УДК 004

О.Безушко, магістр гр. КН-22М-2
Центральноукраїнський національний технічний університет

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ РЕІНЖИНІРИНГУ ТА РЕФАКТОРИНГУ ПРОГРАМНОГО КОДУ ДО ПЛАТФОРМИ .NET

У статті розроблено програмне забезпечення, яке призначено для системи реінжинірингу та рефакторингу програмного коду до платформи .NET. Метою розробки є дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET. Об'єктом дослідження є процес реінжинірингу та рефакторингу програмного коду до платформи .NET. Предметом дослідження є методи реінжинірингу та рефакторингу програмного коду до платформи .NET. Методи дослідження базуються на методах інженерії програмного забезпечення, методах математичної статистики, методах розробки програмного забезпечення. Результат роботи – програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET. В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Постановка проблеми. Щоб не відставати від конкуренції та зробити споживачів задоволеними вашим цифровим продуктом у 2021 році, потрібно підтримувати його програмний компонент на найвищому рівні стандартів якості.

Одним із способів досягти цього є частий перегляд і вдосконалення коду. У цьому відношенні існує два основних підходи до підтримки надійності програмного забезпечення – рефакторинг і реінжиніринг.

У роботі далі детальніше розглянемо ці дві практики, дослідимо їх відмінності та дізнаємось про основні переваги оновлення програмного забезпечення для бізнесу.

Аналіз останніх досліджень і публікацій. При аналізі останніх досліджень і публікацій [1-20] було виявлено певні прогалини у забезпеченні системи реінжинірингу та рефакторингу програмного коду до платформи .NET**.**

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем реінжинірингу та рефакторингу програмного коду до платформи .NET.

– Дослідження системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

– Програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Об'єктом дослідження є процес реінжинірингу та рефакторингу програмного коду до платформи .NET.

Предметом дослідження є методи реінжинірингу та рефакторингу програмного коду до платформи .NET.

Методи дослідження базуються на методах інженерії програмного забезпечення, методах математичної статистики, методах розробки програмного забезпечення.

Виклад основного матеріалу. Маючи справу із застарілим програмним забезпеченням, важливо розуміти, що можна зробити з програмним забезпеченням. Застаріле програмне забезпечення часто складається з програмного забезпечення, яке залишалося

працювати протягом тривалого часу без надто багатьох внутрішніх змін, стратегія «не виправляти те, що не зламано». Оскільки компілятори таких мов, як Fortran, є зворотно сумісними, часто можна скомпілювати та запустити ці старі програми. Але в якийсь момент виникає необхідність мати справу зі старим кодом. Отже, як цього досягти? *Чи підлягає переробці* чи *рефакторингу* код?

Реінжиніринг означає внесення фундаментальних змін до коду. Ось три основні методи реінжинірингу:

1. Портування – програми модифікуються для роботи на новій апаратній платформі.
2. Переклад – програми перекладаються із застарілої мови на сучасну.
3. Міграція – програми конвертуються зі старої мови на новіший діалект.

По суті, це нічим не відрізняється від роботи, яку проводили б у старій будівлі. Її можна повністю перенести на нове місце, повністю перебудувати або зробити новою, включивши лише фасад оригінальної будівлі.

Рефакторинг, з іншого боку, залишає речі недоторканими. Рефакторинг передбачає зміну частини програмного забезпечення таким чином, що зовнішня поведінка коду залишається незмінною, але його внутрішня структура та архітектура покращуються. Це схоже на модернізацію сантехніки та електрики старої будівлі. Він все ще функціонує і виглядає так само, але інфраструктура була покращена. Рефакторинг бере під контроль код, що занепадає, покращуючи читабельність і зручність обслуговування існуючого коду. Рефакторинг виконується, щоб виправити короткі шляхи, усунути дублювання та мертвий код, а також зробити дизайн і логіку зрозумілими. Щоб краще і зрозуміліше використовувати мову програмування. Це не обов'язково означає, що код перенесено на новий діалект мови. Рефакторинг часто є частиною життєвого циклу програмного забезпечення і може не бути націленим конкретно на застарілий код.

