Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи безпечної розробки при використанні технології Agile”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-24М
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Розиєв А.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Кислун О.А.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

АНОТАЦІЯ

Розиєв А. Дослідження та програмна реалізація системи безпечної розробки при використанні технології Agile. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи безпечної розробки при використанні технології Agile.

Метою розробки є дослідження та програмна реалізація системи безпечної розробки при використанні технології Agile.

Об'єктом дослідження є процес безпечної розробки при використанні технології Agile.

Предметом дослідження є методи безпечної розробки при використанні технології Agile.

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

Результат роботи – програмна реалізація системи безпечної розробки при використанні технології Agile.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, безпечна розробка, Agile

ABSTRACT

Rozyiev A. Research and software implementation of a secure development system using Agile technology. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software has been developed, which is intended for a secure development system using Agile technology.

The purpose of the development is the research and software implementation of a secure development system using Agile technology.

The object of the research is the process of secure development using Agile technology.

The subject of the research is the methods of secure development using Agile technology.

The research methods are based on the methods of software development theory, methods of mathematical statistics, and methods of software development.

The result of the work is the software implementation of a secure development system using Agile technology.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs with Windows 10/11.

The program is developed in the Python environment.

Keywords: computer engineering, secure development, Agile

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ	–	програмне забезпечення
ПП	–	програмній продукт

ВСТУП

Актуальність теми. Популярність Web-застосунків як основного інструмента онлайн-бізнесу росте рік у рік, одночасно із цім усе більше залучаючи зловмисників. Тим часом, відповідно до досліджень Contrast Security, порядку 80% Web-застосунків у середньому містять до 45 уразливостей, з яких принаймні одна має високий рівень критичності. Така ситуація свідчить про недостатню увагу до питань безпеки в процесі розробки. Із впровадженням методологій прискореної розробки Agile положення ще більше збільшується.

Швидкість розробки збільшується рік у рік. Раніше новий реліз з'являвся один раз у квартал або один раз у рік, зараз же, в умовах зростаючої конкуренції, всі замовники прагнуть якнайшвидше реалізовувати новий функціонал. У зв'язку із цім доводиться скорочувати цикли розробки: релізи випускаються кілька разів у тиждень або навіть у день, так що стандартний підхід, коли про безпеку замислювалися лише в самому кінці, при такій моделі працює дуже погано.

Насамперед якщо не власники бізнесу, то власники застосунків повинні приділяти пріоритетну увагу безпеки й усвідомлювати, як наслідку зломів, компрометацій і витоку даних позначаться на прибутку компанії. Крім того, необхідно переглянути роль фахівця з ІБ: з охоронця/наглядача, що заважає всім процесам і вставляє ціпка в колеса, як його часто сприймають, він повинен перетворитися в партнера й ІБ-наставника.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи безпечної розробки при використанні технології Agile.

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

– Огляд існуючих систем безпечної розробки при використанні технології Agile.

– Дослідження системи безпечної розробки при використанні технології Agile.

– Програмна реалізація системи безпечної розробки при використанні технології Agile.

Об'єктом дослідження є процес безпечної розробки при використанні технології Agile.

Предметом дослідження є методи безпечної розробки при використанні технології Agile.

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

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод безпечної розробки при використанні технології Agile.

– Розроблено вітчизняний продукт безпечної розробки при використанні технології Agile, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі безпечної розробки при використанні технології Agile.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2025 р.), основні положення випускної

кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи безпечної розробки при використанні технології Agile, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

По статистиці, на один фахівця з інформаційної безпеки доводиться в середньому 100-200 розроблювачів. У таких умовах забезпечити належний контроль украй складно, а виходить, необхідно ділитися компетенціями й робити їхньою частиною загального процесу. Тобто безпекою повинні займатися не тільки фахівці з ІБ, але й розроблювачі, тестувальники, менеджери продукту. Правильна організація цих процесів і залучення в них всіх команд, впровадження й насадження культури інформаційної безпеки і є нове завдання ІБ-фахівців.

З одного боку, це означає трансформацію ролі й значимості ІБ у компанії, з іншого боку – зміну використовуваних інструментів. Сучасні додатки збираються з модулів і компонентів, часто модулів на базі відкритого коду, а самі готові додатки розподілені між різними мікросервісами. У такому середовищі застосування традиційних засобів аналізу захищеності не завжди ефективно, і не всі статичні аналізатори вихідного коду можуть робити швидкі інкрементальні сканування коду при коротких циклах розробки. Тому для реалізації безпеки на будь-якому етапі розробки потрібні нові інструменти, доступні не тільки ІБ-фахівцям.

Обґрунтуванням заміни інструментів може служити той факт, що на усунення критичної уразливості наприкінці циклу розробки звичайно витрачається величезна кількість ресурсів. Іншими словами, чим пізніше в процес створення застосунку впроваджується безпека, тим дорожче воно обходиться. Ми вже підійшли до того моменту, коли кількість критичних для бізнесу застосунків росте з у роки в рік експоненціально, посилюється й конкуренція, тому будь-яка зупинка діяльності, викликана інцидентом безпеки, миттєво позначається на конкурентоспроможності компанії і її бізнесі в цілому.

Приступившись уже сьогодні до перегляду ролі й функцій інформаційної безпеки в процесі розробки застосунків: по-перше, зробивши ІБ загальною відповідальністю різних команд, по-друге, впровадивши методології AgileSec у процес розробки, – компанія одержить конкурентну перевагу в самий найближчий час.

По суті SecAgile і AgileSec – це один підхід, застосовуваний з однієї й тією же метою, але з різними акцентами на безпеці. Інакше кажучи, хтось ставить ІБ на перше місце, хтось – на останнє. Звідси й виникли терміни SecAgile і AgileSec.

Щоб розібратися в самому підході, необхідно мати подання про те, що таке Agile. Раніше в розробці виділялися різні зони відповідальності – кожний відповідав винятково за свою ділянку. Розроблювачі не мали подання про те, де в реальній інфраструктурі буде використовуватися застосунок, як його стануть масштабувати й т.д. А системні адміністратори й служба експлуатації займалися питаннями інфраструктури й були схильні покладати відповідальність на програмістів, якщо ОС і сервери функціонували без збоїв, але при цьому фіксувалися якісь неполадки в роботі застосунку. Таке перекладання відповідальності гальмувало процес випуску релізів, знижуючи тим самим загальну ефективність розробки.

Саме у зв'язку з реальною потребою в переломі ситуації й сформувалася парадигма об'єднання практик розробки, адміністрування й тестування в рамках загального процесу створення застосунку, іменований Agile. Важливо відзначити, що зараз усе ще не існує єдиного стандарту Agile – це скоріше якийсь загальний підхід, коли відповідальність за застосунок розділена між усіма, хто бере участь у його створенні й обслуговуванні.

Точно так само, як і в ситуації з виникненням природної необхідності в створенні Agile, з'явилася потреба у вбудовуванні безпеки в цей підхід. Адже немає такої точки на часовій шкалі розробки застосунку, коли наступає момент для забезпечення безпеки, – раніше захисту не було, а тепер її настав час впровадити. Так не буває. Цім потрібно займатися увесь час, на кожному етапі

життєвого циклу застосунку. У результаті й виникла ідея об'єднання Agile і ІБ. Наприклад, на етапі тестування якості коду безпека повинна бути інтегрована в цей процес. Тести на безпеку варто проводити паралельно з тестами на функціональність, а не по завершенні останніх. Безпека повинна бути частиною міри якості програмного продукту. Недостатньо просто враховувати кількість помилок і виявлених проблем – при виявленні уразливостей на етапі тестування, продукт не можна переводити на наступну стадію. В ідеалі поява проломів у коді потрібно мінімізувати ще до етапу тестування. Для цього фахівцям з ІБ необхідно взаємодіяти з розроблювачами, займатися їхнім навчанням.

Зовсім ясно, що неможливо зробити кожного розроблювача фахівцем з ІБ, оскільки це різні області діяльності й вимагають різного рівня підготовки. Однак, відповідно до концепції Agile, адміністраторам необхідно мати подання про особливості функціонування програмного продукту, а розроблювачам – про «поводження» їхнього коду в реальному середовищі. Точно так само відповідно до AgileSec фахівці з ІБ повинні впроваджувати базові концепції інформаційної безпеки на рівнях розробки й адміністрування.

Таким чином, у рамках AgileSec роль фахівців з безпеки складається в поліпшенні процесу розробки сучасних застосунків з погляду їхньої захищеності й, природно, у захисті того середовища, у якій цей застосунок буде працювати. У результаті відповідальність за безпеку розподіляється, хоча й нерівномірно, між всіма учасниками процесу, а не покладає винятково на фахівця з ІБ.

Розроблювачі повинні увесь час пам'ятати про завдання забезпечення ІБ і додержуватися принципів безпечної розробки застосунків хоча б у тій мері, щоб не допускати базових помилок. А керівники повинні ввести відповідні KPI. Наприклад, щотижня випускається черговий реліз. Метрикою якості коду з погляду безпеки може служити кількість повторюваних уразливостей у коді, тобто тих, що раніше вже були виявлені й некоректно усунуті. Або кількість нових уразливостей і т.д.

Тоді служба ІБ одержить важелі впливу на розроблювачів і можливість мотивувати їх на виконання встановлених вимог.

Інше завдання ІБ – захистити середовище, у якій застосунок працює: контейнери, системи розподілених обчислень, віртуальні машини, сервери. Адже чим складніший застосунок, тим більше точок відмови – місць можливого проникнення зловмисників.

1.2 Область застосування

Процес розробки сповільнює саме традиційний підхід ІБ – наприклад, застосування сканерів для аналізу коду на завершальному етапі створення застосунку.

Звичайно ж, впровадження методології AgileSec, її притирання й налагодження вимагають певних зусиль і чреваті затримками спочатку: необхідно знайти й впровадити потрібні інструменти, навчити людей. Однак, коли процес налагоджений, розробка буде йти з колишньою швидкістю й при цьому буде безпечною.

Якщо ж намагатися робити по старинці (швидка розробка, а потім перевірка на безпеку), то, крім зі строків релізів продукту, це приведе до постійного торгу щодо вибору змін, які необхідно внести в код для його захисту.

Як показала практика, компанії готові впроваджувати процеси безпечної розробки, інвестувати в навчання фахівців і придбання необхідних інструментів безпеки для підтримки прискорених циклів розробки. На цей момент уже зложилося розуміння того, що створення споконвічно безпечного продукту дозволяє, особливо в таких критичних сферах, як фінанси, обігнати конкурентів і при цьому уникнути ситуації, коли один інцидент ставить під погрозу весь бізнес.

Додаткові засоби захисту, звичайно ж, потрібні як засіб лікування симптомів. Однак зовнішній захист найменш надійний, адже жоден постачальник відповідних рішень не дає ніякої гарантії. Як жоден розроблювач антивірусів не

несе відповідальності у випадку вірусної епідемії, так і жоден виробник засобів захисту Web- і мобільних застосунків не заявляє про захист на 100%. Це в принципі неможливо.

Накладені засоби безпеки скоріше служать для виявлення атак на ранньому етапі, щоб заблокувати доступ зловмисникам до внутрішніх систем, для запобігання їхнього подальшого проникнення в них. У випадку Web-застосунків вони дозволяють виявити й зупинити успішну атаку, що стала можлива через наявність уразливостей, не замічених розроблювачами. Але всі подібні засоби, наприклад Web Application Firewall, можна потенційно обійти, адже атака може бути видозмінена таким чином, що не буде замічена системою захисту, або з'явиться абсолютно новий вид атаки, раніше невідомий.

Використання зовнішніх засобів безпеки – це якась додаткова, компенсаційна міра. Основна – виявлення уразливостей у коді застосунку.

Адже успішне відбиття атаки – це не підсумок боротьби зі зловмисником, це всього лише перепочинок, яку потрібно використовувати для пошуку уразливостей у коді і їхнього усунення. І це перше, що необхідно зробити, тому що засіб безпеки може сьогодні працювати, а завтра виявиться неефективним, наприклад, через помилку в конфігурації, запуску віртуальної машини в іншому ЦОДі й т.п.

Потрібно лікувати причину хвороби, а не її симптоми, тому пріоритет необхідно віддавати вдосконалюванню самого продукту.

Застосування алгоритмів машинного навчання в автоматизованих системах захисту, звичайно, перспективно, до того ж кількість атак і інцидентів постійно зростає й проаналізувати їх всі вручну, щоб обновити сигнатури, уже неможливо. Однак поки немає ніяких гарантій точності роботи й повноти цих алгоритмів.

Так, дійсно, на ринку Web Application Firewall і систем захисту Web-застосунків замість сигнатурного методу (або на додаток до нього) набирає популярність застосування евристичних методів, за допомогою яких система

вчиться розуміти, що є нормальним поводженням, а що – аномальним. Однак після впровадження таких систем виникає чимало проблем: їхнє навчання займає досить багато часу, а після відновлення програмного продукту алгоритми доводиться перебудовувати, оскільки зміна логіки роботи користувача сприймається як аномальне поводження. Це приводить до помилкових спрацьовувань, вимагає додаткових зусиль по адмініструванню такої системи й т.д.

Неможливо повністю виключити роботу аналітиків, замінивши їхніми алгоритмами. Найбільш перспективна, на мій погляд, комбінація ручних методів і машинного навчання, коли фахівці займаються аналізом атак, що відбулися, потенційних погроз і їхніх джерел. Та й у клієнтів найбільшу довіру викликають продукти тих компаній, у яких, крім алгоритмів для аналізу погроз, є великий штат кваліфікованих дослідників.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи безпечної розробки при використанні технології Agile, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

SDL (Security Development Lifecycle)

Створення безпечних застосунків – це одна із самих актуальних тем, що хвилюють розроблювачів. Тому розділ присвячено анонсованим Microsoft нововведенням в області процесу забезпечення безпеки всього циклу розробки – SDL (Security Development Lifecycle), включаючи бета-версію інструмента для моделювання потенційних уразливостей у створюваному програмному забезпеченні – SDL Threat Modelling Tool. Нагадаємо, що сьогодні процес SDL, що представляє собою строгу методику розробки безпечного програмного забезпечення, застосовується в самій корпорації Microsoft при створенні всіх її продуктів на кожному етапі процесу розробки програмного забезпечення.

У цій області стануть доступні як засоби безпеки, убудовані в продукти довільного призначення, так і спеціалізовані засоби забезпечення безпеки. До першої категорії ставляться засоби автентифікації, авторизації, технології шифрування, а до другого – інструменти забезпечення захисту даних і протидії погрозам.

Давайте згадаємо, із чого Microsoft почали. Нам відомо багато погроз (віруси, інше шкідливе ПЗ), на які повинне адекватно реагувати програмне забезпечення. Це означає, що потрібні продукти, здатні захистити від цих погроз інше ПЗ й дані (наприклад, Outlook, Word, архіви електронної пошти). Microsoft допомагаємо виробникам ПЗ вносити в нього необхідні зміни, а крім того, в останні роки Microsoft самі внесли багато змін у продукти й операційну систему.

Останнім часом спостерігається помітна криміналізація області, пов'язаної з порушенням безпеки. Якщо раніше автори шкідливого ПЗ намагалися просто нашкодити користувачам без якої б те не було матеріальної вигоди, то сьогодні вони роблять це з корисливих міркувань, що створює додаткову небезпеку. Тому сьогодні наше завдання полягає у впровадженні життєвого циклу безпечної розробки застосунків (Secure Development Lifecycle, SDL) при створенні ПЗ самої Microsoft і ПЗ для платформ Microsoft. Microsoft допомагаємо незалежним розроблювачам, які створюють застосунки на основі наших платформ. Microsoft розробили модель оптимізації життєвого циклу безпечної розробки застосунків SDL Optimization Model – це і є спосіб, за допомогою якого сьогодні потрібно здійснювати розробку застосунків.

Для часткового впровадження SDL для того, щоб ваші додатки могли протистояти погрозам, необхідно зробити ряд невеликих кроків – на цей рахунок Microsoft можемо дати рекомендації, наприклад із приводу керування списком користувачів і їхніх прав, у якості одного з перших кроків у цьому напрямку.

У найближчі два роки Microsoft, швидше за все, зіштовхнеться з набагато більшою, у порівнянні з попереднім періодом, кількістю атак, адже з кожним днем їхнє число зростає. Такий висновок випливає з аналітичних звітів, і це викликає серйозну заклопотаність. Особливо багато атак буде спрямовано проти операційних систем сімейства Windows. Це означає, що Microsoft повинні захистити дану платформу, а також сприяти захисту розроблених для неї застосунків.

Думаю, що потрібно захищати всі можливі мішені. SDL – це всеосяжний процес, що включає й моделювання погроз, і аналіз коду, і обробку виключень, адже його призначення – робити процедуру завдання збитків усе більше й більше скрутною для атакуючих, що повинне змусити їх відмовитися від атаки. Найважливіше – цей процес повинен бути непомітний для клієнта, оскільки він звичайно не цікавиться подібними проблемами й завданнями. Клієнтові потрібно мати застосунки, що працюють у звичному режимі, а Microsoft повинні допомагати йому виконувати роботу, створюючи безпечні додатки.

Які аргументи зможе висунути керівник проекту по розробці програмного забезпечення менеджерові вищої ланки або замовникові, доводячи, що в продукті варто реалізувати ті або інші механізми забезпечення безпеки? При розробці комерційного ПЗ всі хочуть заощадити час, гроші, ресурси, і багатьом представляється дуже зручним зробити це за рахунок безпеки.

Іноді таких менеджерів вдається переконати, пред'явивши статистичні дані про фінансовий збиток, викликаному атаками, які повідомляються аналітичними агентствами, урядовими організаціями, дослідницькими компаніями. Однак найчастіше, незважаючи на те що Microsoft наводить подібні дані як аргументи протягом довгого років, менеджерів ці проблеми не торкають, оскільки він не бачать за цифрами реальної людини, що підписує чеки, і їм здається, що всі проблеми від них дуже далекі.

Microsoft розуміє, що задоволеність клієнтів прямо не пов'язана з реалізацією засобів захисту й вони не люблять про це говорити. Проте можна довідатися, який статус даних клієнта, чи конфіденційні вони. А потім поцікавитися, чи не хоче клієнт захистити ці дані, які ризики їхньої крадіжки або компрометації, які при цьому будуть втрати, і підвести клієнта до думки, що його дані мають потребу в захисті. У такий спосіб ви ідентифікуєте ризики. Захист даних коштує грошей, але їхній витік обійдеться набагато дорожче.

Є також зміст обговорити з експертом в області інформаційної безпеки, скільки коштує реалізація різних її складових і які з них потрібні в цьому випадку, обміркувати, із чого почати, як побудувати й удосконалити надалі процес забезпечення безпеки. На цей рахунок Microsoft може дати рекомендації, що включають і навчання, і самоосвіта, і реалізацію.

Крім моделі SDL Optimization Model Microsoft недавно анонсували інструмент моделювання, що допоможе користувачам реалізувати SDL. Він дозволяє швидко оцінити необхідні кроки в області захисту розроблювального застосунку. Цей інструмент допоможе й розроблювачам, і менеджерам вищої ланки зрозуміти, яка структура витрат на безпеку, скільки часу це займе й на що конкретно повинні піти інвестиції в безпеку.

SDL Optimization Model саме і є найменш витратний спосіб реалізації захисту даних. До речі, і цю модель, і відповідний інструмент моделювання можна завантажити із сайту компанії Microsoft.

Спочатку Microsoft створює модель застосунку, описуючи його архітектуру (застосунку, сховища даних), щоб визначити можливі погрози (Рисунок 2.1). А потім просимо інструмент оцінити цю архітектуру в плані безпеки й показуємо клієнтові цю оцінку, що може бути дуже корисно в тому випадку, якщо клієнт не знає, як ідентифікувати можливі погрози. При цьому інструмент не забуде про такі погрози, як втрата КПК, або про зовнішні погрози, непідконтрольні компанії. Все це реалізовано в даному інструменті.

