeSpan GetFunctionRunTime(int StartAddress, int EndAddress)

{

const int MAX\_OP\_TRIES = 3;

int trynumber = 0;

connectionHndl = 0;

port = 0;

uvsc\_cb newCallback = new uvsc\_cb(keilCallback);

UVSC\_STATUS operationStatus = UVSC\_STATUS.UVSC\_STATUS\_FAILED;

UVSC\_RUNMODE uvRunmode = UVSC\_RUNMODE.UVSC\_RUNMODE\_NORMAL;

String startAddress = StartAddress.ToString();

String endAddress = EndAddress.ToString();

BKRSP[] bkrsp = new BKRSP[128];

int bkRspSize = System.Runtime.InteropServices.Marshal.SizeOf(typeof(BKRSP));

int bkParamsSize = System.Runtime.InteropServices.Marshal.SizeOf(typeof(BKPARM));

// Creating object of 1st BP params

BKPARM bk1params = new BKPARM();

bk1params.type = BKTYPE.BRKTYPE\_EXEC;

bk1params.count = 1;

bk1params.accSize = 0;

bk1params.nCmdLen = 1;

bk1params.szBuffer = startAddress;

bk1params.nExpLen = (uint)startAddress.Length + 1;

// Creating object of 2nd BP params

BKPARM bk2params = new BKPARM();

bk2params.type = BKTYPE.BRKTYPE\_EXEC;

bk2params.count = 1;

bk2params.accSize = 0;

bk2params.nCmdLen = 1;

bk2params.szBuffer = endAddress;

bk2params.nExpLen = (uint)endAddress.Length + 1;

stopWatch.Reset();

unsafe

{

// Should to do something more flexible

operationStatus = UVSC\_Init(4823, 4832);

operationStatus = UVSC\_STATUS.UVSC\_STATUS\_FAILED;

{

short connectionHandler = 0, localport = 0;

// trying 3 times to open connection with uVision

while (operationStatus != UVSC\_STATUS.UVSC\_STATUS\_SUCCESS && trynumber != MAX\_OP\_TRIES)

{

operationStatus = UVSC\_OpenConnection(null, &connectionHandler, &localport, uvCmd, uvRunmode, newCallback, null, null, false, null);

trynumber++;

}

if (operationStatus != UVSC\_STATUS.UVSC\_STATUS\_SUCCESS)

{

// Connection can't be establish, return 0

return new TimeSpan(0);

}

else

{

connectionHndl = connectionHandler;

port = localport;

}

}

//Entering to Debug mode

operationStatus = UVSC\_DBG\_ENTER(connectionHndl);

if(operationStatus != UVSC\_STATUS.UVSC\_STATUS\_SUCCESS)

{

// Something went wrong, return 0

return new TimeSpan(0);

}

operationStatus = UVSC\_DBG\_CREATE\_BP(connectionHndl, bk1params, bkParamsSize, bkrsp, ref bkRspSize);

if (operationStatus != UVSC\_STATUS.UVSC\_STATUS\_SUCCESS)

{

// Can't create first BP, return 0

return new TimeSpan(0);

}

operationStatus = UVSC\_DBG\_CREATE\_BP(connectionHndl, bk2params, bkParamsSize, bkrsp, ref bkRspSize);

if (operationStatus != UVSC\_STATUS.UVSC\_STATUS\_SUCCESS)

{

// Can't create second BP, return 0

return new TimeSpan(0);

}

trynumber = 1;

operationStatus = UVSC\_STATUS.UVSC\_STATUS\_FAILED;

while (operationStatus != UVSC\_STATUS.UVSC\_STATUS\_SUCCESS && trynumber != MAX\_OP\_TRIES)

{

// There is need some delay to continue messaging with uVision

Thread.Sleep(1000);

operationStatus = UVSC\_DBG\_START\_EXECUTION(connectionHndl);

trynumber++;

}

if (operationStatus != UVSC\_STATUS.UVSC\_STATUS\_SUCCESS)

{

// Can't start execution, return 0

return new TimeSpan(0);

}

else

{

// Execution started, looking for first BP

runningFlag = UVSC\_DEBUG\_STATE.DEBUG\_STATE\_LOOKING\_FOR1\_BP;

}

//waiting for signal of stopping exectuion on 1st BP

while (runningFlag != UVSC\_DEBUG\_STATE.DEBUG\_STATE\_TARGET\_STARTING\_EXECUTING) ;

Thread.Sleep(1000);

operationStatus = UVSC\_DBG\_START\_EXECUTION(connectionHndl);

//waiting for signal of stopping exectuion on 2nd BP, which mean that time measurement must be finished

while (runningFlag != UVSC\_DEBUG\_STATE.DEBUG\_STATE\_FINISHED) ;

// Exit from Debug mode

UVSC\_DBG\_EXIT(connectionHndl);

Thread.Sleep(1000);

//Closing connection with uVision

UVSC\_CloseConnection(connectionHndl, false);

UVSC\_UnInit();

}

return stopWatch.Elapsed;

}

Д.3. Вихідний код функції обробника подій, які надходять від середовища Keil uVision

private static void keilCallback(IntPtr cb\_custom, UVSC\_CB\_TYPE type, ref UVSC\_CB\_DATA data)

{

if ((data.msg.data.cmdRsp.cmd == UV\_OPERATION.UV\_DBG\_STOP\_EXECUTION) && (data.msg.data.cmdRsp.StopR.nBpNum != -1) &&

(runningFlag == UVSC\_DEBUG\_STATE.DEBUG\_STATE\_LOOKING\_FOR1\_BP || runningFlag == UVSC\_DEBUG\_STATE.DEBUG\_STATE\_TARGET\_STARTING\_EXECUTING))

{

if(data.msg.data.cmdRsp.StopR.nBpNum == 0)

{

// FIRST BP

runningFlag = UVSC\_DEBUG\_STATE.DEBUG\_STATE\_TARGET\_STARTING\_EXECUTING;

}

else if (data.msg.data.cmdRsp.StopR.nBpNum == 1)

{

// Execution stopped at 2nd BP

stopWatch.Stop();

runningFlag = UVSC\_DEBUG\_STATE.DEBUG\_STATE\_FINISHED;

}

}

else if(data.msg.data.cmdRsp.cmd == UV\_OPERATION.UV\_DBG\_START\_EXECUTION && runningFlag == UVSC\_DEBUG\_STATE.DEBUG\_STATE\_TARGET\_STARTING\_EXECUTING)

{

// Starting execution after entering on 1st BP

stopWatch.Start();

}

return;

}