Вартість реінжинірингу програмного забезпечення

Можливо, це не має сенсу, коли ви чуєте, що переписування вже написаного коду є економічно ефективним, але вислухайте нас. Удосконалення та вдосконалення існуючого коду зараз заощадить ваші гроші. Ось деякі основні математики, зроблені Джоном Вірголіно, щоб довести це. Причина, чому в більшості випадків це правда, проста: кожна система програмного забезпечення (за винятком, можливо, справді маленьких і базових) будується ітераційно. Ви починаєте з основних і базових функцій, таких як керування користувачами та авторизація, перевірка будь-якого введення користувача, система виставлення рахунків тощо. Після цього ви додаєте функції для бізнесу, такі як багаторівневий каталог, система пропозицій, розширена аналітична система та будь-які інші корисні функції. Таким чином, покращуючи код у ядрі, ви покращуєте наявну та майбутню функціональність. Крім того, ви можете змінити ядро, щоб воно було готове до деяких майбутніх функцій, про які ви не думали півроку тому. Коротше кажучи, легше помістити ще одну коробку в сховище, коли вона акуратна й охайна. Крім того, залежно від системи, витрати на рефакторинг можуть навіть окупитися відразу після його впровадження. Ось чому.

Покращена продуктивність

Досить часто перероблений код працює швидше. Це тому, що розробники вже знають, як поточна система використовує ресурси. Вони вже знають, які запити SQL завжди викликаються разом і можуть комбінуватися. Вони бачили, як система працює у виробництві або в якомусь постановочному середовищі, де навантаження наближене до реальних умов. А хороша продуктивність завжди на користь. Це означає швидшу реакцію системи – те, що ви, ваші клієнти та навіть розробники оціните під час створення нових чудових функцій.

Є багато прикладів, коли рефакторинг програмного забезпечення та реінжиніринг покращили продуктивність системи, і ось три, щоб проілюструвати це.

Знижений ризик

У розробці нового програмного забезпечення завжди є елемент ризику. Це означає, що можуть виникнути проблеми з розробкою, кадровим забезпеченням або специфікацією.

Рефакторинг передбачає поступовий підхід до вдосконалення системи, а не радикальну заміну системи. Під час рефакторингу зменшується ймовірність втрати критичних бізнес-знань або створення системи, яка не відповідає потребам користувачів. Оскільки це можна здійснювати поетапно, ваш бізнес завжди має робочу систему, а кінцевим користувачам надається можливість поступово та легко адаптуватися до оновленої частини. Крім того, поетапна модернізація системи допомагає новим співробітникам дізнатися про потреби вашого бізнесу, недокументовані функції та особливості.

Хоча переробка всього з нуля може вивести ваш бізнес на новий рівень, ризики та витрати, ймовірно, різко зростуть. Отже, рефакторинг може бути не найшвидшим способом розвитку вашого бізнесу, але, мабуть, одним із найбезпечніших. Поступова реінжиніринг програмного забезпечення контролює рівень ризику, каже Майкл Р. Олсем у своїй роботі «Поступовий підхід до реінжинірингу програмних систем». Ключове слово тут «поступове». Без цього реінжиніринг міг би закінчитися двома різними, несумісними програмними системами. Тут вам допоможуть принципи безперервної інтеграції та безперервного розгортання.

Завжди готовий до вдосконалень

Як ми вже зазначали, більшість розробок програмного забезпечення передбачає вдосконалення існуючої системи, а не створення нової. Код має бути структурований таким чином, щоб вітати нові функції. Користувачі звикли до оновлення програмного забезпечення кожні шість місяців або навіть рідше, ринки постійно змінюються, і додавання нових функцій у ваше програмне забезпечення означає бути в курсі.

Макс Канат-Александер, автор «Простоти коду», припускає у своїй статті , що знадобиться менше часу, щоб навести порядок у системі, перш ніж почати додавати нові функції. Щоб довести це, він згадує, як його команда завершувала проекти швидше, ніж команди, які працювали над новими кодовими базами з кращими інструментами, коли це робили.

«Добре, це звучить переконливо», – скажете ви, – але що, якщо я не маю чітких термінів для нових функцій? Що, якщо я пропущу рефакторинг і зосереджуся лише на розробці нового?» Це слизький шлях у розробці програмного забезпечення. Дозвольте познайомити вас з містером Боргом. Технічний борг.

Дуже рідко архітектура програмної системи є ідеальною з самого початку. Без будь-якого рефакторингу рано чи пізно додати нові функції в систему буде практично неможливо.

Зворотне проектування

Зворотне проектування варто розглянути, якщо існуюча система, яка завоювала довіру та впевненість користувачів, потребує модернізації. У цьому випадку це дозволить створити серйозний програмний продукт в найкоротші терміни. Зворотне проектування також є хорошим тестом безпеки програмного забезпечення. Він широко використовується, щоб переконатися, що система не має будь-яких серйозних недоліків безпеки або вразливостей. Це допомагає зробити систему надійною та захищає її від хакерів і шпигунського програмного забезпечення.