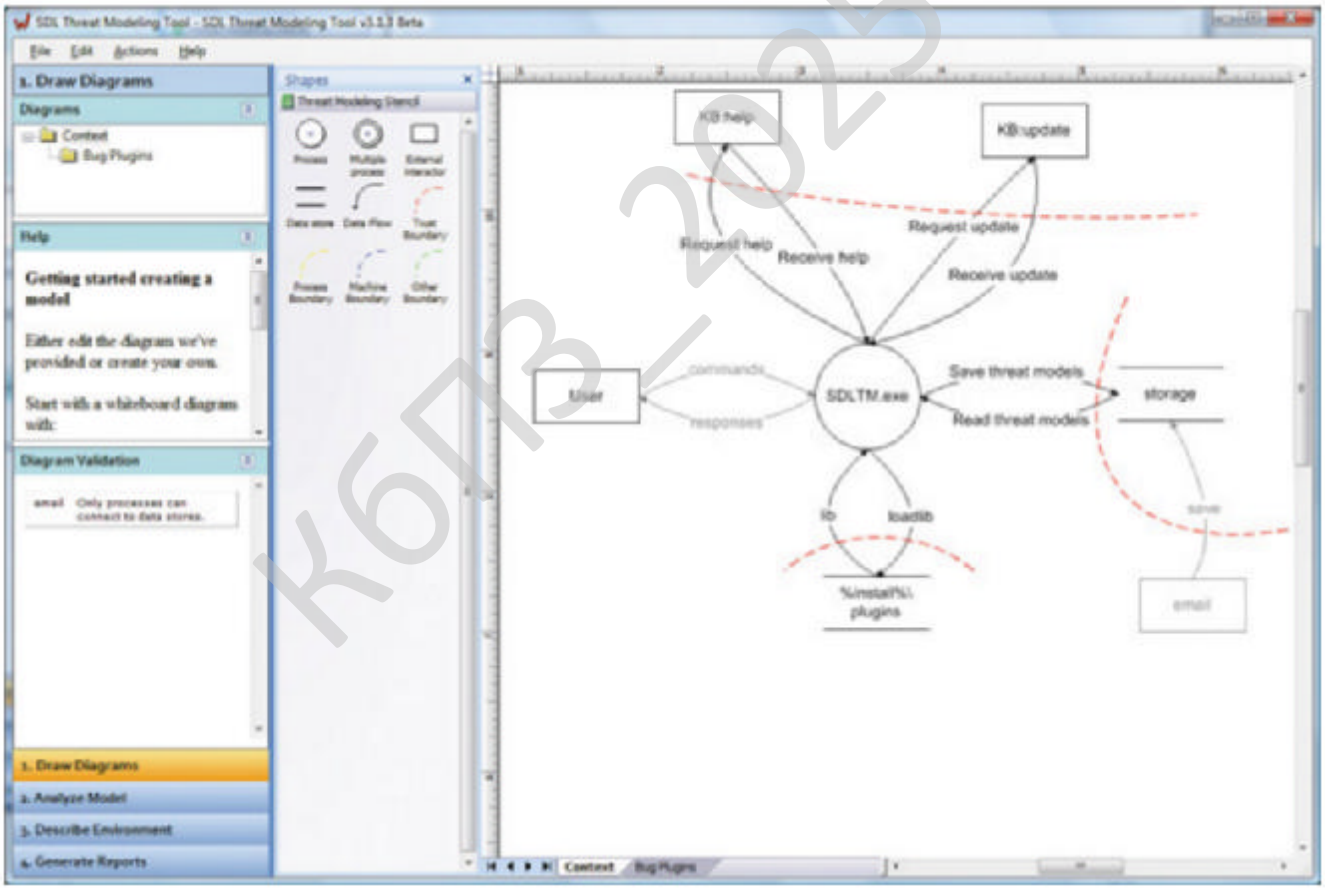

Рисунок 2.1 – Моделювання архітектури майбутнього застосунку в SDL Threat Modelling Tool

Іншими словами, Microsoft створює діаграму, на підставі якої автоматично генерується опис можливих дій і погроз. У комплекті поставки інструмента є файл довідкової системи з докладним описом того, що реально являють собою ці погрози, і з докладними нагадуваннями про необхідні міри (наприклад, що потрібно захищати відомості про кредитні карти покупців, скажемо, зашифрувавши їх). За допомогою цих відомостей можна переконувати менеджерів у наявності відповідних ризиків і включати в контракт реалізацію відповідного захисту, а також відповідні дані поміщати, приміром, у базу даних дефектів продукту поряд з результатами тестування.

Microsoft говорить про основну функціональність інструмента для аналізу погроз і способів захисту (Рисунок 2.2). Крім цього з його допомогою можна відслідковувати залежності між компонентами рішення, у тому числі з урахуванням засобів захисту даних інших виробників, генерувати звіти й про уведену інформацію, і про результати аналізу, і про засоби усунення проблем.

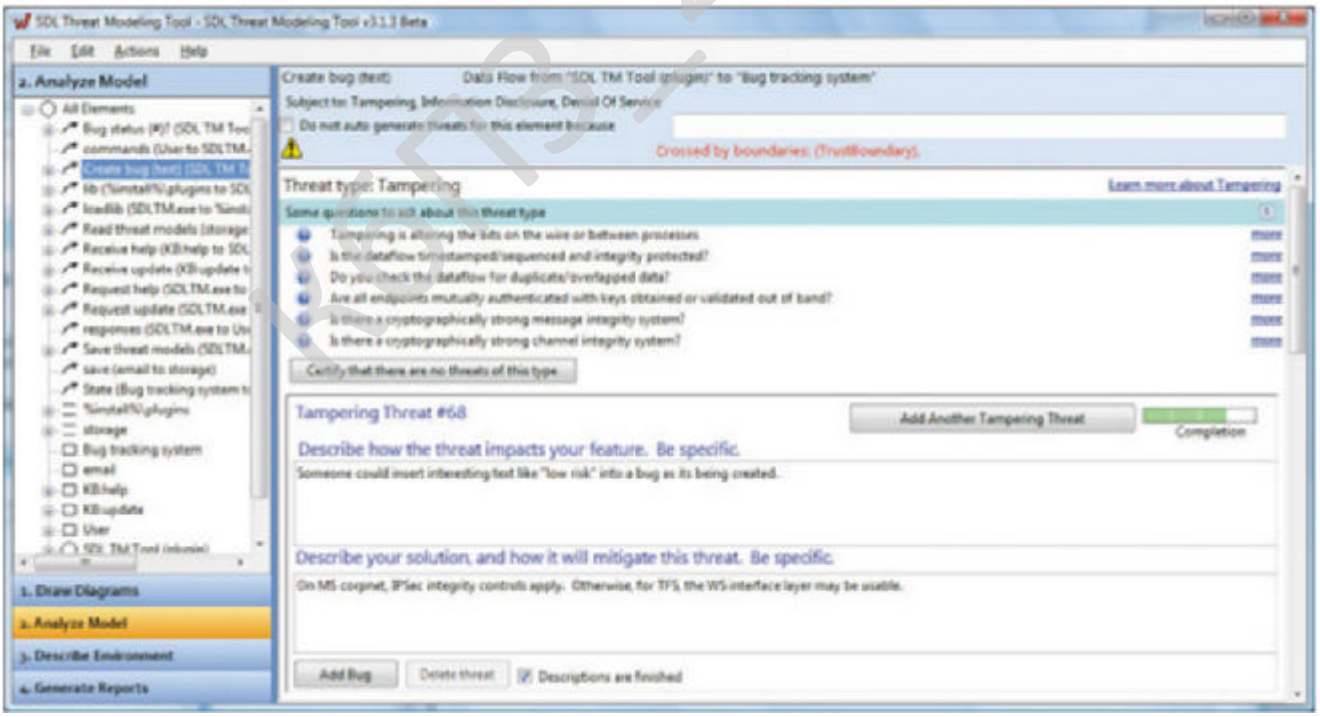

Рисунок 2.2 – Аналіз уразливостей майбутнього застосунку за допомогою SDL Threat Modelling Tool

SDL Optimization Model і інструмент моделювання дозволяють здійснювати оцінки необхідних мір безпеки за мінімальний час – 5-10 хв. За цей період ви одержите звіт про аналіз моделі з рекомендаціями мер безпеки, які треба прийняти при розробці застосунку (Рисунок 2.3). Якщо раніше для оцінки погроз потрібно було вивчити відповідну літературу, провести експерименти, що могло зажадати кілька тижнів роботи, то за допомогою нашого інструмента Microsoft можемо істотно скоротити цю частину циклу розробки. Крім того, для скорочення часу розробки Microsoft пропонуємо відповідні фрагменти коду в комплектах поставки наших продуктів, які ви можете модифікувати й використовувати у ваших рішеннях.

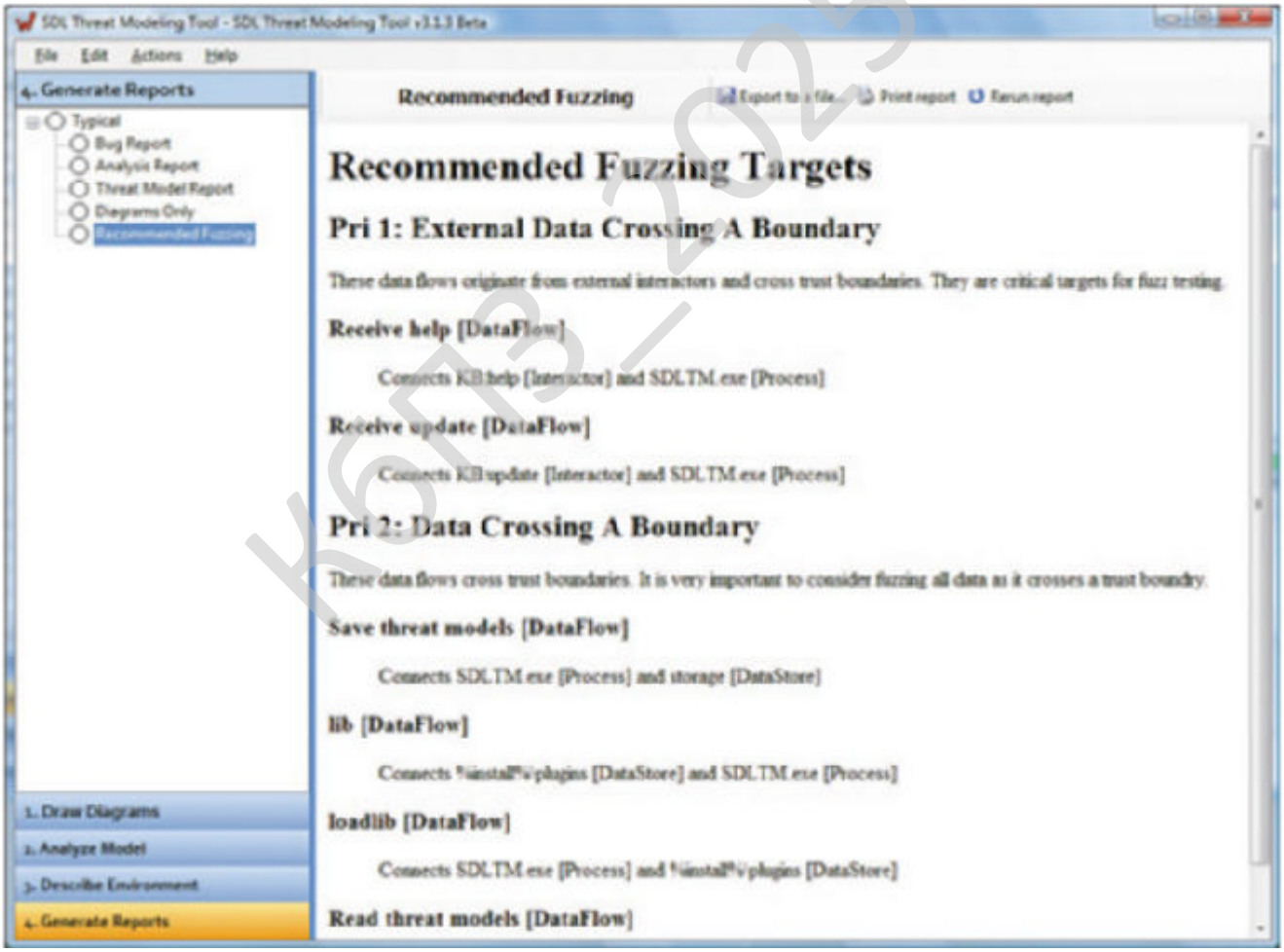

Рисунок 2.3 – Звіт з описом погроз і рекомендованих мір безпеки

Даний інструмент зможе показати можливі погрози, але в нас немає достовірної інформації про оточення моделюємого застосунку, щоб здійснити подібну оцінку ризиків для різних клієнтів. Тому Microsoft не враховуємо ймовірність настання ризику, а просто вказуємо, що він існує. Потім відповідно до SDL варто вирішувати, займаємося Microsoft чи проблемою ні – і це вже бізнес-рішення. Можливо, від якихось погроз варто захищатися у випадку, якщо продукт пишеться для зовнішнього клієнта й підданий інтернет-погрозам, але не коштує, якщо він функціонує усередині корпоративної мережі. Однак такі рішення повинні приймати менеджери й користувачі.

При розробці будь-якої програмної системи, будь це простий вебсайт, десктоп-застосунок або складний трирівневий комплекс, рано або пізно виникають питання безпеки. Не можна виключити, що та система, що ви розробляєте, буде якимось образом атакована. Причому, залежно від типу системи, її складності, застосовуваних технологічних рішень, вектори атак і їхнього наслідку можуть мати самий різний характер. Можливо, час від часу хтось у команді проводить аналіз безпеки розроблювальної системи, проводиться моделювання. Куди гірше якщо ці питання залишаються на потім. Результати можуть бути досить жалюгідними, якщо вашу систему зламують у режимі комерційної експлуатації й вам доводиться похапцем створювати виправлення для виявленого пролому. Очевидно, що питання, пов'язані з безпекою краще вирішувати, починаючи із самих ранніх етапів створення системи, таких як аналіз вимог і архітектурного моделювання. Але найкраще це робити на всьому життєвому циклі системи, інтегрувавши в процес розробки спеціальні кроки, призначені для рішення цих питань. Одним з таких процесів є Security Development Lifecycle – набір практик спрямованих на підвищення безпеки розроблювальних систем.

SDL це процес який дозволяє переконатися в необхідному рівні безпеки розроблювальної системи. SDL базується на основі практик спрямованих на навчання команди, підготовку звітності й безпосередні дії пов'язані з аналізом

безпеки розроблювальної системи й імплементацією механізмів спрямованих на поліпшення безпеки. Ці практики у вигляді конкретних кроків легко лягають на звичний спіральний цикл розробки програмного забезпечення.

Деякі із цих практик самі по собі можуть поліпшити безпеку розроблювальної системи. Але як показує досвід, їхнє застосування в рамках процесу розробки дозволяє значно поліпшити результат і знизити витрати.

Всі члени команди, перед тим як безпосередньо почати розробку, повинні пройти тренінг по безпеці й вивчити важливі моменти, пов'язані з поточними трендами в цій області. Базовий рівень цього тренінгу повинен містити в собі:

1. Безпечний дизайн:
– Зниження областей атаки.
– Глибокий захист.
– Принцип найменших привілеїв.
– Безпека за замовчуванням.
2. Моделювання погроз, включаючи наступні теми:
– Огляд моделей погроз.
– Вплив моделі погроз на дизайн.
– Обмеження для стилю кодування, що базуються на моделі погроз.
3. Безпечне кодування, включаючи наступні теми:
– Переповнення буфера (для застосунків на C і C++).
– Помилки цілочисельної арифметики (для застосунків на C і C++).
– Крос-сайтовий скриптинг (для Веб-застосунків).
– SQL-ін'єкції (для застосунків взаємодіючих із БД).
– Слабка криптографія.
4. Тестування безпеки, включаючи наступні теми:
– Розходження між тестуванням безпеки й функціональним тестуванням.
– Аудит ризиків.
– Методи тестування безпеки.

5. Забезпечення приватності, включаючи наступні теми:

– Типи приватної інформації.

– Приватність: кращі практики дизайну.

– Аудит ризиків.

– Приватність: кращі практики розробки.

– Приватність: кращі практики тестування.

Звичайно, при вивченні цих тим необхідно враховувати роль (тестер, розроблювач, менеджер, аналітик), проте, дуже важливо щоб всі члени команди пройшли цей тренінг.

Аналіз вимог: Що варто враховувати в контексті безпеки

Уже на рівні аналізу вимог до системи потрібно враховувати важливі аспекти, пов'язані з безпекою. При цьому важливо розділяти між собою «безпечні вимоги» і «вимоги безпеки». В SDL є деякий набір практик, і трохи таких з таких практик – «Аналіз безпеки й приватності вимог» може бути застосована на етапі аналізу вимог. Ця практика призначена для ідентифікації функціональних вимог, у яких є необхідність поглибленого вивчання питань пов'язаних з безпекою. Такий аудит може містити в собі наступну інформацію:

– Яка частина ПЗ вимагає аналізу погроз перед релізом?

– Яка частина ПЗ вимагає аналізу дизайну в контексті безпеки?

– Яка частина ПЗ вимагає додаткового аналізу погрози проникнення незалежною групою?

– Є чи додаткові питання безпеки й ризики, які можуть бути знижені?

– Границі нечіткого тестування в контексті безпеки.

– Який рівень погрози розголошення приватних даних?

Дизайн із урахуванням безпеки

При дизайні застосунків так само можна застосувати ряд практик, які допоможуть підвищити безпека застосунку. У першу чергу це зниження площі поверхні можливих атак (Attack Surface Reduction) і моделювання погроз. Незважаючи на близький взаємозв'язок цих двох понять, перший механізм має на

увазі активне зниження можливостей зловмисника на експлуатацію невідомих проломів у безпеці. Для зниження площі можливих атак можна застосовувати механізми пошарового захисту й принципи найменших привілеїв. Моделювання погроз у свою чергу дозволяє припустити, які компоненти системи можуть бути розглянуті як вектори атак. Зручним інструментом для моделювання погроз є Microsoft Thread Modeling Tool базований на класифікації STRIDE.

Реалізація з урахуванням безпеки

Етап реалізації, як правило, найбільш трудомістка частина проекту. Щоб полегшити це завдання використовується безліч засобів, технологій і готових компонентів. У контексті безпеки важливо щоб перелік цього інструментарію був заздалегідь зафіксований. Рекомендованим підходом є формування переліку дозволених при імплементації інструментів. Скрізь де тільки можна варто виключати або спеціально позначати використання небезпечних або функцій, що вийшли із уживання, і компонент. На додаток, важливе використання автоматизованих засобів, таких як статичний аналіз коду.

Фаза верифікації (тестування) з урахуванням безпеки

На цій фазі можливе використання таких практик як динамічний аналіз коду (під час виконання) який дозволяє швидко виявити аномальне поводження функцій, псування пам'яті, використання привілейованих функцій і інші критичні проблеми. Одним з інструментів який може придасться для рішення цього завдання є AppVerifier. На застосунок до стандартних функціональних тестів так само варто додати плаваюче тестування яке призначено для перевірки застосунку в режимі уведення невірних даних, неправильно сформованих параметрів і інших умов які можуть привести до аномального поводження але при цьому однаково система повинна бути в безпечному стані. Так само на цьому етапі важливим є перевірка змодельованих векторів атак на попередніх фазах для того щоб переконатися в коректності сформованих моделей.