У цьому світі, що постійно змінюється, програмне забезпечення має бути придатним для обслуговування, багаторазового використання та зручним для розширення, щоб задовольнити клієнтів і перемогти конкурентів. Програмне забезпечення стає дедалі складнішим, і ігнорування технічної заборгованості, пропускаючи рефакторинг, з часом повернеться й переслідуватиме ваш бізнес. І якщо ви тільки запускаєте наступну чудову програму, реінжиніринг – це практика, яку варто розглянути. Можливо, цей старий, подряпаний компакт-диск із купою коду на C і ботаном-розробником – це все, що зараз потрібно вашій компанії. Ви можете знайти приклади успішних тривалих програмних проектів, які не мають етапів «рефакторингу». Але це лише тому, що розробники постійно переробляють існуючий код. Вам не потрібно зупиняти процес розробки та планувати рефакторинг. Хороші технічні керівники та архітектори програмного забезпечення все одно

роблять це. Отже, якщо у вас є чудова команда розробників, ваш програмний продукт, ймовірно, уже неодноразово перероблявся.

Концепція зворотного проектування та рефакторингу в інженерії програмного забезпечення

Процес зворотного проектування починається з вилучення детальної інформації про проект, а з цього витягується абстракція проекту високого рівня. Детальна (низькорівнева) інформація про проект витягується з вихідного коду та існуючих проектних документів. Ця інформація включає структурні діаграми, описи даних і PDL для опису деталей обробки. Подібний підхід, але автоматизований, описано в інших місцях для відновлення документів Джексона та Уорньєра/Орра з коду. Представлення проекту високого рівня витягується з відновленого детального проекту та виражається за допомогою діаграм потоку даних і потоку керування. У цьому документі термін «відновлений дизайн» буде використовуватися для позначення вилученого дизайну.

Етапи процедури описані нижче:

1. Збирайте інформацію. Зберіть всю можливу інформацію про програму. Джерела інформації включають вихідний код, проектні документи та документацію для системних викликів і зовнішніх маршрутів. Необхідно також визначити персонал, який має досвід роботи з програмним забезпеченням.

2. Вивчіть інформацію. Перегляньте зібрану інформацію. Цей крок дозволяє людині, яка виконує відновлення, ознайомитися із системою та її компонентами. На цьому етапі можна сформулювати план диссемблера програми та запису відновленої інформації.

3. Витягніть структуру. Визначте структуру програми та використовуйте її для створення набору структурних діаграм. Кожен вузол на структурній діаграмі відповідає рядку, який викликається в програмі. Таким чином, діаграма записує ієрархію викликів програми. Для кожного ребра діаграми необхідно записати дані, передані до вузла та повернуті цим вузлом.

4. Функціональність запису. Для кожного вузла на структурній діаграмі запишіть обробку, виконану в програмному рядку, що відповідає цьому вузлу. PDL можна використовувати для вираження функціональності програмних рядків. Для системних і бібліотечних рядків функціональність може бути описана англійською або більш офіційною нотацією.

5. Запис потоку даних. Відновлену структуру програми та PDL можна проаналізувати для виявлення перетворень даних у програмному забезпеченні. Ці кроки перетворення показують обробку даних, виконану в програмі. Ця інформація використовується для розробки набору ієрархічних діаграм потоків даних, які моделюють програмне забезпечення.

6. Запис контрольного потоку. Визначте керуючу структуру високого рівня програми та запишіть її за допомогою діаграм потоку керування. Це стосується високорівневого контролю, який впливає на загальну роботу програмного забезпечення, а не низькорівневого контролю обробки.

7. Перегляньте відновлений дизайн. Перегляньте відновлений дизайн на узгодженість із доступною інформацією та правильність. Визначте будь-які відсутні елементи інформації та спробуйте знайти їх. Перегляньте дизайн, щоб переконатися, що він правильно представляє програму.

8. Створіть документацію. Завершальним етапом є створення проектної документації. Потрібно буде записати інформацію, що пояснює мету програми, огляд програми, історію тощо. Ці відомості, швидше за все, не міститимуться у вихідному коді та повинні бути відновлені з інших джерел.

Рефакторинг використовується для покращення якості коду, надійності та зручності обслуговування протягом життєвого циклу програмного забезпечення. Дизайн коду та якість коду покращуються завдяки рефакторингу. Рефакторинг також підвищує продуктивність розробника та збільшує кількість повторного використання коду. Наприклад, якщо два

методи використовують подібний фрагмент коду, загальний код можна рефакторингувати в інший метод, який потім можуть викликати два батьківські методи.