Фаза Випуску з урахуванням безпеки

На цій фазі важливе створення плану по реакції інцидентів пов'язаних з безпекою, у яких буде задекларований порядок взаємодії й реакції на виявлені погрози або проникнення. Фінальні релізи (RTM, RTW) повинні відповідати всім заздалегідь обговореним на стадії дизайну умовам безпеки перед розгортанням. Якщо це не так, навіть при дотриманні всіх функціональних вимог, необхідний повтор стандартного циклу розробки для фіксації проблем пов'язаних з безпекою.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – це об’єктно-орієнтована мова програмування високого рівня загального призначення з відкритим кодом. Це визначення може бути важким для новачків, тому розглянемо кожну характеристику окремо, щоб зрозуміти, що вона означає:

– Відкритий вихідний код: це безкоштовно та доступно для подальших покращень, таких як додавання корисних функцій або виправлення помилок.

– Об’єктно-орієнтована: заснована не на функціях, але в об’єктах з певними атрибутами й методами.

– Високий рівень: зручний для людини, а не для комп’ютера.

– Загальне призначення: можна використовувати для створення будь-яких програм.

Ця мова використовується в будь-якому програмному забезпеченні, про яке ви тільки можете подумати. Ви можете використовувати його для створення веб-сайтів, штучного інтелекту, серверів, програмного забезпечення для бізнесу та багато іншого. Також застосовується в науці про дані, аналізі даних, машинному навчанні, інженерії даних, веб-розробці, розробці програмного забезпечення та інших галузях.

Переваги та недоліки Python

Переваги:

– Її легко читати, вчити та писати. Це мова програмування високого рівня з англійським синтаксисом. Це полегшує читання та розуміння коду. Її дійсно легко зрозуміти і вивчити, тому багато людей рекомендують Python новачкам. Вам потрібно менше рядків коду для виконання того ж завдання в порівнянні з іншими основними мовами, такими як C/C++ та Java.

– Підвищує продуктивність. Це дуже продуктивна мова. Завдяки її простоті розробники можуть зосередитися на розв'язанні проблеми. Їм не потрібно витрачати багато часу на розуміння синтаксису або поведінку мови програмування. Ви пишете менше коду та виконуєте більше завдань.

– Інтерпретована мова. Python мова, що інтерпретується, а це означає, що вона безпосередньо виконує код по рядку. Якщо сталася помилка, вона зупиняє подальше виконання та повідомляє про її виникнення. Вона показує лише одну помилку, навіть якщо у програмі їх кілька. Це спрощує налагодження.

– Динамічно типізована. Python не визначає тип змінної, доки ми не запустимо код. Вона автоматично надає тип даних, коли відбувається процес виконання. Фахівець може не турбуватися про оголошення змінних та типи даних.

– Безкоштовна та з відкритим вихідним кодом. Ця мова постачається під схваленою OSI ліцензією з відкритим вихідним кодом. Це робить його безкоштовним для використання та розповсюдження. Ви можете завантажити вихідний код, змінити його та навіть розповсюджувати свою версію. Це корисно для організацій, які хочуть використати свою версію для розробки.

– Підтримка великих бібліотек. Стандартна бібліотека Python є величезною, ви можете знайти майже всі функції, необхідні для вашого завдання. Таким чином ви не залежите від зовнішніх бібліотек.

– Портативність .У багатьох мовах, таких як C/C++, потрібно змінити свій код, щоб запустити програму на різних платформах. З Python все інакше. Ви тільки пишете один раз і запускаєте її будь-де.

Недоліки:

– Низька швидкість. Вище ми обговорювали, що це інтерпретована мова з динамічною типізацією. Порядкове виконання коду часто призводить до повільного виконання. Динамічна природа Python також є причиною її низької швидкості, оскільки їй доводиться виконувати додаткову роботу при виконанні коду. Тому вона не підходить для цілей, де швидкість важливий аспект проєкту.

– Неефективна для пам’яті. Ця мова програмування використовує великий обсяг пам’яті, це може бути недоліком при створенні програм, коли віддають перевагу оптимізації пам’яті.

– Слабка у мобільних обчисленнях. Python зазвичай використовується у серверному програмуванні. Ми не бачимо – її на стороні клієнта або в мобільних програмах з таких причин: вона не заощаджує пам’ять і має повільну обчислювальну потужність у порівнянні з іншими мовами.

– Доступ до бази даних. Програмувати на цій мові легко, але коли ми взаємодіємо з базою даних, її не вистачає. Рівень доступу до бази даних у Python примітивний та недостатньо розвинений у порівнянні з іншими популярними технологіями.

– Помилки виконання. Це мова з динамічною типізацією, тому тип даних змінної може змінюватись у будь-який час. Змінна, що містить ціле число, у майбутньому може містити рядок, що може призвести до помилок виконання.

Застосування Python:

– Для аналізу даних. Дані стали цінним активом у будь-якій сучасній галузі, і більшість компаній зацікавлені у збиранні, обробці та аналізі релевантних даних, щоб витягти з них цінну інформацію для бізнесу. І тут Python виходить за межі будь-якої конкуренції. Python особливо цінна тим, що крім великої стандартної бібліотеки надає величезний набір додаткових модулів,

розроблених спеціально для аналітичних цілей. Найвідоміші бібліотеки Python для аналізу даних – це pandas і NumPy . Ці інструменти дозволяють робити з вашими даними майже все, наприклад, очищати і аналізувати їх, вивчати статистику або візуалізувати приховані тенденції у ваших даних.

– Для візуалізації даних. Візуалізація даних – це окрема частина аналізу даних, яка допомагає нам подавати інформацію, необроблену чи очищену, у більш змістовній формі. Тут Python знову входить у гру, пропонуючи широкий спектр інструментів візуалізації даних. Найпопулярніші з них – matplotlib і заснований на ній seaborn. Використовуючи їх, ми можемо створювати буквально всі види візуалізації: від найпростіших до складніших.

– Для машинного навчання. Машинне навчання (ML) є основою більшості завдань науки даних. Він є областю штучного інтелекту, пов’язаною з використанням алгоритмів, що дозволяють машинам вивчати закономірності та тенденції на основі історичних даних, щоб робити прогнози на основі невідомих даних. – Використовуючи методи ML, ми можемо створювати моделі, які можуть точно передбачити швидкість відтоку клієнтів компанії, оцінити ризик виникнення у людини певного захворювання, визначити оптимальне розташування автомобілів таксі й т.д. За допомогою Python ми можемо побудувати модель ML, використовуючи лише три рядки коду.

– Для розробки програмного забезпечення. Крім свого багатостороннього застосування в галузях науки про дані, Python використовується на кожному етапі розробки програмного забезпечення, включаючи контроль складання, автоматичну безперервну компіляцію, прототипування, відстеження помилок, тестування та обслуговування програмного забезпечення. За допомогою цієї мови можемо створювати аудіо- або відеопрограми на основі методів штучного інтелекту, машинного навчання, API (інтерфейсів прикладного програмування), GUI (графічних інтерфейсів) або будь-якого іншого типу програмного забезпечення.

– Для веброзробки. У той час як для створення візуальної частини вебсайту ми переважно будемо використовувати такі мови, як HTML, CSS та JavaScript, для його невидимої частини ми часто вибираємо Python. Серед масштабних вебсайтів та програм, створених за допомогою цієї мови, варто згадати Google, Facebook, Instagram, YouTube, Dropbox та Reddit.

– Для автоматизації задач/скриптингу. Це відмінний інструмент для написання програм для автоматизації різних завдань, що повторюються. Цей процес називається скриптингом. Зокрема, можна робити скрипти для роботи з файлами та папками. Наприклад, можна створювати, перейменовувати, перетворювати, розділяти, об’єднувати або видаляти файли, перевіряти їх на наявність помилок. Ви також можете використовувати автоматизацію Python для пошуку та завантаження інформації з Інтернету, заповнення та надсилання онлайн-форм та надсилання регулярних повідомлень або електронних листів.

Яким фахівцям потрібно володіти Python:

– Фахівець з даних.

– Аналітик даних.

– Інженер даних.

– Інженер з машинного навчання.

– Журналіст даних.

– Архітектор даних.

– Повний стек веб-розробника.

– Backend-розробник.

– DevOps-інженер.

– Інженер-програміст.

Можемо зробити висновок, що Python ще довго буде популярною мовою, хоч і має низку недоліків. Цю мову використовують для створення вебсайтів, штучного інтелекту, серверів, програмного забезпечення для бізнесу, аналізу даних, машинного навчання, інженерії даних та для багатьох інших областей. Це перспективна і затребувана навичка, яка необхідна у всіх галузях.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи безпечної розробки при використанні технології Agile.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Agile – це гнучка методологія розробки (англ. Agile software development). Являє собою революційну концепцію, у рамках якої виконується розробка програмного забезпечення. У рамках даної концепції існує кілька методик.

Всі ці методики ставлять метою мінімізацію ризиків, досягається ця мета розробкою (проектуванням) короткими ітераціями.

Основні цінності Agile утримуються в “Маніфесті Agile”:

– Люди і їхня взаємодія важливіше, ніж процеси й засоби.

– Працююча система важливіше, ніж вичерпна документація.

– Співробітництво із замовником важливіше, ніж обговорення умов контракту.

– Реагування на зміни важливіше, ніж проходження плану.

Scrum – це конкретна технологія (методи ведення проекту й ролі учасників процесу), що реалізує принципи Agile.

Дана технологія дозволяє в жорстко фіксовані й невеликі за часом ітерації, називані **спринтами (sprints)**, надавати замовникові працююче ПЗ з новими можливостями, для яких визначений найбільший пріоритет.

Вимоги до результатів спринту визначаються на початку спринту (планування спринту) і не можуть змінюватися в процесі спринту. Невелика тривалість спринту надає процесу передбачуваність і гнучкість.

Ролі Scrum

1. **Scrum Master.** Бізнес-**а**налітик, керівник проекту. Проводить наради, стежить за дотриманням технології Scrum, знімає протиріччя й направляє команду. Основний обов'язок – забезпечення виконання технології Scrum.

2. **Product Owner.** Власник проекту, функціональний замовник. Представляє інтереси замовника (кінцевих користувачів).

3. **Development Team.** Команда фахівців (розроблювачів). Складається з фахівців різного профілю- аналітиків, архітекторів, програмістів, тестувальників. Команда відповідає за результат як єдине ціле.

Елементи Scrum

Sprints (Спринт)

Спринт – ітерація в процесі, у ході якої створюється новий результат. Спринт жорстко фіксований у часі – від 1 до 4 тижнів. Ніж коротше спринт, тим гнучкішим є процес розробки, оскільки після кожного спринту вимоги до системи можуть коректуватися на підставі зворотного зв'язка від замовника. Відповідно, знижуються ризики роботи в неправильному напрямку. З іншої сторони при більш тривалому спринті знижуються витрати на наради, і більше залишається часу на рішення завдань проекту.

Project backlog (журнал завдань проекту)

Project backlog – журнал побажань проекту. Це список вимог до системи, упорядкований за пріорітетом – важливості реалізації. Журнал побажань можуть доповнювати всі учасники процесу.

Sprint backlog (журнал завдань спринту)

Sprint backlog – журнал побажань спринту. Містить функціональність, відібрану власником проекту (**Product Owner**) для реалізації на поточному спринті.

Burndown chart (Діаграма згоряння завдань)

Burndown chart – діаграма згоряння завдань. Демонструє обсяг зробленої й роботи, що залишилася, щодо строку проекту. Діаграма актуалізується щодня. Передбачено два види діаграм:

– Діаграма згоряння для спринту – показує, скільки вже завдань зроблене й скільки ще залишається зробити в поточному спринті.

– Діаграма згоряння для проекту – показує, скільки вже завдань зроблене й скільки ще залишається зробити до завершення проекту.

Abnormal Termination (Зупинка спринту)

Abnormal Termination – зупинка спринту. Спринт може бути зупинений раніше його планового строку закінчення у виняткових ситуаціях. Наприклад, якщо завдання спринту не можуть бути досягнуті або якщо вони стали неактуальними. Рішення про зупинку приймається командою або Власником проекту. Після зупинки починається новий спринт.

Sprint Planning Meeting (Планування спринту)

Sprint Planning Meeting – планування спринту. Відбувається на початку кожного спринту:

– З беклога проекту вибираються завдання, зобов'язання по виконанню яких за спринт приймає на себе команда в даному спринті.

– На основі обраних завдань створюється беклог спринту. Кожне завдання оцінюється в годинах. Рішення завдання не повинне займати більше 12 годин або одного дня. При необхідності завдання розбивається на підзадачі.

– Обговорюється й визначається, яким образом буде реалізований цей обсяг робіт.

– Тривалість наради обмежена зверху 4-8 годинами залежно від тривалості ітерації, досвіду команди й тому подібного. Нарада ділиться на 2 частині:

– Бере участь власник проекту й скрам команда. Вибирають завдання з беклога проекту.

– Бере участь тільки команда: обговорюють технічні деталі реалізації, наповнюють **Sprint Backlog** (беклог спринту).

Daily Scrum meeting (Щоденна нарада)

Daily Scrum meeting – щоденна нарада команди. Правила проведення наради:

– проводиться в те саме час, у тому самому місці;

– не більше 15 мінут;

– кожному учасникові треба відповісти на 3 питання:

– Що я зробив учора

– Що я планую зробити сьогодні

– Що мені заважає

Scrum of Scrums (Скрам над скрамом)

Scrum of Scrums – Скрам над скрамом – нарада декількох Scrum-команд. Проводиться після щоденної скрам наради. Дозволяє декільком скрам командам обговорювати роботу, фокусуючись на загальних областях і взаємній інтеграції. Повістка та ж, що й на щоденному скрам нараді плюс наступні питання:

– Що кожна команда зробила з моменту попередньої щоденної наради?

– Що кожна команда зробить до наступної щоденної наради?

– Є чи проблеми, що заважають або сповільнюють роботу кожної команди?

– Потрібно чи іншій команді зробити щось із завдань вашої команди?

Sprint review meeting (огляд підсумків спринту)

Sprint review meeting – огляд підсумків спринту. Проводиться наприкінці спринту:

– Команда демонструє результати спринту всім зацікавленим особам. Залучається максимальна кількість глядачів.

– Всі члени команди беруть участь у демонстрації (одна людина на демонстрацію або кожного показує, що зробив за спринт).

– Не можна демонструвати незавершену функціональність.

– Обмежена чотирма годинами залежно від тривалості ітерації й обсягу результату.

Retrospective meeting (Ретроспективна нарада)

Retrospective meeting – ретроспективна нарада. Проводиться наприкінці спринту. Обговорення результатів спринту:

– Члени команди висловлюють своя думка про минулий спринт.

– Відповідають на два основних питання:

– Що було зроблено добре в минулому спринті?

– Що треба поліпшити в наступному?

– Виконують поліпшення процесу розробки (вирішують питання й фіксують удалі рішення).

– Обмежена одним-трьома годинами.

У цей час про гнучкі технології виконання проектів **Agile/**Scrum говорять всі частіше. У цій статті я напишу про своє бачення плюсів і мінусів застосування Scrum при виконанні проектів з позиції замовника.

Agile, Scrum, Kanban – ці слова вже встигли примелькатися всім, хто має яке-небудь відношення до розробки продуктів, бізнес-процесам, організації праці. Багато хто вважають, що за цими технологіями майбутнє. Але, що цікаво, деякі можуть пояснити розходження між цими поняттями, наприклад, відповістити на запитання про Scrum і Agile: різниця існує й у чому вона полягає?».

Отже, спершу визначимося з методологічним апаратом:

Agile – **клас гнучких методологій розробки**; набір цінностей і базових принципів в області розробки ПЗ, основна суть яких зводиться до взаємодії функціональних крос-команд що самоорганізуються. У суть Agile закладені адаптивне планування, еволюційна розробка, максимально швидкий випуск нових версій продукту, постійна доробка й швидка відповідь на виникаючі складності. На сьогоднішній день Agile уже давно вийшов за рамки розробки програмного забезпечення, а його різні дочірні методології використовуються в різних сферах керування.

Scrum – одна з методологій у рамках Agile. В основі даної методології лежать «спринти» – короткі жорстко фіксовані за часом ітерації, у рамках яких як виконавець так і замовник виконуються певний набір завдань.

Kanban – система, що передбачає організацію розробки/ праці/ виробництва/ постачання відповідно до принципу Just In Time (або точно в строк). В основі Kanban лежить виробнича й постачальницька система Toyota, що

у другій половині XX століття ледве було не знищила весь американський автопром.

Основними цінностями в концепції **Scrum** оголошені комунікація й готовність до змін, а не тверде проходження первісному плану. Коли ж **Scrum** може принести тільки користь, а коли – шкода?

Тверда підрядна технологія

Суть підряду дуже проста. Замовник замовляє й чітко визначає в специфікації до договору, що саме він хоче одержати**.** У договорі чітко фіксується вартість і строк. Замовник передає виконавцеві матеріали, придатні для обробки. Виконавець перевіряє матеріали, і якщо вони непридатні, то негайно сповіщає про це Замовникові й зупиняє роботи. Замовник зобов'язаний робити Виконавцеві сприяння в процесі робіт. Якщо Виконавець не уклався в строк, або надав настільки неякісний результат, що можна припустити недосягнення результату як такого – Замовник має право розірвати договір і зажадати повернення авансу в повному розмірі. Якщо ж результат досягнуть, то Замовник приймає роботи, підписує акт виконаних робіт, оплачує роботу.

При виконанні проектів по впровадженню ERP-систем звичайно полягають саме такі договори. У договорі визначаються фіксований бюджет і строк на кожний етап робіт, саме завдання (безпосередньо в договорі або окремо в ТЗ), обов'язку Замовника на кожному етапі, однозначні критерії результату. Зміни у вимогах у процесі робіт або не допускаються зовсім, або додатковий обсяг робіт оплачується додатково за окремою згодою.

Підкреслимо ключове положення цієї моделі взаємодії: у результаті оцінки Виконавцем поставленої споконвічно завдання в договорі встановлюється твердий строк і вартість етапу. Тому зміна завдання в процесі виконання робіт не допускається. Це правило діє, наприклад, і при ремонті квартири, і при впровадженні ERP-систем.

Якщо використовується ітераційний підхід, то після закінчення етапу робіт стають ясними завдання наступного етапу, а виходить, що випливають

обсяги робіт. Відповідно, у додатковій угоді фіксується строк і вартість на наступний етап. Буває ще цікавіше – при впровадженні ERP-систем Виконавець (найчастіше щоб виграти тендер) фіксує бюджет на весь проект цілком, іноді на 1-2 роки вперед.

Замовники приймають бюджет, думаючи, що й строки, і бюджети будуть дотримуватися. При цьому, відповідно до договору, етапи робіт у кожному разі закриваються актами й оплачуються окремо, чи ледве не щомісячно. Не виключено, що через кілька місяців з'ясується, що бюджет необхідно збільшити в 2-3 рази (мов, вимоги до системи змінилися, а «ми про це не домовлялися, у ТЗ цього немає!»). Не хочете – до побачення.

Основний недолік твердої технології – істотні обмеження гнучкості у вимогах і рішеннях, в умовах виконання робіт, обмеження гнучкості участі Замовника в процесі виконання етапу. Така гнучкість у ході виконання робіт украй важлива, тому що на початку кожного етапу детальні, вичерпні, точні вимоги й організаційні умови сформулювати важко – високий ступінь невизначеності системи. Інтегратори-виконавці часто говорять: «Клієнт сам не знає, чого хоче»… Зовсім тупикові ситуації виникають, коли клієнт – який не знає, чого хоче – вимагає зробити систему «під ключ».

Оскільки строки й бюджети визначаються на початку кожного етапу, будь-які коректування вимог до системи з боку Замовника, а також будь-які зміни умов у процесі виконання робіт на етапі (наприклад, обсяг участі й сприяння з боку Замовника) можуть створити додаткову трудомісткість для Виконавця, що зажадає перегляду бюджету й строків. Але перегляд бюджетів і строків – потенційно конфліктна ситуація!

Отже, Замовник зацікавлений у можливості зміни вимог у процесі виконання робіт, але підрядна технологія не дає йому цього робити, викликаючи конфлікт із Виконавцем.

Для підвищення гнучкості – у бюджет і строки заставляються ризики. Крім того, досвідчені Виконавці передбачають і прописують процедуру

«Керування змінами», відповідно до якої в остаточному підсумку всі зміни первісних вимог проходять через процедури ініціювання й оцінки.

Вартість ризиків Виконавця, відбитих у бюджеті проекту, у подібних роботах може досягати більше 50% від вартості етапу. Інакше кажучи, Виконавець продає замовникові не тільки результат робіт, але й свої ризики. Якщо ризики не реалізуються, вартість робіт для Замовника виявляється завищеною щодо реальної трудомісткості цих робіт, а Виконавець дістає додатковий прибуток. Буває й навпаки – реалізовуються незаплановані ризики понад бюджет, у результаті чого Виконавець виявляється в збитку.

Таким чином, підрядна технологія обмежує гнучкість Замовника в зміні вимог, умов роботи й ступеня своєї участі, коли роботи вже початі, і «корабель відійшов від берега». Необхідна процедура керування змінами... але немає гарантії, що ця процедура зможе передбачити всі ситуації.

Більші витрати сторони несуть на формулюванні всіх умов робіт і вимог до результату при початку етапу, а також на формалізації правил керування змінами. І це при тім, що умови й вимоги можуть швидко втрачати актуальність у процесі виконання етапу! У результаті частина розробленого й оплаченого функціонала може виявитися незатребуваною в процесі експлуатації системи – нерідка ситуація при підрядному способі.

Чому ж Замовники, як правило, вимагають, щоб впровадження складних систем виконувалося по договорах підряду? Відповідь проста:

– Інші способи поділу відповідальності для Замовника менш комфортні. У договорі Замовник замовляє, а Виконавець – виконує, ця логіка породжує поділ відповідальності сторін за проект, зручне для Замовника.

– Інші (не підрядні) технології вимагають більшої відповідальності з боку замовника.

Чому Виконавці охоче погоджуються на підряд?

– Щоб забезпечити гнучкість у вимогах, Виконавці закладають у фіксовану вартість проекту ризики. Якщо ризики не реалізуються, Замовник переплачує.

– Вимоги до результату погодяться «на березі», а виходить, Замовник знову втрачає гнучкості, а Виконавець одержує можливість не робити потрібний Замовникові функціонал, посилаючись на те, що «у ТЗ цього немає». Так працювати спокійніше: зафіксував вимоги й “уперед”...

Гнучкість вимог у процесі виконання робіт

Через високу невизначеність системи на початку проекту й неможливості передбачити всі, замовники прагнуть міняти вимоги в процесі виконання робіт. З'являються нові ідеї, відбуваються зміни на підприємстві, з'ясовується, що замість інтеграції зі старою системою краще доробити нову систему й т.д.

Гнучкість вимог життєво необхідна для досягнення результату, дійсно потрібного підприємству.

Гнучка технологія (Scrum)

Зміст концепції Scrum у тому, що кожний етап робіт розбивають на короткі ітерації фіксованої тривалості (від 1-й тижня до місяця). На початку кожної ітерації визначають завдання сторін – Замовника й Виконавця – тільки на дану ітерацію. У процесі ітерації (у термінології Scrum – «**Спринт**») зміни в завданнях Сторін, вимогах і умовах виконання робіт не допускаються. По завершенні ітерації Сторони аналізують результати й ставлять завдання на наступну ітерацію; і при цьому вужі **допускається змінювати умови** виконання робіт.

Дуже важливо щільна й безперервна взаємодія Виконавця й Замовника. Якнайбільше спілкуватися, бути однією командою – от ключовий момент даного підходу!

Перевага гнучкої технології:

– За рахунок того, що ітерації набагато коротше, ніж весь етап робіт, досягається набагато більша гнучкість у прийнятті рішень і керуванні змінами в

ході робіт. Між ітераціями Замовник не обмежений у прийнятті організаційних рішень, зміні рамок проекту й так далі. Очевидно, що сторони рухаються до результату більше коротким шляхом, чим при підряді, оскільки можуть безупинно вносити зміни в проект.

– Виключаються конфліктні ситуації через зміни, тому що в центр такої технології поставлена можливість зміни. Якщо при підряді зміна – це неприємне відхилення, що вимагає розбирань і з'ясування, хто кому повинен, то в Scrum зміни – це суть і складова процесу.

– Тарифікація ітерацій теж може бути гнучкої – або оплата виробляється по фактичній трудомісткості (відповідно до погодинної ставки роботи співробітника); або вартість ітерації фіксується на початку ітерації, виходячи із прогнозу її трудомісткості. У результаті, Замовник оплачує тільки фактичну трудомісткість і не переплачує за ризики.

Прогноз вартості кожної ітерації виконується досить просто по погодинному тарифі (якщо відомо кількість співробітників Виконавця, планованих на ітерацію з повним завантаженням).

Таким чином, при використанні гнучких технологій бюджет проекту на певний період будується на підставі прогнозу трудомісткості й кількості приваблюваних співробітників Виконавця.

Як правило, при використанні гнучких технологій вартість проекту не перевищує його вартості при використанні твердих технологій, оскільки кінцева вартість прямо залежить від сумарної трудомісткості. Трудомісткість гнучкої технології може виявитися навіть менше за рахунок того, що не розробляється незатребуваний функціонал, а виходить, менше буде й вартість проекту в цілому.

Однак технології Scrum властиві свої недоліки. Один з них – ризик несистемного підходу. Визначення всіх вимог до системи до початку робіт дозволяє продумати систему в цілому ще до її реалізації. У ході реалізації частини системи враховуються властивості всієї системи, всі вимоги до неї, а не тільки до одному з її сегментів. У цьому й полягає системний підхід.

При несистемному підході висока ймовірність, що замість системи вийде набір незв'язаних елементів – зовсім як у древній східній казці, у якій мудреці за хвостом, хоботом і ногами не побачили слона:

Тому якщо їсти можливість ретельно продумати функціонал і вимоги до початку реалізації, їй треба скористатися. Зрозуміло, якщо є вся необхідна інформація. Це ідеальний випадок, і таке буває рідко, тому найкращим варіантом у таких ситуаціях є Scrum, що дозволяє формувати вимоги до системи в процесі її реалізації й експлуатації. Якщо ж зміни не передбачаються, то Scrum – не кращий вибір.

Наприклад, при розробці типових рішень (не під конкретного замовника з його неочевидними «хотелками») систему краще продумувати відразу в повному обсязі, цілісно. Оскільки зовнішнього непередбаченого замовника немає – у цьому випадку замовником, по суті, є сама команда розроблювачів – вимоги до системи відомі споконвічно. Звичайно, якщо розроблювачі знають, за що беруться.

Отже, керування змінами при гнучкій технології Scrum відбувається найбільше ефективно. Варто врахувати, що в проектах з високим ступенем невизначеності системи на початку робіт керування змінами є ключовим чинником, що впливає як на кінцевий результат, так і на оптимізацію бюджету. У тому числі значно знижуються ризики розробки непотрібного функціонала, або функціонала, що не відповідає потребам бізнесу.

Договір підрядний, а працюємо по Scrum

Переваги Scrum всі частіше можна спостерігати, коли реалізується один із класичних проектних ризиків – “звалювання” проекту в Scrum по факту, хоча за договором робота виконується по підряду – із твердими строками й вартістю. Таке “звалювання” дуже вигідно активному Замовникові, тому що дозволяє досягти бажаного результату, відштовхуючись щораз від фактичного результату проміжного етапу робіт. Замовникові набагато простіше зрозуміти, що він хоче одержати насправді, якщо він відштовхується від того, що вже зроблено.

Така ситуація характерна не тільки для IT-проектів. При ремонті квартири, будівництві будинку, і будь-якому проекті, коли складно чітко формалізувати остаточний результат, Замовник може почати контролювати безупинно кожний дрібний етап і вносити коректування в споконвічну постановку завдання. Це приводить Виконавця в замішання, тому що строки й трудомісткість роботи неконтрольована ростуть, а бюджет проекту не міняється. Виконавець починає думати, що «Замовник не знає, що хоче», це потенційно конфліктна ситуація. Насправді, Замовник і не може знати, що хоче – це цілком нормально. Що він хоче, він починає розуміти тільки після того як побачить, що виходить по факті – тільки після чергової ітерації робіт. Це Scrum.

На IT проекті "звалювання" проявляється в такий спосіб.

Як правило це відбувається на етапі реалізації (кодування). Як тільки Замовник просить показувати проміжні результати, і починає їх приймати, не чекаючи пред'явлення всього обсягу робіт на випробування, чекайте перехід по факті в Scrum. Приймання Замовником уже першої ітерації робіт швидше за все спричинить частковий перегляд Замовником усього технічного завдання, а це вже Scrum. Зрозуміло, для кінцевого результату й задоволення Замовника це благо. А перегляд ТЗ – це зміна трудомісткості, звичайно в більшу сторону. І якщо не укласти додаткову угоду на збільшення вартості, то ущемляються інтереси Виконавця. А це конфліктна ситуація.

Виникає питання – чи не простіше споконвічно домовитися працювати по Scrum? Адже справедлива оплата робіт повинна відповідати фактичній трудомісткості, що при "звалюванні" в Scrum не піддається точному розрахунку!

Що ж виходить? Замовник може заперечувати Scrum при укладанні договору, але потім фактично змусити Виконавця працювати по Scrum. У рамках фіксованого строку й бюджету. І не забуваємо, що прострочення строку здачі робіт (через постійне внесення Замовником дрібних коректувань у завдання) загрожує Виконавцеві серйозною відповідальністю, тому що Виконавець відповідає за строк здачі робіт. Невиконання строку надає право Замовникові

висувати обґрунтовані претензії. Більше того, Замовник може дорікати Виконавцю в недостатньому професіоналізмі, тому що Замовник схильний часто свої коректування в постановку Завдання розцінювати як свої вимоги виправити недоліки.

3.2 Розробка структурної схеми

На рисунку 3.1 зображена структурна схема програмного забезпечення, яке реалізує процес безпечної розробки при використанні технології Agile з застосуванням обфускації коду. Розглянемо цю схему більш детально.

Процес безпечної розробки при використанні технології Agile з застосуванням обфускації може бути здійснений над кожним з вище перерахованих видів подання програмного коду, тому прийнято виділяти наступні рівні процесу безпечної розробки при використанні технології Agile з застосуванням обфускації:

– нижчий рівень, коли процес безпечної розробки при використанні технології Agile з застосуванням обфускації здійснюється над ассемблерним кодом програми, або навіть безпосередньо над двійковим файлом програми, що зберігає машинний код;

– вищий рівень, коли процес безпечної розробки при використанні технології Agile з застосуванням обфускації здійснюється над вихідним кодом програми написаному мовою високого рівня.

Здійснення безпечної розробки при використанні технології Agile з застосуванням обфускації на нижчому рівні вважається менш комплексним процесом, але при цьому більш важко реалізованим з ряду причин. Одна із цих причин полягає в тому, що повинні бути враховані особливості роботи більшості процесорів, так як спосіб безпечної розробки при використанні технології Agile з застосуванням обфускації, прийнятний на одній архітектурі, може виявитися неприйнятним на іншій.

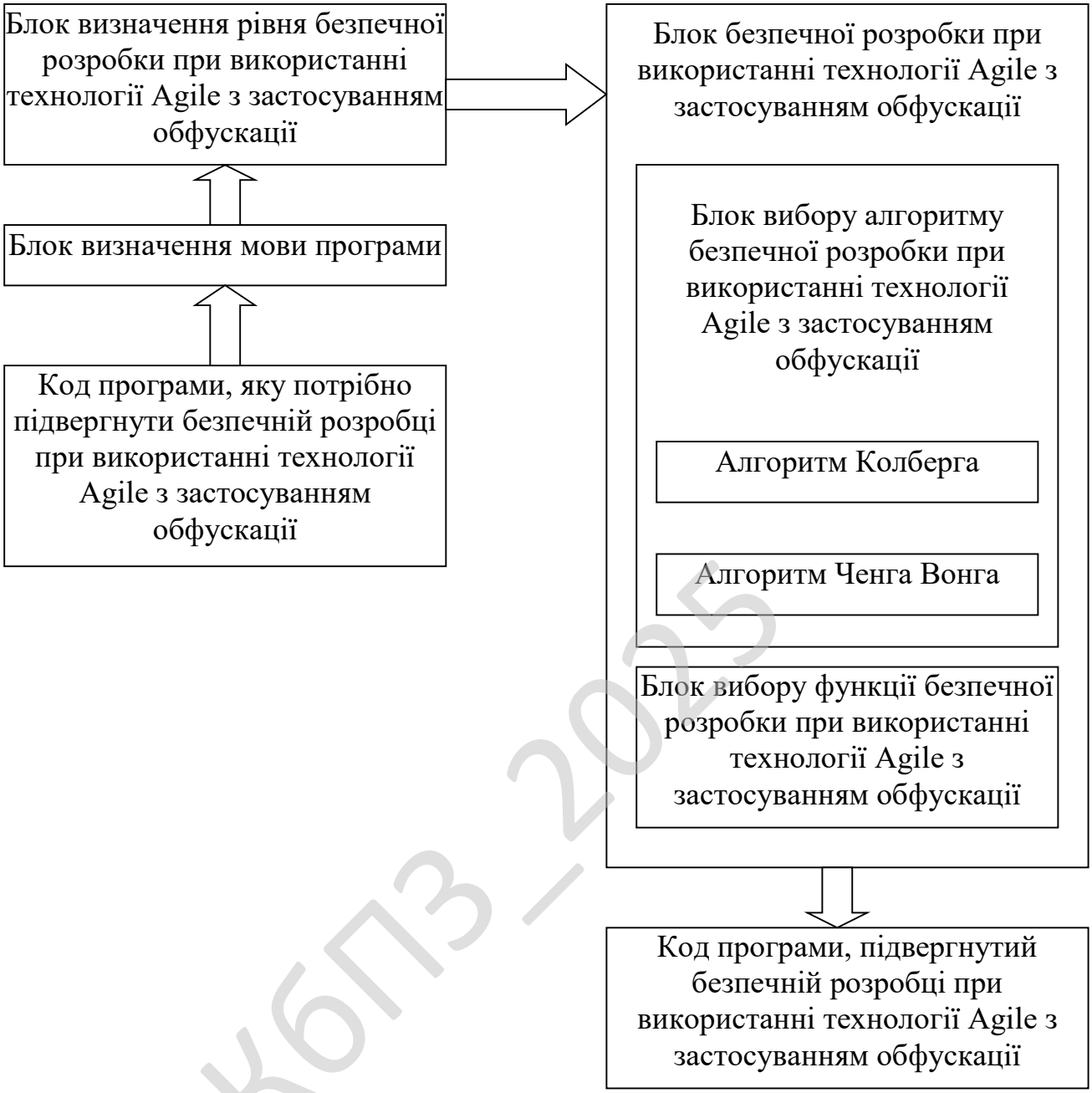

Рисунок 3.1 – Структурна схема системи

Також на сьогоднішній день процес низькорівневої безпечної розробки при використанні технології Agile з застосуванням обфускації досліджений мало (напевно так як не встиг одержати широкої популярності).

Більшість існуючих алгоритмів і методів безпечної розробки при використанні технології Agile з застосуванням обфускації (включаючи ті які будуть розглянуті нижче) можуть бути застосовані для здійснення процесу

безпечної розробки при використанні технології Agile з застосуванням обфускації як на нижчому, так і на вищому рівні.

Також іноді може бути неефективно, піддавати безпечній розробці при використанні технології Agile з застосуванням обфускації весь код програми (наприклад, через те, що в результаті може значно знизиться час виконання програми), у таких випадках доцільно здійснювати обфускацію тільки найбільш важливих ділянок коду.

Алгоритми процесу безпечної розробки при використанні технології Agile з застосуванням обфускації

Алгоритм безпечної розробки при використанні технології Agile з застосуванням обфускації в більшості випадків розглядається як алгоритм, який повинен дотримуватися обфускатор (незалежна програма, що здійснює процес безпечної розробки при використанні технології Agile з застосуванням обфускації над переданим їй кодом).

На даний момент існують різні алгоритми здійснення процесу безпечної розробки при використанні технології Agile з застосуванням обфускації, починаючи від загальних (абстрактних) алгоритмів процесу безпечної розробки при використанні технології Agile з застосуванням обфускації й закінчуючи більше просунутими. Ці алгоритми створювалися відповідно до можливостей тої або іншої мови програмування, і на сьогодні більшість із них адаптовано безпосередньо під мови програмування високого рівня. Нижче представлене короткий опис деяких з них.

Алгоритм Колберга ("Collberg`s algorithm").

Даний алгоритм оперує наступними вхідними значеннями:

– Програма "A", яка складається з вихідних або об'єктних (двійкових) файлів "{C1,C2}".

– Стандартні бібліотеки, використовувані програмою "{L1,L2}".

– Набір процесів, що трансформують, "T{T1,T2}".

– Певний фрагмент коду "S", що витягає із програми "A", і який безпосередньо буде підданий трансформації.

– Набір функцій "E{E1,E2}" які будуть визначати ефективність застосування певних процесів, що трансформують, "{T1,T2}" до фрагмента коду "S".

– Набір функцій "I{I1,I2}" які будуть визначати важливість фрагмента коду "S", і залежно від цього будуть задавати певне значення змінної "RequireObfuscation" (чим "S" важливіше тим ця змінна буде зберігати більше значення).

– Дві числові змінні "AcceptCost" > 0, "RequireObfuscation" > 0, де перше зберігає інформацію про доступне максимальне збільшення системних ресурсів які потрібні для програмі "A" після того як вона піддасться безпечної розробки при використанні технології Agile з застосуванням обфускації, а друг змінна буде зберігати значення необхідного рівня здійснення безпечної розробки при використанні технології Agile з застосуванням обфускації (чим важливіше фрагмент коду "S", тим це значення повинне бути більше).

Алгоритм Колберга має таку послідовність операцій:

– Завантаження елементів "{C1,C2}" програми "A".

– Завантаження бібліотек "{L1,L2}".

– Здійснення безпечної розробки при використанні технології Agile з застосуванням обфускації над програмою "A", шляхом виділення фрагмента коду "S" і визначення найбільш ефективного процесу трансформації для нього. Цей етап повторюється доти, поки не буде, досягнуть необхідний рівень безпечної розробки при використанні технології Agile з застосуванням обфускації "RequireObfuscation" або припустиме збільшення ресурсів "AcceptCost".

– Генерація программ, що трансформується, "A'".

Алгоритм Колберга вважається загальним алгоритмом здійснення процесу безпечної розробки при використанні технології Agile з застосуванням обфускації (тобто він не визначає, як саме повинен здійснюватися, той або інший метод

безпечної розробки при використанні технології Agile з застосуванням обфускації), нижче буде розглянутий більше спеціалізований алгоритм, так як він описує послідовність здійснення одного з методів безпечної розробки при використанні технології Agile з застосуванням обфускації, а саме безпечної розробки при використанні технології Agile з застосуванням обфускації керування.

Алгоритм Ченги Ванга.

У якості вхідних даних алгоритм приймає типову процедуру, написану мовою високого рівня. Процес безпечної розробки при використанні технології Agile з застосуванням обфускації кожної такої процедури складається із трьох етапів:

– Створення графа потоку керування цієї процедури (граф задається безліччю блоків і безліччю зв'язків з'єднуючих їх), після чого граф розбивається, шляхом заміни циклічних конструкцій у ньому на конструкції типу "if (умова) goto".

– Нумерація всіх блоків у графі, і додавання в код процедури змінної (наприклад "swVar"), яка зберігає номер наступного виконуваного блоку.

– Приведення графа до однорідного ("плоскому") виду.

Вище описаний варіант алгоритму безпечної розробки при використанні технології Agile з застосуванням обфускації ("Chenxi Wang's algorithm") є не сильно стійким, так як визначити наступний виконуваний блок, неважко (він у нашім випадку буде зберігатися в змінної "swVar"). Тому для підвищення його стійкості вводять масив (наприклад "@gg"), що містить крім номерів блоків, не потрібну інформацію, у результаті запис "$swVar = S6", можна замінити на щось подібне "$swVar = $gg[$gg[1] + $gg[3]]".