Рефакторинг програмного забезпечення = «Реструктуризація існуючого коду шляхом зміни його внутрішньої структури без зміни зовнішньої поведінки»

Етапи рефакторингу

Рефакторинг включає:

– Розробку = (додавання функцій, рефакторинг).

– Рефакторинг = (тестування, невеликі кроки).

– Маленькі кроки = один із рефакторингу типи «Зберіть речі разом, коли зміни є разом».

– Витягти методи.

– Перемістити методи.

– Перейменувати методи.

– Замінити Temp на Query.

– Замінити умови на поліморфізм.

– Замінити код типу на State/Strategy.

– Самоінкапсуляція польових додатків рефакторингу.

Ми використовуємо три приклади, щоб поясніть деякі базові рефакторингу

1. Метод вилучення:

– сигналізується коментарями.

– Один вхід, один вихід.

– збільшити рівень опосередкованості.

– зменшити довжину методу.

– збільшити ймовірність повторного використання.

2. Метод Move:

– Метод Place разом з об’єктом; об’єднайте речі разом, коли зміни є разом.

3. Замініть умови поліморфізмом:

– Перемикачі – це «жорсткий код», поліморфізм кращий для програм.

Ми використовуємо три приклади, щоб пояснити деякий базовий рефакторинг

1. Метод вилучення:

– сигналізований коментарями.

– Один вхід, один вихід.

– збільшити рівень опосередкованості.

– зменшити довжину методу.

– збільшити ймовірність повторного використання.

2. Перемістити метод:

– Розмістити метод разом з об'єктом; об’єднайте речі разом, коли зміни є разом.

3. Замініть умови на поліморфізм:

– Перемикачі є «жорстким кодом», поліморфізм кращий для розширюваності в об’єктно-орієнтованих програмах.

Розробка структурної схеми

Всім відомо що зараз є велика кількість вихідного коду на C/C++. Мови програмування застаріли та їх перестали використовувати та залишилась велика кількість вихідного коду на цих мовах. Цей код не має розвинутого інтерфейсу чи багатофункціональність але в цих кодах є реалізація математичних алгоритмів, високопродуктивних та новаторських підходів у реалізації математичних розв’язків рівнянь.