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

З цього рисунку ми бачимо, що програма безпечної розробки при використанні технології Agile з застосуванням обфускації виконує наступні дії над кодом, який визначається заданими видами безпечної розробки при використанні технології Agile з застосуванням обфускації:

1. Лексична обфускація:

– Видалення всіх коментарів у коді програми.

– Видалення різних пробілів.

– Заміна імен ідентифікаторів.

– Додавання різних зайвих операцій.

– Зміна розташування блоків.

2. Обфускація даних.

а) Обфускація зберігання:

– Зміна інтерпретації даних певного типу.

– Зміна строку використання сховищ даних.

– Перетворення статичних даних у процедурні.

– Поділ змінних.

– Зміна подання (або кодування).

б) Обфускація з'єднання:

– Об'єднання змінних.

– Реструктурування масивів.

– Зміна ієрархій спадкування класів

в) Обфускація переупорядкування.

3. Обфускація керування.

а) Обчислювальна обфускація:

– Розширення умов циклів.

– Додавання недосяжного коду.

– Усунення бібліотечних викликів.

– Додавання надлишкових операцій.

– Розпаралелювання коду.

б) Обфускація з'єднання:

– Вбудовування функцій.

– Добування функцій.

– Чергування, об'єднання фрагментів коду програми.

– Клонування.

– Трансформація циклів.

– Розгорнення циклів.

– Поділ циклів.

в) Обфускація послідовності.

Перейдемо до більш детального опису функціональної схеми розробленого програмного продукту безпечної розробки при використанні технології Agile з застосуванням обфускації коду програми.

Процеси безпечної розробки при використанні технології Agile з застосуванням обфускації можна класифікувати по видах, залежно від способу модифікації коду програми.

Лексична обфускація

Найбільш проста, полягає у форматуванні коду програми, зміні його структури, таким чином, щоб він став нечитабельним, менш інформативним, і важким для вивчення.

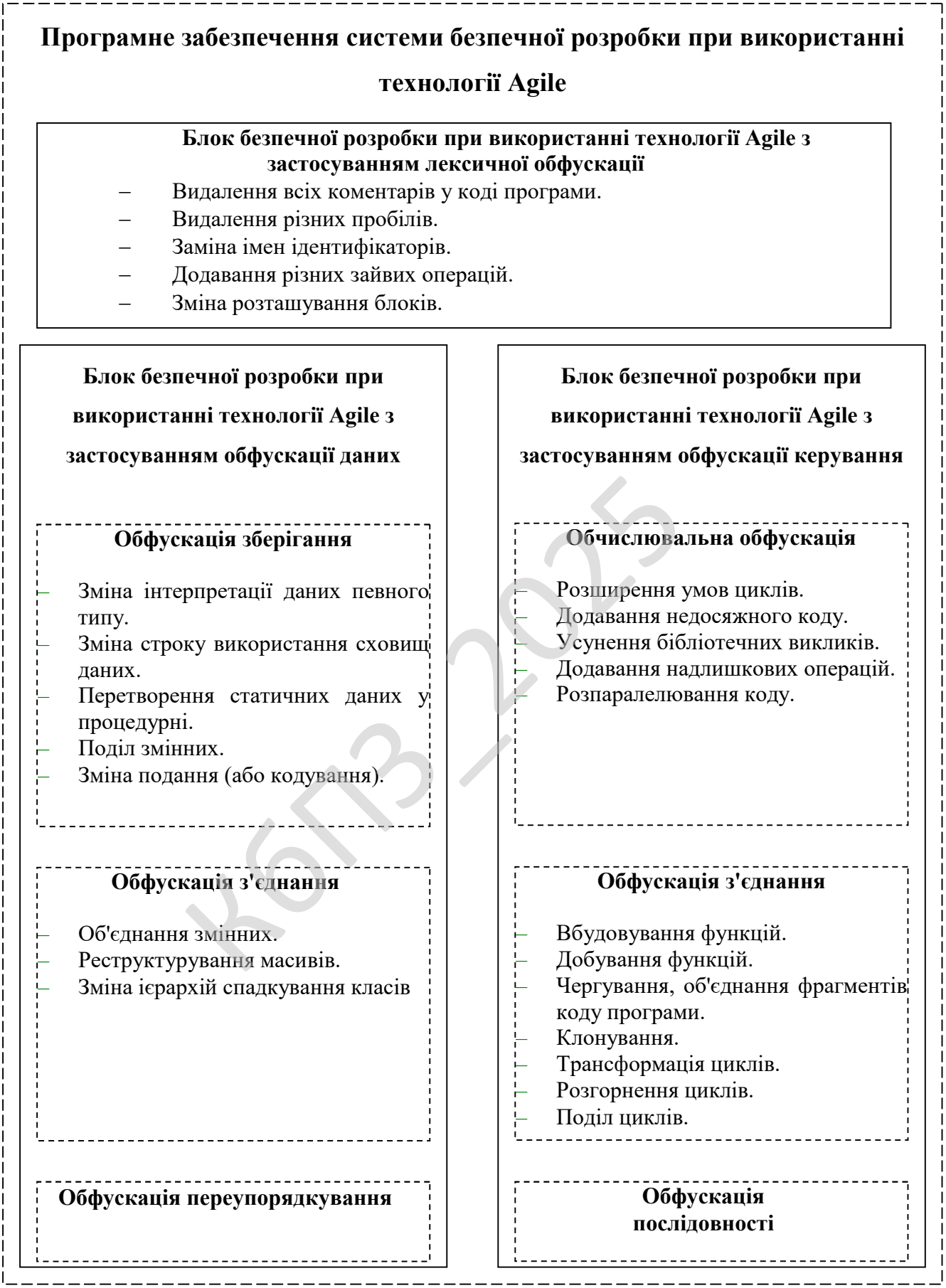

Рисунок 3.2 – Функціональна схема системи

Обфускація такого виду містить у собі:

– Видалення всіх коментарів у коді програми, або зміна їх на ті, що дезінформують.

– Видалення різних пробілів, відступів які звичайно використовують для кращого візуального сприйняття коду програми.

– Заміну імен ідентифікаторів (імен змінних, масивів, структур, хешів, функцій, процедур і т.д.), на довільні довгі набори символів, які важко сприймати людині.

– Додавання різних зайвих (сміттєвих) операцій.

– Зміна розташування блоків (функцій, процедур) програми, таким чином, щоб це не яким образом не вплинуло на її працездатність.

Зміна глобальних імен ідентифікаторів варто робити в кожній одиниці трансляції (один файл вихідного коду), так щоб вони мали однакові імена (у противному випадку програма, яка захищається, може стати не функціональною). Також варто враховувати специфічні ідентифікатори, прийняті в тій мові програмування, на якому написана програма, яка захищається, імена таких ідентифікаторів, краще не змінювати.

Дана обфускація програмного коду, у порівнянні з іншими, дозволяє порівняно швидко привести вихідний код програми, у нечитабельний стан. Один з її недоліків полягає в тому, що вона ефективна тільки для здійснення високорівневої безпечної розробки при використанні технології Agile з застосуванням обфускації.

Обфускація керування

Обфускація такого виду здійснює заплутування потоку керування, тобто послідовності виконання програмного коду.

Більшість її реалізацій ґрунтується на використанні непрозорих предикат, у якості, який виступають, послідовності операцій, результат роботи яких складно визначити (саме поняття "предикат" виражає властивість одного об'єкта (аргументу), або відносини між декількома об'єктами).

Визначення. Предикат "P" уважається непрозорим предикатом, якщо його результат відомий тільки в процесі безпечної розробки при використанні технології Agile з застосуванням обфускації, тобто після здійснення процесу безпечної розробки при використанні технології Agile з застосуванням обфускації, визначення значення такого предиката, стає важким.

Позначимо непрозорий предикат, що повертає завжди значення TRUE як "P(t)", а повертаючий значення FALSE, як "P(f)", тоді непрозорий предикат, що може повернути кожне із цих двох значень (тобто або TRUE, або FALSE, що нам невідомо) як "P(t,f)". Ці позначення, будуть використовуватися далі в контексті опису безпечної розробки при використанні технології Agile з застосуванням обфускації керування. Непрозорі предикати можуть бути:

– Локальними – обчислення втримуватися усередині одиночного вираження (умови), наприклад (запис "(f)" після умови перевірки, указує, що це предикат типу "P(f)").

– Глобальними – обчислення втримуватися усередині однієї процедури (функції.

– Міжпроцедурними – обчислення втримуватися усередині різних процедур (функцій).

Ефективність безпечної розробки при використанні технології Agile з застосуванням обфускації керування в основному залежить від використовуваних непрозорих предикат, це змушує створювати як можна складні для вивчення, і прості, гнучкі у використанні непрозорі предикати, але рівною мірою також не маловажну роль має час їхнього виконання, а також кількість виконуваних операцій, крім усього цього предикат не сильно повинен відрізнятися від тих функцій, які виконує сама програма, і не повинен містити надмірну кількість обчислень, у противному ж випадку зловмисник, зможе відразу його виявити. Так як часто для дебезпечної розробки при використанні технології Agile з застосуванням обфускації використовують технологію статичного аналізу, а одним з її недоліків є складність (трудомісткість) статичного аналізу структур

покажчиків, то звичайно в процесі безпечної розробки при використанні технології Agile з застосуванням обфускації керування використовують стійкі непрозорі предикати, які дозволяють використовувати недоліки технології статичного аналізу.

Основна ідея стійких непрозорих предикатів полягає в тому, що в програму, у процесі безпечної розробки при використанні технології Agile з застосуванням обфускації додається код, що створює набір динамічних структур, а також глобальних покажчиків, які будуть посилатися на різні елементи усередині цих структур. Крім цього, даний код повинен іноді обновляти ці структури (додавати нові елементи в них, поєднувати або розділяти деякі їх, змінювати значення глобальних покажчиків, і т.д.), але таким чином, щоб при цьому минулому збережені деякі умови, наприклад "покажчик p і q ніколи не будуть указувати на той самий елемент" або "покажчик p може посилатися (указувати) на покажчик q" і таке інше. Ці умови в наслідку дозволяють створювати необхідні непрозорі предикати.

Методи що дозволяють здійснити обфускацію керування, класифікуються на три основних групи:

Обчислювальна обфускація. Зміна дотичного головної структури потоку керування. До них можна віднести:

– Розширення умов циклів. Для цього звичайно використовують непрозорі предикати, таким чином, щоб вони не яким образом не впливали на кількість виконань циклічного коду.

– Додавання недосяжного коду, (який не буде виконуватися в процесі роботи програми).

– Усунення бібліотечних викликів. Більшість програм, використовують функції, які визначені в стандартних бібліотеках вихідної мови, на якому писалася програма (наприклад, у Сі це бібліотека "libc"), робота таких функцій добре документована й часто відома зловмисникам, отже, їхня присутність у коді програми, може допомогти в процесі її реверсивної інженерії. Тому імена

функцій зі стандартних бібліотек, також бажано додати безпечної розробки при використанні технології Agile з застосуванням обфускації, тобто змінити на найбільш безглузді, які потім будуть фігурувати в коді програми, яка захищається. Один зі способів рішення такої проблеми, полягає у використанні в програмі власної версії стандартних бібліотек (які виходять у результаті перейменовування всіх функцій в оригінальній стандартній бібліотеці), це не змінить істотно час виконання програми. Але для того, щоб така програма була стерпної, і могла використовуватися багатьма користувачами, її потрібно буде поставляти разом зі зміненою версією стандартної бібліотеки, що значно збільшить розмір програми. Тому такий спосіб рішення проблеми неефективний. При здійсненні такої безпечної розробки при використанні технології Agile з застосуванням обфускації треба в першу чергу ґрунтуватися на особливостях стандартної бібліотеки вихідної мови (те, як у ній взаємозалежні імена функцій з їхніми кодами й т.д.).

– Додавання надлишкових операцій (мертвого коду) у ті ділянки програмного коду, які найбільш важкі (визначально) для вивчення. Часто надлишкові операції, використовуються для розширення арифметичних виражень (наприклад, у непрозорих предикатах)

– Розпаралелювання коду, полягає в поділі коду на окремі незалежні ділянки, які під час роботи програми будуть виконуватися паралельно (тобто одночасно), така обфускація також може полягати в імпровізації розпаралелювання коду програми, для цього створюється так званий макет процесу, що насправді не буде виконувати ніяких корисних операцій.

Обфускація з'єднання. Об'єднання або поділ певних фрагментів коду програми, для того щоб забрати логічні зв'язки між ними. Нижче наведені основні методи, що дозволяють здійснити таку обфускацію:

– Вбудовування функцій, здійснюється шляхом вбудовування коду функції, у місця її виклику (якщо її код буде убудований в усі місця її виклику, тоді саму функцію можна забрати з коду програми).

– Добування функцій, є зворотною дією, стосовно вбудовування функцій. Здійснюється в результаті об'єднання деякої групи взаємозалежних операторів у коді вихідної програми в окрему функцію (при необхідності для цієї функції можна визначити деякі аргументи), що потім заміщають ці групи операторів. Але варто врахувати, що таке перетворення може бути знято компілятором у процесі компіляції коду програми.

– Чергування, об'єднання фрагментів коду програми (функцій наприклад), що виконують різні операції, воєдино (в одну функцію, при цьому в таку функцію, варто додати об'єкт, залежно від значення якого, буде виконуватися код однієї з об'єднаних функцій).

– Клонування, даний метод дозволяє ускладнити аналіз контексту використання функцій, і об'єктів використовуваних у коді вихідної програми. Процес клонування функцій складається у виділенні певної функції "F", часто використовуваної в коді програми, після чого над кодом цієї функції здійснюється трансформація, і створюється її клон "F'", що також буде доданий у код вихідної програми, при цьому частина викликів функції "F" у коді вихідної програми, буде заміщена на виклик функції "F'". У результаті цього в зловмисника створиться подання про те, що функції "F", і "F'" різні. Клонування об'єктів здійснюється аналогічним способом.

– Трансформація циклів. Цикли зустрічаються в коді різних програм, і їх також можна додати трансформації. Блокування циклів, полягає в додаванні вкладених циклів в існуючі, у результаті робота існуючих циклів буде заблокована, на якийсь діапазон значень.

– Розгорнення циклів, повторення тіла циклу один або більше раз (якщо кількість виконуваних циклів відомо в процесі здійснення безпечної розробки при використанні технології Agile з застосуванням обфускації (наприклад, дорівнює "N"), то цикл, може бути, розгорнуть повністю, у результаті повторення його тіла в коді N раз).

– Поділ циклів, цикл, що складається з більш ніж однієї незалежної операції можна розбити на кілька циклів (які повинні виконуватися однакову кількість разів), попередньо розбивши на кілька частин, його тіло.

Бажано здійснювати над вихідним циклом послідовно всі перераховані вище трансформації циклів, це дозволить ускладнити його статичний аналіз.

Обфускація послідовності. Полягає в переупорядкуванні блоків (інструкцій переходів), циклів, виражень.

Обфускація даних

Така обфускація пов'язана із трансформацією структур даних. Вона вважається більше складною, і є найбільш просунутою й часто використовуваною. Її прийнято ділити на три основні групи, які описані нижче.

Обфускація зберігання. Полягає в трансформації сховищ даних, а також самих типів даних (наприклад, створення й використання незвичайних типів даних, зміна подання існуючих і т.д.). Нижче наведені основні методи, що дозволяють здійснити таку обфускацію:

– Зміна інтерпретації даних певного типу. Як відомо збереження, будь яких даних у сховищах (змінних, масивах і т.д.) певного типу (ціле число, символ) у процесі роботи програми, дуже розповсюджене явище. Наприклад, для переміщення по елементах масиву дуже часто використовують змінну типу "ціле число", що виступає в ролі індексу. Використання в цьому випадку змінних іншого типу можливо, але це буде не тривіально й може бути менш ефективно. Інтерпретація комбінацій розрядів даних, що втримуються в сховищі, здійснюється залежно від його типу. Так, наприклад, можна сказати, що 16-розрядна змінна цілого типу утримуючої комбінації розрядів 0000000000001100 представляє ціле число 12, але це проста угода, дані в такий змінній можна інтерпретувати по-різному (не обов'язково як 12, а, наприклад як 1100 і т.д.).

– Зміна строку використання сховищ даних, наприклад перехід від локального їхнього використання до глобального й навпаки.

– Перетворення статичних (незмінюванних) даних у процедурні. Більшість програм, у процесі роботи, виводять різну інформацію, що найчастіше в коді програми представляється у вигляді статичних даних таких як рядка, які дозволяють візуально орієнтуватися в її коді й визначати виконувані операції. Такі рядки також бажано змінити безпечної розробки при використанні технології Agile з застосуванням обфускації, це можна зробити, просто записуючи кожний символ рядка, використовуючи його ASCII код, наприклад символ "A" можна записати як 16-річне число "0x41", але такий метод банальний. Найбільш ефективний метод, це коли в код програми в процесі здійснення безпечної розробки при використанні технології Agile з застосуванням обфускації додається функція, що генерує необхідний рядок відповідно до переданими їй аргументами, після цього рядка в цьому коді віддаляються, і на їхнє місце записується виклик цієї функції з відповідними аргументами. Також до статичних даних відносяться числові константи, які можуть бути також трансформоване, наприклад число 1 можна представити як: (a + 1 – b), де a = b.

– Поділ змінних. Змінні фіксованого діапазону можуть бути розділені на дві й більше змінних. Для цього змінну "V" що має тип "x" розділяють на "k" змінних "v1,...,vk" типу "y" тобто "V == v1,...,vk". Потім створюється набір функцій що дозволяють витягати змінну типу "x" зі змінних типу "y" і записувати змінну типу "x" у змінні типу "y".

– Зміна подання (або кодування).

Обфускація з'єднання. Один з важливих етапів, у процесі реверсивної інженерії програм, заснований на вивченні структур даних. Тому важливо постаратися, у процесі безпечної розробки при використанні технології Agile з застосуванням обфускації, ускладнити подання використовуваних програмою структур даних. Наприклад, при використанні безпечної розробки при використанні технології Agile з застосуванням обфускації з'єднання це досягається завдяки з'єднанню незалежних даних, або поділу залежних. Нижче наведені основні методи, що дозволяють здійснити таку обфускацію:

– Об'єднання змінних. Дві або більше змінних "v1,...,vk" можуть бути об'єднані в один змінну "V", якщо їхній загальний розмір ("v1,...,vk") не перевищує розмір змінної "V". Наприклад, розглянемо простий приклад об'єднання двох коротких цілочисельних змінних "X","Y" (розміром 16 біт) в одну цілочисельну змінну "Z" (розміром 32 біта). Для цього скористаємося формулою: Z(X,Y) = 2^16 * Y + X, що дозволить, зневажаючи додаванням, визначати значення Y, тобто нехай X = 12, Y = 4 => Z = 65536 * 4 + 12 = 262156, тепер знаючи "Z" для знаходження "Y" можна 262156 / 65536 = 4.000183105 або приблизно 4. При здійсненні арифметичних операцій над значеннями змінних "X", "Y" зберігаються в "Z" потрібно враховувати вище наведену формулу.

– Реструктурування масивів, полягає в заплутуванні структури масивів, шляхом поділу одного масиву на декілька підмасивів, об'єднання декількох масивів в один, згортання масиву (збільшуючи його розмірність) і навпаки, розвертання (зменшуючи його розмірність). Наприклад, один масив "@A" можна розділити на декілька підмасивів "@A1, @A2", при цьому один масив "@A1" буде містити парні позиції елементів, а другий "@A2" непарні позиції елементів масиву "@A". Під згортанням масиву розуміється створення з одномірного масиву, двовимірного. Наприклад, одномірний масив "A" з попереднього приклада, що має розмір 5 можна замінити двовимірним масивом "B" розміром 2.

– Зміна ієрархій спадкування класів, здійснюється шляхом ускладнення ієрархії спадкування за допомогою створення додаткових класів або використання помилкового поділу класів.

Обфускація переупорядкування. Полягає в зміні послідовності оголошення змінних, внутрішнього розташування сховищ даних, а також переупорядкуванні методів, масивів (використання нетривіального подання багатомірних масивів), певних полів у структурах і т.д.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3. Основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

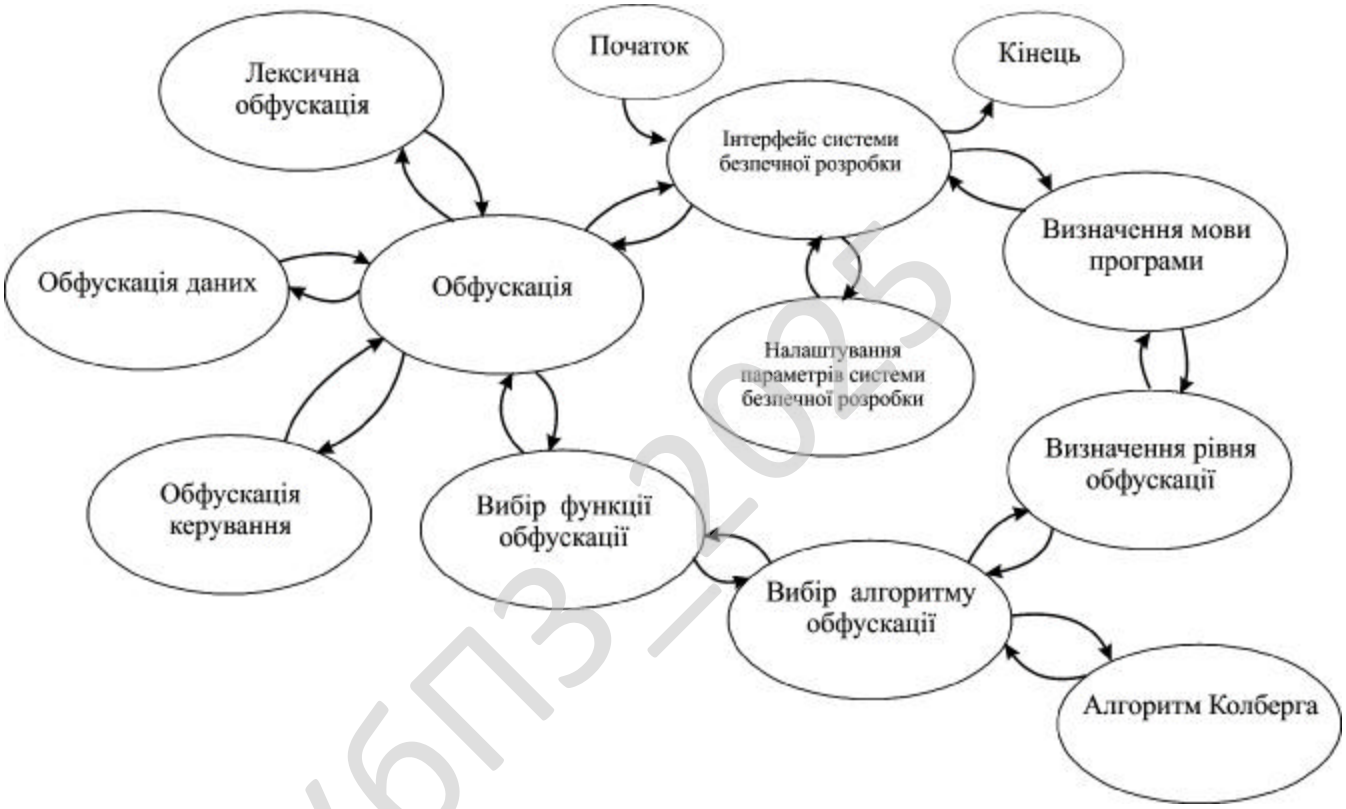

Рисунок 3.3 – Діаграма взаємодії процесів

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграми потоків даних містять чотири типи елементів:

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо реалізацію бакалаврської дипломної роботи. Були проведені розрахунки і підібрані набори тестових даних для перевірки правильності реалізації проектних рішень. Блок-схеми показують весь процес роботи системи з підсистемами та частково доказують правильність вибраних проектних рішень. Тому від точності і детальної блок-схеми залежить результат всієї програми. При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає високого рівня декомпозиції задач на класи.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми. З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підсистеми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання. Останнім кроком безпечної розробки при використанні технології Agile з застосуванням обфускації даних є зміна ієрархій спадкування класів. При виборі користувачем безпечної розробки при використанні технології Agile з застосуванням обфускації керування відбувається виконання наступних послідовних кроків реалізації програмного продукту.

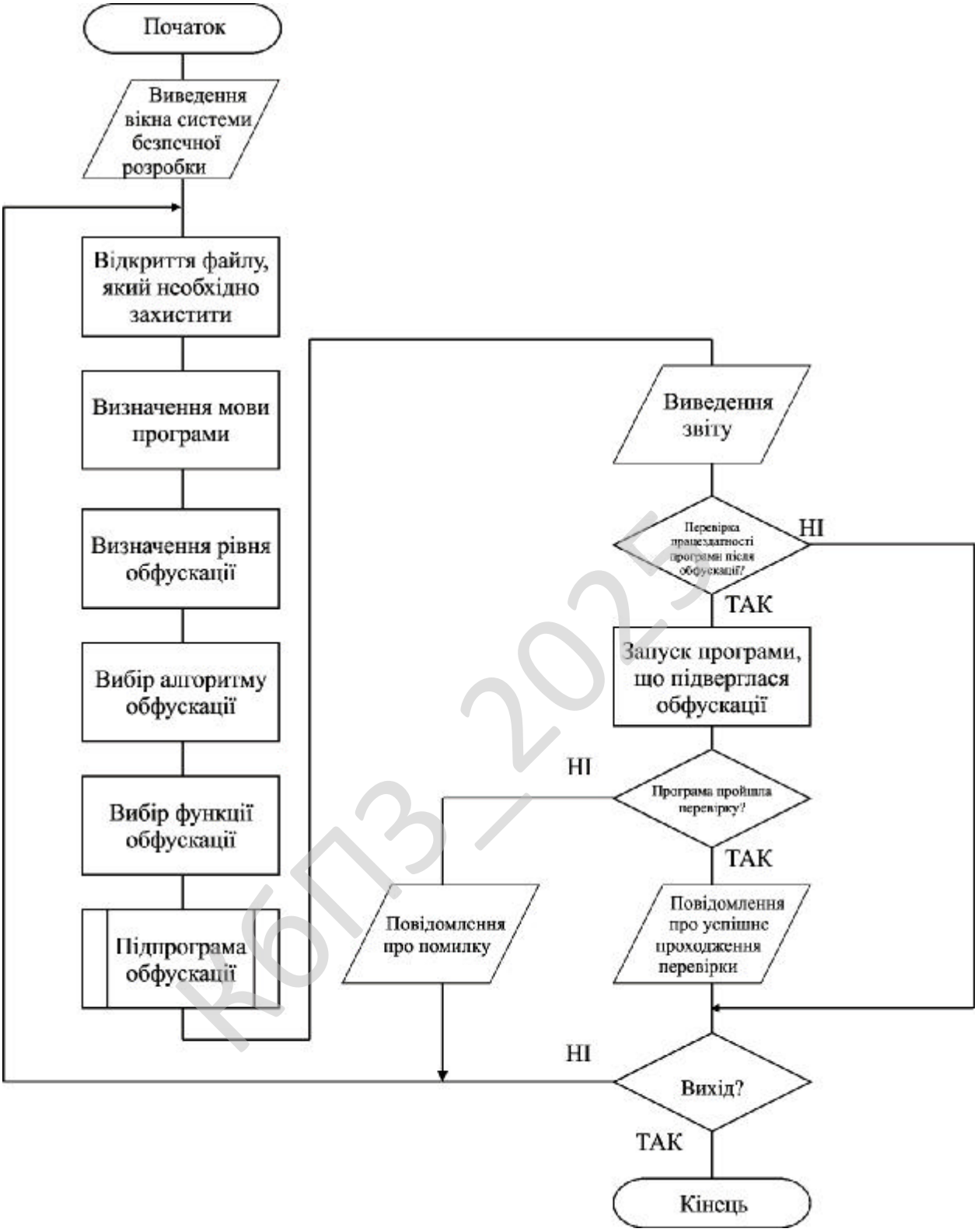

Рисунок 4.1 – Блок-схема основної програми

Спершу відбувається розширення умов циклу. Після цього відбувається усунення бібліотечних викликів. Наступним шагом є додавання надлишкових

операцій. За додаванням надлишкових операцій відбувається розпаралелювання коду. Після виконання розпаралелювання коду відбувається вбудовування та додавання функцій.

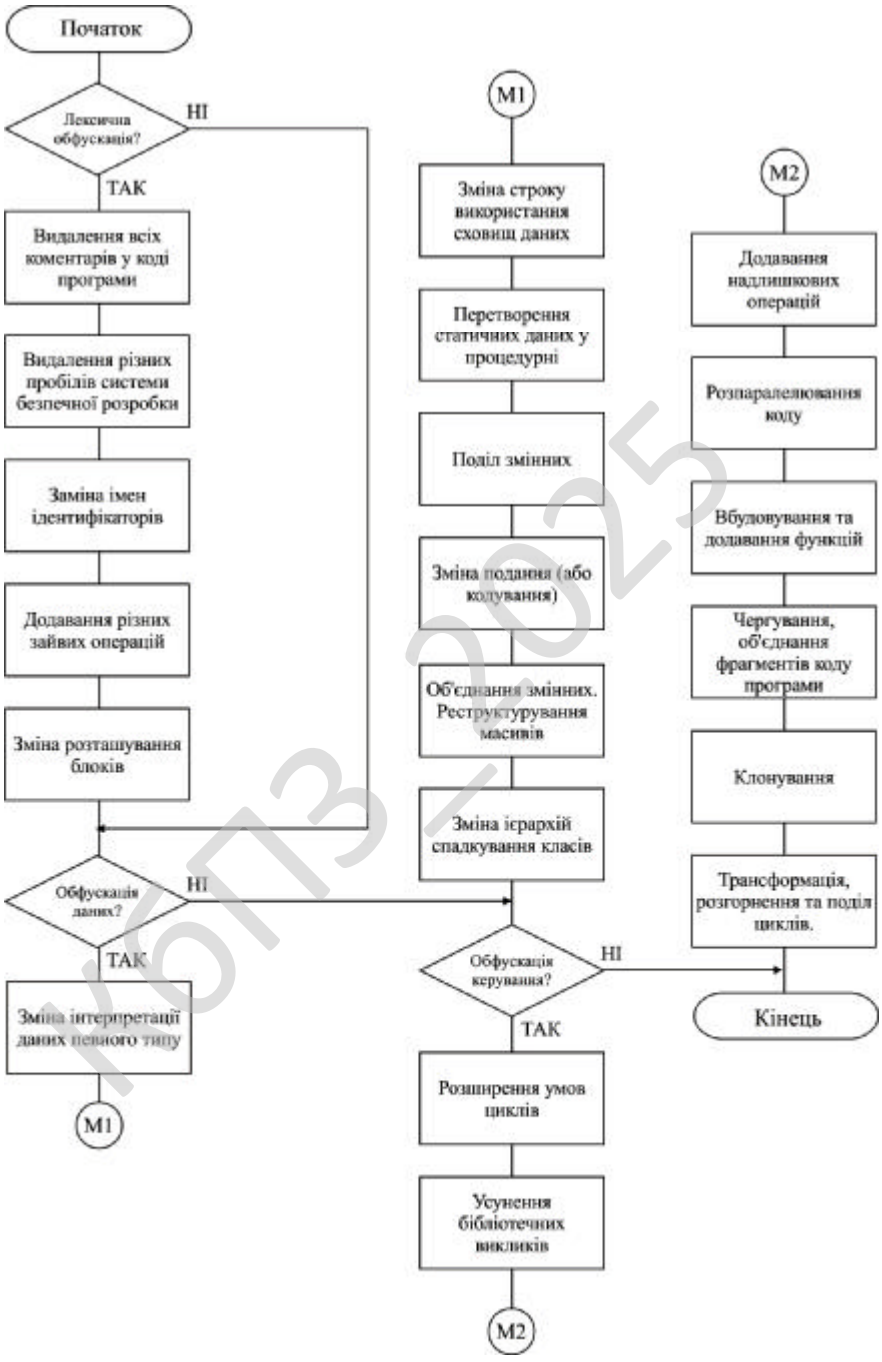

Рисунок 4.2 – Блок-схема роботи підпрограми

Наступним кроком є встановлення чергування та об’єднання фрагментів коду програми. Після цього відбувається клонування цих блоків.

Останнім кроком безпечної розробки при використанні технології Agile з застосуванням обфускації керування є трансформація, розгорнення та поділ циклів.

Розглянувши та описавши блок-схеми основної програми та підпрограми, яка реалізує процедуру безпечної розробки при використанні технології Agile з застосуванням обфускації, перейдемо до розгляду практичної реалізації безпечної розробки при використанні технології Agile з застосуванням обфускації.

Існують різні типи обфускаторів: одні займаються інтерпретуємими мовами типу Perl або PHP і «кореблять» вихідні тексти (видаляють коментарі, дають змінним безглузді імена, шифрують строкові константи й т.д.), інші «перемелюють» байт-код віртуальних машин Java і .NET, що технічно зробити набагато сутужніше. Більше розвинені обфускатори вламуються безпосередньо в машинний код, «розбавляючи» його сміттєвими інструкціями й виконуючи цілий ряд структурних (рідше математичних) перетворень, що змінюють програму до невпізнанності.

Фактично, це поліморфні генератори. Проблема в тому, що поліморфний генератор може за лічені секунди згенерувати хоч мільярд безглуздих команд, перемішавши їх з декількома кілобайтами корисного коду, що дозволяють сучасні процесори й жорсткі диски. Нехай навіть із втратою ефективності, але всім вже давно наплювати на ефективність.

Дизассемблери ще не навчилися видаляти «сміття» в автоматичному режимі, а проаналізувати мегабайти коду вручну нереально. Потрібні передові методики реконструкції потоку керування, що розплавляють «засмічений» код і поділяючі його на «корисні» і «марні» фракції. Їх немає навіть на рівні «теоретичного розуміння». Хоча деякі ідеї на цей рахунок є (наприклад накладення маршруту трасування на графи залежностей за даними), до практичної реалізації ще далеко.

Обфускація здійснює протидію зловмисникам, перешкоджаючи реконструкції алгоритмів і швидкому взлому захистів, але ці проблеми мерхнуть перед ситуацією в антивірусній індустрії. Щоб зламати програму, аналізувати її алгоритм у загальному-то необов'язково. Зате виявити шкідливий код (він же malware) без цього вже не вдасться!

У найпростішому випадку поліморфний генератор просто «накачує» програму купою нічого не значущих команд типу nop, xchg reg,reg, що ніколи не виконуються переходами типу xor reg,reg/jnz junk, де xor – значима команда, а junk – «мертвий код».

– or ch, ch ; «сміття», що не впливає на регістр ch, але, що впливає на регістр прапорів, однак цей вплив перекривається наступним xor;

– xor eax,eax ; потенційно значима команда;

– seto bl ; «сміття», що встановлює bl в 1, якщо є переповнення, а після xor його завжди немає;

– repne jnz short loc_43409A ; «сміття», що передає керування, якщо не нуль, але після xor прапор нуля завжди встановлений, плюс безглуздий префікс repne;

– rep jnp short loc_43408D ; «сміття», що передає керування, якщо непара, але після xor прапор парності завжди встановлений;

– jo short loc_434094 ; «сміття», що передає керування, якщо прапор переповнення встановлений, а він скинутий xor;

– xchg ebx,ebx ; «сміття», обмін регістрів ebx місцями.

От приклад вихідного методу (перед обфускацією):

Код:

```
private void CalcPayroll(SpecialList employeeGroup)
{
   while(employeeGroup.HasMore())
   {
     employee = employeeGroup.GetNext(true);
     employee.UpdateSalary(); DistributeCheck(employee);
   }
```

```
}
```

А от результат (після):

Код:

```
private void _1(_1 _2)
{
  while(_2._1())
  {
    _1 = _2._1(true);
    _1._1();
    _1(_1);
   }
}
```

Спробуйте зрозумійте зміст алгоритму після безпечної розробки при використанні технології Agile з застосуванням обфускації. На це піде час – треба довідатися, що за класи беруть участь, спробувати зрозуміти їхнє призначення, у загальному має бути велика праця.

У цьому завдання безпечної розробки при використанні технології Agile з застосуванням обфускації – утруднити для розуміння вихідний код, заплутати й усунути логічні зв'язки в коді.

Плюси:

– Обфускатори роблять дизассембльований код важким для вивчення, перетворюючи IsLicensed() в x().

– Деякі обфускатори використовують баги ILDASM для захисту від дизассемблінгу в ньому (Salamander).

– Деякі обфускатори навіть конвертують код в native код, роблячи марним дизассемблінг (Salamander).

– Деякі обфускатори шифрують і пакують ваш exe і referenced збірки в один exe-файл, так що розмір програми може зменшитися 2-4 рази й не піддається дизассемблінгу (Thinstall).

Мінуси:

– Продукт залишається дизассембльованим.

– Зібрати складання після дизассемблінгу не буде важко.

– IL код – доступний для читання й розуміння в порівнянні с ассемблерним.

– «Захист» обфускаторів, які використовують баги ILDASM будуть неспроможні перед дизассемблерами інших розроблювачів.

Більш складні обфускатори «перемішують» код, закручуючи потік керування в заплутану спіраль умовних/безумовних переходів, що використовують техніку «перекриття» команд. Деякі байти належать відразу двом, а в деяких випадках і трьом (!) машинним інструкціям, що «засліплює» дизассемблери, змушуючи їх генерувати неповний і неправильний лістинг.

Однак в інтерактивному режимі все-таки можна дизасемблювати код, але дуже вже утомливо. Краще скористатися трассером, що генерує лістинг реально виконуваних машинних команд. Заодно позбуваємося від частини сміття й «мертвого» коду.

Фрагмент лістингу, сгенерированный розробленою програмою. Демонстрація техніки «перекриття» машинних команд, використовуваної обфускаторами

```
.adata:0043400E loc_43400E: ; CODE XREF: .adata:00434023j
.adata:0043400E ; .adata:loc_43401A j
.adata:0043400E mov eax, 0EBB907EBh
.adata:00434013
.adata:00434013 loc_434013: ; CODE XREF: .adata:loc_43401Dj
.adata:00434013 seto bl ; стрибок у середину команди
.adata:00434016 or ch, bh
.adata:00434018 jmp short loc_434025
.adata:00434018
.adata:0043401A loc_43401A: ; CODE XREF: .adata:00434009j
.adata:0043401A repne jmp short near ptr loc_43400E+4
.adata:0043401D
.adata:0043401D loc_43401D: ; CODE XREF: .adata:loc_43400Cj
.adata:0043401D jmp short near ptr loc_434013+2
```

Зверни увагу на команду «043401Dh:jmp short loc_434013+2», що стрибає за адресою 434013h+2h == 434015h, тобто в середину інструкції 434013h:seto bl. Саме що в середину! З погляду дизассемблера, команда є «атомарною»

структурною одиницею, тобто неподільної. Насправді ж усяка машинна інструкція складається з послідовності байт і може бути виконана з будь-якого місця! У всякому разі, x 86-процесори не вимагають вирівнювання коду. Інакше кажучи, не існує «команд» – існують тільки байти. Якщо почати виконання інструкції не з першого байта, одержимо зовсім іншу команду! Щоб виконати перехід «043401Dh:jmp short loc_434013+2», необхідно підвести курсор до мітки loc_434013 і нажати <U> (так ми «розкришимо» дизассемблерний код на байти), а після перейти за адресою 434015h і нажати <C>, тим самим перетворивши байти в дизассемблерний код.