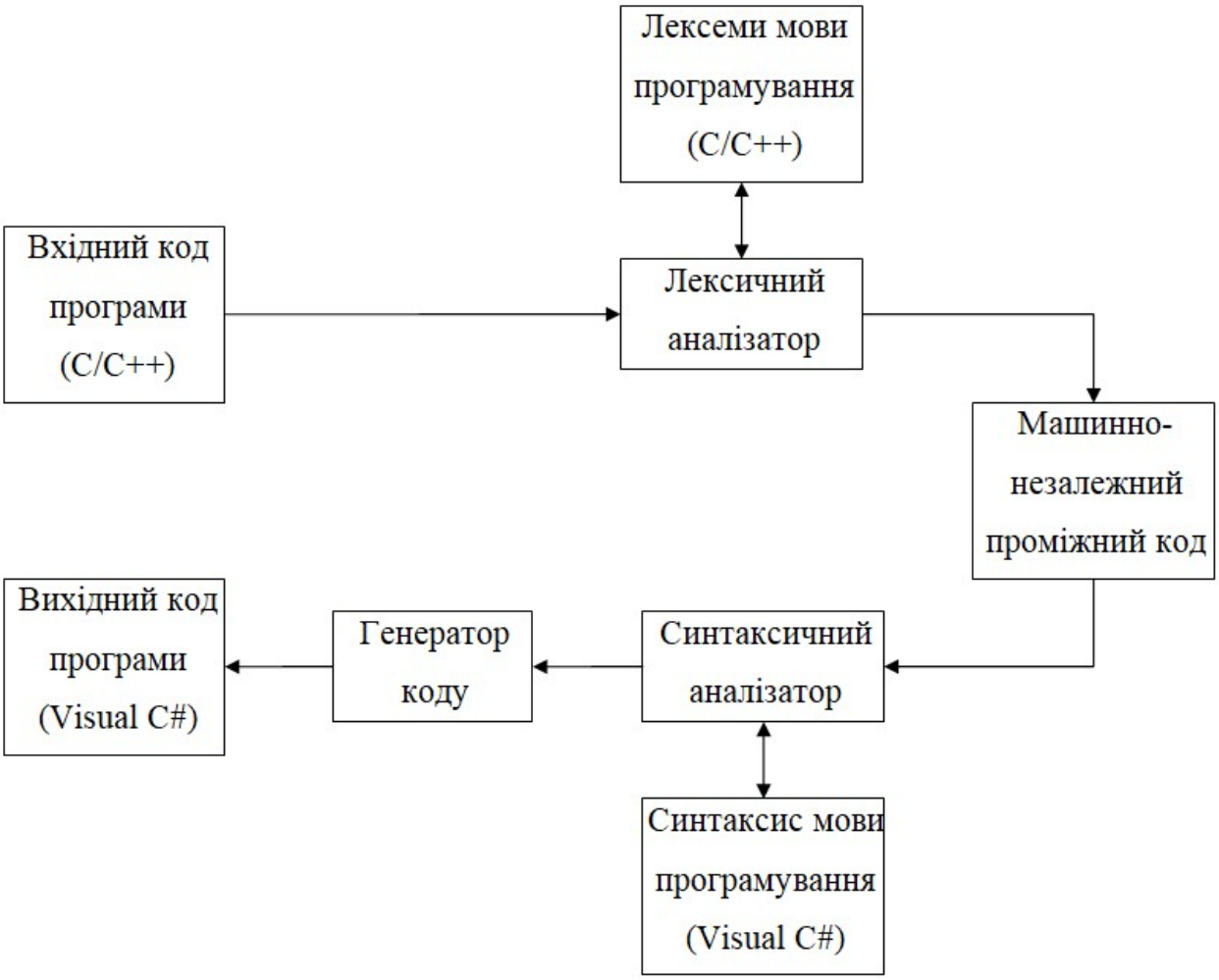

Рисунок 1 – Структурна схема системи

Незважаючи на те, що ці програми складалися досить давно, ідеї реалізації математичних теорій ніколи не старіють, особливо багато алгоритмічно-математичного вихідного коду використалося у вищих навчальних закладах.

Всі ці наробітки й рішення були заблоковані при старінні мови програмування.

Завдяки цьому розробка напрямку перекладу коду із застарілої мови програмування в більш новий завжди актуальна, код котрий надалі буде більш детально розглядатися та перероблятися під новий лад. Можливо й таке, що код після перетворення не буде у повній мірі робити, але сама структура алгоритму залишиться.

У розробленій, в результаті виконання магістерського проектування, програмі використається переклад частин вихідного коду з мови програмування C/C++ у Visual C#.

На рисунку 1 зображена структурна схема системи де розглянута будова транслятора. При розробці схеми транслятора був проведений аналіз з теорії трансляторів [1-10].

З рисунку добре видно що транслятор розбито на декілька блоків а саме – блок лексичного аналізатора який взаємодіє з набором лексем мови програмування C/C++, проміжного машино-незалежного коду, синтаксичного аналізатора з синтаксисом мови Visual C# та генератора коду. Детальний розгляд роботи цих компонентів розглянуто на функціональній схемі. На вхід транслятора поступає код C/C++, перед тим як запустити його у лексичний аналізатор (почати його оброблювати) проходить перевірка коду, що це дійсно є код C/C++. Ця дія виконується простою підстановкою шаблонного коду початку програми C/C++.

Висновки. У статті наведені теоретичне узагальнення й рішення наукового завдання дослідження методів реінжинірингу та рефакторингу програмного коду до платформи .NET.

Рішення даного завдання полягало у вирішенні наступних задач: Був проведений огляд існуючих систем реінжинірингу та рефакторингу програмного коду до платформи .NET; Досліджена система реінжинірингу та рефакторингу програмного коду до платформи .NET; На основі отриманих результатів досліджень створена програмна реалізація системи реінжинірингу та рефакторингу програмного коду до платформи .NET. Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання реінжинірингу та рефакторингу програмного коду до платформи .NET. Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Список літератури

1. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.
2. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.
3. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.
4. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143
5. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». Communications in Computer and Information Science, 2021, vol 1486. Springer, Cham. pp 169-184.
6. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.
7. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.
8. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.
9. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». International Journal of Computer Network and Information Security (IJCNIS). Vol. 12, No. 3, 2020. PP.33-43.
10. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.
11. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.
12. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.
13. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.
14. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.
15. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P. 707-712.
16. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.701-706.
17. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced

Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.
18. Вінтенко Б.Ю., Смірнов О.А., Коваленко А.С., Смірнов С.А., Буравченко К.О. «Дослідження вимог міжнародних стандартів IEC60880 та IEC62138 з розробки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». Системи управління, навігації та зв'язку, 2023, вип. 3(73), С. 155-166.
19. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». Кібербезпека: освіта, наука, техніка. 2024. №3(23), С. 111-131.
20. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». Системи управління, навігації та зв'язку, 2023, вип. 2(72), С. 170-178.
21. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.