«Розкриття» накладеної команди

```
.adata:0043400E unk_43400E db 0B8h ; ¬ ; CODE XREF: .adata:loc_434023j
.adata:0043400F db 0EBh ;
.adata:00434010 db 7
.adata:00434011 db 0B9h ; |
.adata:00434012 loc_434012: ; CODE XREF: .adata:loc_43401Aj
.adata:00434012 jmp short loc_434023
.adata:00434014
.adata:00434014 nop
.adata:00434015
.adata:00434015 loc_434015: ; CODE XREF: .adata:loc_43401Dj
.adata:00434015 jmp short loc_43401F ; стрибок сюди
.adata:00434017
.adata:00434017 std
.adata:00434018 jmp short loc_434025
.adata:0043401A
.adata:0043401A loc_43401A: ; CODE XREF: .adata:00434009j
.adata:0043401A repne jmp short loc_434012
```

На місці seto bl виникла пари інструкцій jmp loc_43401F/std. Який із двох лістингів правильний? Окремо – ні той, ні інший. Вони стають «правильними» тільки вдвох! Однак удержати ці подробиці в голові нереально. Залишається заганяти «альтернативний» лістинг у коментарі. Якщо та сама машинна команда має три й більше «точки входу», то коментарі вже не рятують і виникає плутанина, що змушує використовувати замість дизассемблера трассер.

Витончені обфускатори відслідковують залежності за даними, впроваджуючи осмислені інструкції з «нульовим ефектом». Пояснимо на конкретному прикладі. Допустимо, обфускатору зустрілася конструкція:

Оригінальний код до безпечної розробки при використанні технології Agile з застосуванням обфускації.

```
PUSH EAX ; останнє звертання до eax
MOV EAX,EBX ; реинициализация eax
```

Легко показати, що між останнім звертанням до eax і його реініціалізацією можна як завгодно модифікувати регістр eax без шкоди для виконання програми, оскільки будь-які операції присвоєння однаково будуть перекриті командою mov eax,ebx.

Код після безпечної розробки при використанні технології Agile з застосуванням обфускації

```
push eax ; останнє значиме звертання до eax
xor eax,eax ; сміття
l1:
inc eax ; сміття
jz l2 ; сміття
cmp eax, ebx ; сміття
jnz l1 ; сміття
cmp eax, ecx ; сміття
jge l1 ; сміття
l2:
sub eax, 666h ; сміття
shl eax, 1 ; сміття
mov eax, ebx ; значима реініціалізація eax
```

Також обфускатори можуть тимчасово зберігати регістр на стеці, а потім, позмінюючи його значення, відновлювати колишнє значення.

Тимчасове збереження регістрів на стеці з наступним відновленням:

```
001B:0043402C 50 PUSH EAX ; зберігаємо eax
001B:0043402D 51 PUSH ECX ; зберігаємо ecx
001B:0043402E EB0F JMP 0043403F
001B:0043403F F2EBF5 REPNZ JMP 00434037
001B:00434037 EB0F JMP 00434048
001B:00434048 EBE9 JMP 00434033
001B:00434033 B8EB07B9EB MOV EAX,EBB907EB ; «змінюємо» в eax
```

```
001B:0043403B 08FD OR CH,BH ; «змінюємо» в ch
001B:0043403D EB0B JMP 0043404A
001B:0043404A F3EBE4 REPZ JMP 00434031
001B:00434031 EB0F JMP 00434042
001B:00434042 EBF6 JMP 0043403A
001B:0043403A EB08 JMP 00434044
001B:00434044 F2EB08 REPNZ JMP 0043404F
001B:0043404F 59 POP ECX ; відновлюємо ecx
001B:00434050 58 POP EAX ; відновлюємо eax
```

Команда MOV EAX,EBB907EBh на перший погляд виглядає «значимою», але насправді це «сміття», нейтралізуємо командами push eax/pop eax. По суті, весь цей конгломерат робить нульовий ефект, тобто є зовсім бездіяльним кодом. Так що робити вивід про «значимість» команд потрібно з дуже великою обережністю. Поки не буде доведено, що даний шматок коду дійсно має якийсь ефект, він повинен уважатися «сміттям» за замовчуванням.

Деякі обфускатори люблять впроваджувати підроблені розшифровувачі, які розшифровують і відразу зашифровують довільні фрагменти пам'яті.

«Підроблений» розшифровувач, впроваджений обфускатором

```
00434105 83ED 06 SUB EBP,6
00434108 B8 3B010000 MOV EAX,13B
0043410D 03C5 ADD EAX,EBP
0043410F 33DB XOR EBX,EBX
00434111 81C3 01010101 ADD EBX,1010101
00434117 3118 XOR DWORD PTR DS:[EAX],EBX ; розшифровуємо
00434119 8138 78540000 CMP DWORD PTR DS:[EAX],5478
0043411F 74 04 JE SHORT app_test.00434125
00434121 3118 XOR DWORD PTR DS:[EAX],EBX ; зашифровуємо
00434123 ^EB EC JMP SHORT app_test.00434111
```

Зрозуміло, всі ці дії вносять побічні ефекти (як мінімум, впливають на прапори), і обфускатору доводиться виконувати безліч додаткових перевірок, щоб переконатися, що ці побічні дії не зроблять фатального впливу на програмку, що захищається. Розробка якісного й надійного обфускатора – складне інженерне завдання, але витрачений час коштує того. Марність «інструкцій з нульовим ефектом» уже не розпізнається візуально, і звичайний трассер отут нічим не допоможе. Необхідно трасувати не тільки потік керування, але й потік даних,

тобто відслідковувати реальні зміни значень регістрів/комірок пам'яті, для чого звичайно використовуються графи. Як тільки граф замикається сам на себе, всі «зайві» операції над даними віддаляються й залишається тільки суть.

Існують різні способи аналізу алгоритмів роботи пристроїв, «схема» яким недоступна. «Заплутану» програму можна розглядати як «чорний ящик» із входом і виходом, абстрагуючись від машинного коду й виконуючи аналіз на набагато більше високому рівні.

Багато інформації несуть у собі виклики API-функцій (разом з аргументами й повертаються значеннями, що). Якщо зловмиснику вдасться перехопити й бібліотечні функції разом з RTL, то картина того, що відбувається, в загальних рисах намалюється. Принаймні, зловмисник зможе з'ясувати, до чого «прив’язується» захист, і в такий спосіб він довідається про закінчення іспитового періоду. Часто для взлому більшого не потрібно.

Замість того щоб аналізувати код самої програми, зловмисник досліджує, яким образом вона взаємодіє з «зовнішнім миром», тобто з ОС. Тоді на «внутрішній» мир захисту можна буде забити. Звичайно, не для всіх програм це спрацьовує, але багато хто ламаються саме так.

Незважаючи на те що я працював над ПЗ один в реалізації програми я використовував підходи пришвидшення розробки на основі методологій Agile.

Гнучка́ розробка програмного забезпечення (Agile software development, agile-методи) – клас методологій розробки програмного забезпечення, що базується на ітеративній розробці, в якій вимоги та розв'язки еволюціонують через співпрацю між самоорганізовуваними багатофункціональними командами.

Гнучка розробка – найкращий засіб для підвищення продуктивності розробників програмного забезпечення.

Більшість гнучких методологій націлені на мінімізацію ризиків, шляхом зведення розробки до серії коротких циклів, що мають назву ітерацій, які зазвичай тривають один-два тижні. Кожна ітерація сама по собі виглядає як програмний проект в мініатюрі, і включає всі завдання, необхідні для видачі мінімального

приросту за функціональністю: планування, аналіз вимог, проектування, кодування, тестування і документування. Хоча окрема ітерація, як правило, недостатня для випуску нової версії продукту, мається на увазі те, що гнучкий програмний проект готовий до випуску наприкінці кожної ітерації. Після закінчення кожної ітерації, команда виконує переоцінку пріоритетів розробки.

Agile акцентує увагу на безпосередньому спілкуванні «віч-на-віч». Більшість agile команд розташовані в одному офісі, його іноді називають bullpen. Як мінімум вона включає і «замовників» (замовники, які визначають продукт, також це можуть бути менеджери продукту, бізнес аналітики або клієнти). Офіс може також включати тестувальників, дизайнерів інтерфейсу, технічних авторів і менеджерів.

Основною метрикою agile методів є робочий продукт. Віддаючи перевагу безпосередньому спілкуванню, agile-методи зменшують обсяг письмової документації в порівнянні з іншими методами. Це привело до критики цих методів як недисциплінованих.

Agile – родина процесів розробки, а не єдиний підхід в розробці програмного забезпечення, і визначається Agile Manifesto. Agile не включає практик, а визначає цінності та принципи, якими керуються успішні команди.

Agile Manifesto розроблений і прийнятий 17 розробниками 11-13 лютого 2001 року на лижному курорті The Lodge at Snowbird в горах Юти. Маніфест підписали представники наступних методологій Extreme programming, Scrum, DSDM, Adaptive software development, Crystal Clear, Feature driven development, Pragmatic Programming. Agile Manifesto містить 4 основні ідеї та 12 принципів. Примітно, що Agile Manifesto не містить практичних порад.

Основні ідеї:

– Особистості та їхні взаємодії важливіші, ніж процеси та інструменти;

– Робоче програмне забезпечення важливіше, ніж повна документація;

– Співпраця із замовником важливіша, ніж контрактні зобов'язання;

– Реакція на зміни важливіша, ніж дотримання плану.

Принципи, які роз'яснює Agile Manifesto:

– задоволення клієнта за рахунок ранньої та безперебійної поставки коштовного програмного забезпечення;

– вітання змін вимог навіть наприкінці розробки (це може підвищити конкурентоспроможність отриманого продукту);

– часта поставка робочого програмного забезпечення (кожен місяць або тиждень або ще частіше);

– тісне, щоденне спілкування замовника з розробниками впродовж всього проекту;

– проектом займаються мотивовані особистості, які забезпечені потрібними умовами роботи, підтримкою і довірою;

– рекомендований метод передачі інформації – особиста розмова (віч-на-віч);

– робоче програмне забезпечення – найкращий вимірювач прогресу;

– спонсори, розробники та користувачі повинні мати можливість підтримувати постійний темп на невизначений термін;

– постійну увагу поліпшенню технічної майстерності та зручному дизайну;

– простота – мистецтво не робити зайвої роботи;

– найкращі технічні вимоги, дизайн та архітектура виходять у самоорганізованої команди;

– постійна адаптація до мінливих обставин.

Маніфест та Принципи гнучкої розробки містять високорівневі ідеї щодо того, як потрібно вибудовувати процес розробки програмного забезпечення, щоб успішно завершувати проекти й створювати команди, в яких приємно та цікаво працювати.

Документи визначають, що потрібно для цього зробити, але не говорять, як це зробити. По-іншому й не могло бути, оскільки Маніфест та Принципи народилися внаслідок консенсусу представників різних (хоча й споріднених)

напрямів, які могли знайти спільну основу лише на рівні базових цінностей та принципів.

Критика. Багато керівників проектів, що працюють у традиційних методологіях на кшталт «водоспаду», критикують agile-методи.

Один з повторюваних пунктів критики: при agile-підході часто нехтують створенням «дорожньої карти» розвитку продукту, так само як і управлінням вимогами, в процесі якого і формується така «карта». Гнучкий підхід до управління вимогами не має на увазі далекосяжних планів (по суті, управління вимогами просто не існує в даній методології), а має на увазі можливість замовника раптом і несподівано наприкінці кожної ітерації виставляти нові вимоги, що часто суперечать архітектурі вже створеного і поставленого продукту. Таке іноді призводить до катастрофічних «авралів» з масовим рефакторингом і переробками практично на кожній черговій ітерації.

Крім того вважається, що робота в agile мотивує розробників вирішувати всі прибулі завдання найпростішим і найшвидшим можливим способом, при цьому часто не звертаючи уваги на коректність коду з точки зору вимог базової платформи (підхід «працює, та й добре», при цьому не враховується, що може перестати працювати при найменшій зміні або ж породити важкі до відтворення дефекти після реального розгортання у клієнта). Це призводить до зниження якості продукту і накопиченню дефектів.

Методології. Існують методології, які дотримуються цінностей і принципів заявлених в Agile Manifesto, деякі з них:

1. Agile Modeling – набір понять, принципів і прийомів (практик), що дозволяють швидко і просто виконувати моделювання і документування в проектах розробки програмного забезпечення. Не включає в себе детальну інструкцію з проектування, не містить описів, як будувати діаграми на UML.

Основна мета – ефективне моделювання і документування; але не охоплює програмування та тестування, не включає питання управління проектом, розгортання і супроводу системи. Однак включає в себе перевірку моделі кодом.

2. Agile Unified Process (AUP) спрощена версія IBM Rational Unified Process (RUP), розроблена Скоттом Амблером, яка описує просте і зрозуміле наближення (модель) для створення програмного забезпечення для бізнес-додатків.

3 Agile Data Method – група ітеративних методів розробки програмного забезпечення, в яких вимоги та рішення досягаються в рамках співпраці різних крос-функціональних команд.

4. DSDM заснований на концепції швидкої розробки додатків (Rapid Application Development, RAD). Являє собою ітеративний і інкрементний підхід, який надає особливого значення тривалій участі в процесі користувача/споживача.

5. Essential Unified Process (EssUP).

6. Екстремальне програмування (Extreme programming, XP).

7. Feature driven development (FDD) – функціонально-орієнтована розробка. Використовуване в FDD поняття функції або властивості (feature) Системи досить близько до поняття прецеденту використання, використовуваному в RUP, істотна відмінність – це додаткове обмеження: «кожна функція повинна допускати реалізацію не більше, ніж за два тижні». Тобто якщо сценарій використання досить малий, його можна вважати функцією. Якщо ж великий, то його треба розбити на декілька відносно незалежних функцій.

8. Getting Real – ітераційний підхід без функціональних специфікацій, що використовується для веб-додатків. У даному методі спершу розробляється інтерфейс програми, а потім її функціональна частина.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм MMB, в основі якого лежить змішування операцій різних алгебраїчних груп. MMB – ітеративний алгоритм, що складається з

лінійних дій (XOR і використання ключа) і паралельного застосування чотирьох великих оборотних нелінійних підстановок. Ці підстановки визначаються за допомогою множення по модулю 2^{32}-1 з постійними множниками. У підсумку з'являється алгоритм, що використовує 128-бітовий ключ і 128-бітовий блок.

Алгоритм MMB оперує 32-бітовими підблоками тексту (x_0, x_1, x_2, x_3) і 32-бітовими підблоками ключу (k_0, k_1, k_2, k_3). Це спрощує реалізацію алгоритму на сучасних 64-бітових процесорах. Чергуючись із операцією XOR, шість разів використовується нелінійна функція *f*. Запишемо операції алгоритму (всі операції з індексами виконуються по модулю 4):

$$x_i = x_i \oplus k_i \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+1} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+2} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_i \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+1} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+2} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

Функція *f* виконується в три кроки:

1. $x_i = c_i * x_i$ для $i = 0..3$ (Якщо на вході множення одні одиниці, то на виході – теж одні одиниці).

2. Якщо молодший значущий біт $x_0 = 1$, то $x_0 = x_0 \oplus C$. Якщо молодший значущий байт $x_3 = 0$, то $x_3 = x_3 \oplus C$.

3. $x_i = x_{i-1} \oplus x_i \oplus x_{i+1}$ для $i = 0..3$.

Всі операції з індексами виконуються по модулю 4. Операція множення на кроці 1 виконується по модулі 2^{32}-1. Спеціальний випадок для даного алгоритму:

якщо другий операнд дорівнює 2^{32}-1, результат теж дорівнює 2^{32}-1. В алгоритмі використовуються наступні константи:

$$C = 2aaaaaaa,\ c_0 = 025f1cdb,\ c_1 = 2*c_0,\ c_2 = 2^3 * c_0,\ c_3 = 2^7 * c_0.$$

Константа C – «найпростіша» константа без кругової симетрії, високою трійковою вагою й нульовим молодшим значущим бітом. У константи c_0 є інші особливі характеристики. Константи c_1, c_2 і c_3 – зрушені версії c_0, і служать для запобігання атак, заснованих на симетрії.

Розшифрування виконується у зворотному порядку, Етапи 2 і 3 інверсні їм самим. На етапі 1 замість c_i використовується c_i^{-1}. Значення c_0^{-1} = 0dad4694.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено розроблене у бакалаврської дипломної роботі програмне забезпечення системи безпечної розробки при використанні технології Agile. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок ПЗ.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Верхнього меню: Проект; Параметри; Довідка.

– Розділу обрання групи тестування обфускації.

– Розділу виведення результату роботи системи.

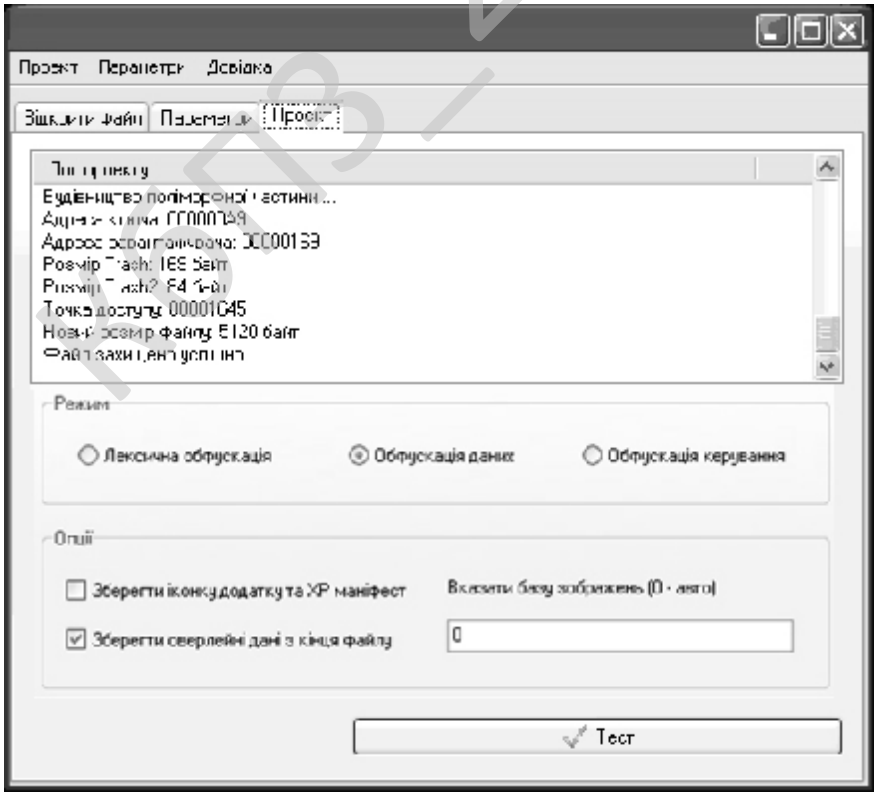

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

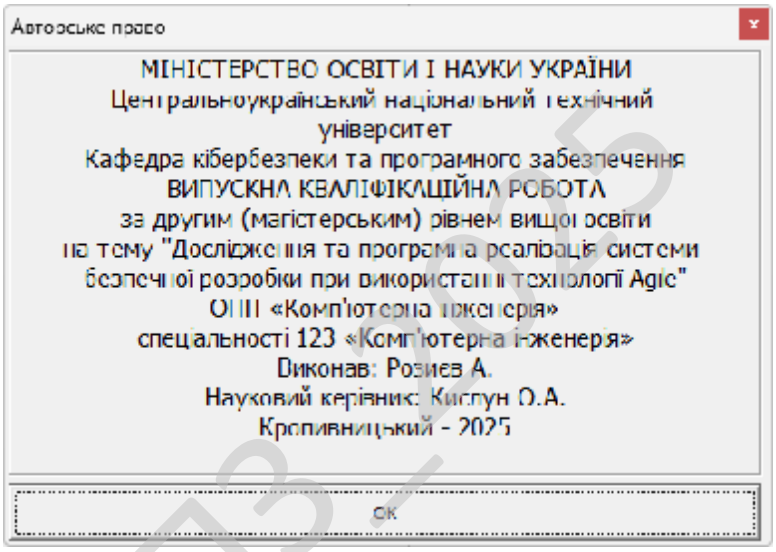

Рисунок 5.2 – Авторське право

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є

унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати IT рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене IT рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються

в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування чорної скриньки.

Основне місце програми тестів «чорної скриньки» — інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;

– Помилок характеристик (необхідна ємність пам'яті і т.д.);

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи безпечної розробки при використанні технології Agile.

Метою розробки є дослідження та програмна реалізація системи безпечної розробки при використанні технології Agile.

Об'єктом дослідження є процес безпечної розробки при використанні технології Agile.

Предметом дослідження є методи безпечної розробки при використанні технології Agile.

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

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод безпечної розробки при використанні технології Agile.

– Розроблено вітчизняний продукт безпечної розробки при використанні технології Agile, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та розробки системи безпечної розробки при використанні технології Agile можуть бути насамперед корисними для підприємств, які мають розгалужену ІТ-інфраструктуру й використовують сервери, мережеве обладнання та корпоративні сервіси для підтримки своєї діяльності. Для таких компаній стабільність і безперебійність роботи інформаційних систем є критично важливими, тому можливість своєчасного виявлення несправностей або перевантажень стає суттєвою конкурентною перевагою. Саме система моніторингу допомагає контролювати роботу мережевих пристроїв у режимі реального часу, виявляючи проблеми ще до того, як вони вплинуть на користувачів.

Особливий інтерес до таких систем можуть проявити ІТ-компанії, які займаються наданням послуг хостингу, розробкою програмного забезпечення або підтримкою клієнтів. Для них швидкість реагування на інциденти та якість технічного обслуговування є показниками репутації, а отже, від роботи системи моніторингу залежить рівень довіри клієнтів і лояльність користувачів. Такі підприємства часто працюють у середовищі, де навіть хвилинна затримка чи зупинка сервера призводить до фінансових збитків, тому автоматизація контролю за станом мережі – це не розкіш, а необхідність.

Крім комерційних компаній, результати дослідження будуть актуальними для державних структур, освітніх установ і організацій, які мають внутрішні мережі та зберігають великі обсяги інформації. У таких установах впровадження системи моніторингу підвищує ефективність роботи ІТ-відділів, зменшує ризик втрати даних і допомагає раціонально використовувати наявні ресурси.

Не менш важливим є значення цієї розробки для навчальних і наукових закладів. Вони можуть використовувати систему як навчальну платформу для підготовки фахівців у сфері інформаційних технологій. Студенти отримають можливість не лише спостерігати за реальною роботою системи моніторингу, а й аналізувати дані, моделювати різні ситуації та вчитися реагувати на інциденти. Таким чином, результати дослідження мають універсальний характер і можуть бути впроваджені як у бізнесі, так і в освіті.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості програмного продукту було проведено експертне опитування серед фахівців у галузі IT-інфраструктури, адміністраторів систем і представників компаній, що мають досвід використання схожих рішень. Експертам було запропоновано оцінити систему за основними критеріями – функціональні можливості, надійність, простота впровадження, масштабованість, вартість експлуатації та потенційна економічна ефективність.

Більшість експертів високо оцінили саме інтелектуальну частину системи – можливість автоматичного сповіщення про інциденти, генерацію аналітичних звітів і прогнозування потенційних відмов обладнання. Особливо було відзначено, що система працює стабільно навіть при великому навантаженні й може адаптуватися до різних типів мережевої інфраструктури, що робить її універсальною.

За результатами оцінки середній рівень привабливості продукту склав 8,7 бала з 10 можливих. Експерти зазначили, що така система може мати великий попит серед середніх і великих підприємств, особливо якщо її вартість залишатиметься конкурентною. Також було підкреслено, що простота інтерфейсу та можливість кастомізації під конкретного користувача є суттєвими перевагами, які підвищують комерційний потенціал рішення.

Таким чином, метод експертних оцінок показав, що система має високу ринкову привабливість, відповідає актуальним потребам бізнесу та може стати успішним продуктом за умови належного маркетингового просування та підтримки користувачів.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості розробки системи безпечної розробки при використанні технології Agile доцільно використовувати витратний метод. Він передбачає визначення всіх фактичних витрат, які були понесені під час створення програмного продукту, включаючи оплату праці розробників, витрати на апаратне забезпечення, ліцензії, тестування та впровадження. Такий підхід дозволяє точно визначити базову собівартість проєкту, що є особливо важливим для невеликих команд і стартапів.

Однак, у випадку комерційного впровадження, доцільно поєднати цей підхід із дохідним методом. Дохідний метод дає змогу оцінити майбутні вигоди, які підприємство отримає після впровадження системи. Наприклад, скорочення простоїв серверів, підвищення ефективності роботи персоналу та зменшення витрат на ручну діагностику мережі є прямими джерелами економічної вигоди.

Такий комбінований підхід дозволяє не лише визначити початкову вартість розробки, а й обґрунтувати економічну доцільність проєкту. Він допомагає потенційним інвесторам побачити не просто витрати, а реальні фінансові перспективи, які відкриває впровадження системи.

У результаті використання комбінованої моделі оцінки можна отримати повну картину вартості та окупності проєкту, що стане основою для прийняття управлінських рішень щодо його реалізації чи масштабування.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Компанія має розгалужену IT-інфраструктуру, яка включає сервери, мережеве обладнання, робочі станції, системи зберігання даних і корпоративні сервіси. До впровадження системи моніторингу контроль за станом мережі здійснювався вручну: адміністратори виявляли проблеми лише після звернень користувачів або повного виходу сервісів із ладу. Це призводило до простоїв, затримок у роботі та фінансових втрат. Основна мета впровадження системи мережевого моніторингу – забезпечити цілодобове автоматичне відстеження стану обладнання, серверів і додатків, оперативне реагування на інциденти, зниження кількості простоїв і запобігання критичним збоям у роботі IT-інфраструктури. Вхідні дані зафіксовано в таблиці 7.1.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	До впровадження	Після впровадження	Економічний ефект
Кількість простоїв серверів на рік	20 випадків	5 випадків	–15
Середня тривалість простою одного сервера	4 години	1 година	–3 години
Середні втрати підприємства за 1 годину простою	25 000 грн	5 000 грн	–20 000 грн
Витрати на ручну діагностику й усунення збоїв	300 000 грн/рік	150 000 грн/рік	–150 000 грн
Вартість впровадження системи моніторингу	—	—	450 000 грн
Річні витрати на підтримку системи	—	—	100 000 грн

Розрахунок економічного ефекту демонструє наступне: зменшення збитків від простоїв – 1 975 000 грн/рік, економія на технічному обслуговуванні – 150 000 грн/рік, сукупний річний ефект – 2 125 000 грн/рік, чистий ефект – 2 025 000 грн/рік, термін окупності (Payback Period) – 0,22 року (~2,5 місяці), коефіцієнт ефективності (ROI) – 450%.

Додаткові (немонетарні) переваги: підвищення стабільності IT-інфраструктури завдяки ранньому виявленню збоїв, зменшення навантаження на IT-персонал через автоматизацію моніторингу, покращення SLA (Service Level Agreement) і задоволеності користувачів, прогнозування потенційних проблем через аналітику та звітність у реальному часі, зростання репутації підприємства, адже мінімізуються ризики затримок у наданні послуг або збою критичних бізнес-процесів.

Таким чином, моніторинг стає не лише технічним інструментом, а й важливою складовою операційної надійності та конкурентоспроможності підприємства.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Просування системи моніторингу має будуватися на поетапному підході, що включає як технічну демонстрацію, так і інформаційне просування. На першому етапі варто створити пілотний проєкт і запропонувати його впровадження у невеликій кількості підприємств для збору відгуків і реальних кейсів. Це дозволить перевірити ефективність системи в реальних умовах і створити довіру до продукту.

Далі важливо забезпечити інформаційну присутність продукту – через участь у галузевих конференціях, IT-форумах, онлайн-презентаціях і спеціалізованих публікаціях. Саме через публічну експертну комунікацію формується репутація розробника та усвідомлення цінності рішення на ринку.

Наступним етапом є розширення партнерських зв’язків. Доцільно співпрацювати з ІТ-компаніями, які займаються інтеграцією корпоративних систем, адже вони можуть пропонувати продукт своїм клієнтам як частину комплексного рішення. Водночас слід розробити гнучку цінову політику – наприклад, ліцензування за кількістю пристроїв або модель передплати, що зробить продукт доступнішим для малого та середнього бізнесу.

Просування має супроводжуватися технічною підтримкою користувачів, оновленнями та навчанням персоналу. Це створює позитивний досвід використання продукту та сприяє формуванню довгострокових відносин із клієнтами. У підсумку правильна стратегія просування допоможе не лише збільшити продажі, а й побудувати впізнаваний бренд на ринку ІТ-рішень.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту варто поєднати прямі продажі з цифровими платформами розповсюдження програмного забезпечення. Власний сайт компанії може стати не лише вітриною продукту, а й каналом комунікації з клієнтами, де вони зможуть отримати демо-версію, консультацію або підтримку. Це сприятиме зниженню витрат на маркетинг і збільшенню довіри.

Додатково ефективним буде впровадження партнерської програми для системних інтеграторів і реселерів, які вже мають доступ до корпоративних клієнтів. Така модель дозволяє розширити охоплення ринку без суттєвих додаткових інвестицій. Також можна запропонувати гібридну форму реалізації: ліцензування для великих компаній і модель SaaS (Software as a Service) для малого бізнесу. Це підвищить доступність системи та дозволить гнучко реагувати на потреби різних сегментів ринку.

Ключовим напрямом оптимізації збуту є створення якісного сервісу після продажу – технічна підтримка, регулярні оновлення, аналітичні звіти. Усе це забезпечує стабільність роботи клієнта й стимулює його до подальшої співпраці.

7.7 Визначення ключових факторів успіху конкретного проєкту

Основним фактором успіху є стабільність і надійність системи. Якщо система моніторингу працює без збоїв і забезпечує реальну користь, вона швидко здобуває довіру користувачів. Технологічна якість продукту, його здатність масштабуватися й інтегруватися з іншими ІТ-рішеннями відіграють ключову роль у його життєздатності.

Другим важливим чинником є професійна команда розробників і технічної підтримки. Клієнти цінують не лише продукт, а й можливість отримати швидку допомогу у випадку проблем або питань. Від рівня компетенції фахівців залежить не лише якість обслуговування, а й довгострокові відносини з партнерами.

Не менш значущим є гнучкість системи – можливість адаптувати її під специфіку кожного клієнта. Різні компанії мають різну інфраструктуру, тому універсальне, але налаштовуване рішення стає перевагою.

І, нарешті, успіх будь-якого ІТ-проєкту визначається здатністю постійно вдосконалюватися. Регулярні оновлення, впровадження нових технологій і зворотний зв’язок із користувачами формують довіру й підтримують актуальність продукту на ринку. Саме ці чинники разом створюють основу для стабільного розвитку та комерційного успіху системи моніторингу.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Вимога щодо впровадження заходів з охорони праці передбачається, зокрема, статтею 13 Закону України «Про охорону праці». У відповідності з цим законом, кожна компанія, в рамках якої реалізуються трудові відносини, зобов'язана вжити всіх необхідних заходів з охорони праці та розробити відповідні документи. Правильний підхід до організації охорони праці на виробництві дає працівникам почуття стабільності, захищеності їх прав та інтересів, уваги з боку керівництва. Налагоджена система охорони праці також знижує плинність кадрів, що позитивно впливає на стабільність роботи підприємств.

Аналізуючи умови працівників ІТ-сфери, на перший погляд, може здатися, що працівники сфери інформаційних технологій не схильні до ризиків на виробництві, та якщо більш глибоко розглянути умови і специфіку праці фахівців сфері ІТ-індустрії, можна виявити ряд факторів які будуть мати негативний сплив на стан охорони праці, так і на самого it-фахівця. Сюди можна віднести як невідповідність освітлення, так і високий рівень шуму, що негативно позначатимуться як на емоційному так і на фізичному стані фахівця, призводитимуть до зниження ефективності праці та виробничих травм. Також, важливим моментом охорони праці ІТ-фахівця є врахування його психологічних можливостей (швидкість реакції, особливості пам'яті та уваги, емоційний стан тощо).

Для того, щоб забезпечити ефективну роботу ІТ-фахівця, потрібно враховувати та максимально компенсувати такі негативні фактори як: надмірне нервово-емоційне навантаження, довготривалі статичні перевантаження, обмежена рухова активність. Всі ці чинники призводить до різноманітних

відхилень у стані здоров'я, зокрема до перевтоми, зниження фізичної та розумової працездатності, неврозів, захворювань серцево-судинної системи тощо.

Метою даного розділу є огляд конкретних умов праці спеціаліста у сфері ІТ-індустрії. Завданнями для даного розділу є: аналіз умов праці на робочому місці фахівця it-індустрії, розробка конкретних рекомендацій щодо покращення умов праці фахівців ІТ-індустрії, огляд пожежної безпеки на ІТ-підприємстві та розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ-фахівці.

8.2 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	3
Довжина	4,6
Висота	3

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	6,9
Об'єм, V	$м^3$	не менше 20.0	20,7

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працюють двоє людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»). Таним чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-24	40-55	0,12
Тепла	23-25	50-70	0,1	24-25	50-65	0,9

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер Xerox WorkCentre 3025BI (3025VBI), електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми України "Природне і штучне освітлення" – ДБН В.2.5-28:2018 [1], у яких прописані вимоги до використання всіх освітлювальних приладів, у тому числі світлодіодних.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5-28:2018 [1], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою

контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк [1], Крім того, все поле зору повинно бути освітлено достатньо рівномірно – це основна гігієнічна вимога. Оскільки яскраве світло на ділянці периферийного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.3 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значень;

Таким чином можна припустити, що основною причиною можливого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до

колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.4 Пожежна безпека

Вимоги до пожежної безпеки на підприємстві неухильно повинен дотримуватися кожен співробітник, а організаційна складова при цьому покладається на посадових осіб за відповідним рішенням керівництва і прописується в посадових інструкціях і положеннях по структурним підрозділам.

Зокрема, вказуються конкретні території, ділянки, зони, об'єкти, цілі будівлі і їх частини, поверхи, на яких відповідального співробітника повинне проводити такі організаційні роботи.

Відповідальні особи зобов'язуються розробити, впровадити та підтримувати в певному інструкцією і положенням на ввірених їм об'єктах протипожежний режим і інструкції відповідно до вимог, викладених в нормативних актах.

Передбачено також створення підрозділу добровільної пожежної охорони та пожежно-рятувальної команди в його складі.

Встановлений режим включає порядки з описом місць спеціального призначення та правила їх користування та утримання, наприклад:

– евакуаційних шляхів;

– так званих «курилок»;

– місць складування продукції та сировини;

– стоянки транспорту.

Також встановлюється порядок роботи та технічного обслуговування:

– вентиляційного устаткування;

– засобів пожежогасіння і захисту від загорянь;

– нагрівальних приладів;

– електрообладнання.

Розробляються і впроваджуються правила роботи з відкритим вогнем і горючими матеріалами. Створюються графіки проходження інструктажів з пожежної безпеки співробітників, а також порядок і терміни перевірок знань пожежно-технічного мінімуму, в тому числі, тих працівників, які відповідальні за цю ділянку роботи на підприємстві. При цьому можуть передбачатися внутрішні лекції, семінари, тренінги та практичні заняття на підприємстві, а також зовнішні – на базі спеціалізованих навчальних центрів з професійними викладачами.

Важливою складовою протипожежного режиму на будь-якому об’єкті є розробка і впровадження порядку дій при виникненні пожежі. Неодмінно має бути план евакуації, описано, як повинні відключатися електроустановки, що і в якій послідовності необхідно робити співробітникам.

Відповідно, для кожного об’єкта, кожного приміщення (крім коридорів, санвузлів, басейнів і подібних приміщень), окремих видів робіт складаються інструкції, за якими повинен працювати персонал, залучений на певних ділянках і в виконанні окремих видів робіт. За інструкціями проводиться навчання (інструктаж) персоналу з подальшим контролем знань.

Детально про те, як розробити протипожежний режим, прописати порядки та інструкції, пояснюють на тематичних курсах і семінарах [2].

8.5 Розрахункова частина

Система освітлення робочого місця користувача ПК має відповідати наступним вимогам (рис. 8.1).

Рисунок 8.1 – Вимоги до системи освітлення робочого місця користувача ПК

Проведемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення, ширина якого складає 6,4 м, довжина – 8,3 м, висота – 3,2 м.

У зазначеному приміщенні працює 8 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F=E\cdot S\cdot K\cdot Z/n,$$

де: F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення.

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$i=S/(h\cdot (A+B)),$$

де:

S – площа приміщення, *S* = 53,1 *м²;*

h – розрахункова висота підвісу, *h* = 3,2 *м. (співпадає з в*исотою стелі, оскільки лампи освітлення закріплюються на стелі)*;*

A – ширина приміщення, *A = 6,4 м;*

B – довжина приміщення, *B* = 8,3 *м.*

Підставимо всі значення у формулу та визначимо індекс приміщення: i=1,1.

Знаючи індекс приміщення, за знаходимо *n = 0,46 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам) [8].* Підставимо всі значення у формулу, визначимо світловий потік: F=57161,7 Лм.

Для розрахунку будемо використовувати стельові *світлодіодні панелі* Призма-72 6400К, світловий потік яких $F_л$ = 7200 *Лм.*

Число ламп визначається по формулі:

$$N=F/F_л$$

де:

F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначимо індекс приміщення:

$$N= 57161,7/ 7200=7,9 \text{ шт.}$$

Приймаємо необхідну кількість *світлодіодних світильників* 8 шт.

Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи безпечної розробки при використанні технології Agile.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів безпечної розробки при використанні технології Agile.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем безпечної розробки при використанні технології Agile.

– Досліджена система безпечної розробки при використанні технології Agile.

– На основі отриманих результатів досліджень створена програмна реалізація системи безпечної розробки при використанні технології Agile.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання безпечної розробки при використанні технології Agile.

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.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ММВ.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розиєв А. Дослідження та програмна реалізація системи безпечної розробки при використанні технології Agile // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025.

2. Aaron Torres. Go Programming Cookbook Second Edition. Packt Publishing Ltd. 2019. 427 p.

3. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с.

4. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

5. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.

6. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.

7. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

8. Вінтенко, Б.Ю., Миронець, І.В., Смірнов, О.А., Коваленко, О.В., Усік, П.С., Буравченко, К.О., Лисенко, І.А. «Логіко-структурна модель комп'ютерно-орієнтованої процедури системи підтримки оперативного персоналу АЕС». *Кібербезпека: освіта, наука, техніка.* 2025. Том 2 № 30. С. 413-427, 2025.

9. Смірнова, Т.В. «Дослідження методів, моделей та сучасних ІТ-рішень для підтримки технологічних процесів у критичній інфраструктурі держави». *Кібербезпека: освіта, наука, техніка.* 2025. Том 2 № 30. С.195-208, 2025.

10. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп’ютерної інтелектуальної системи

підтримки оператора енергоблоку АЕС». *Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка»* (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.

11. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». *International Review on Modelling and Simulations* 18 (1), 2025. pp. 32-42.

12. Вінтенко Б.Ю., Смірнов О.А., Миронець І.В., Смірнова Т.В., Коваленко О.В., Мацуй А.М. «Модель шляхів отримання вхідних даних комп'ютерної інтелектуальної системи підтримки оперативного персоналу АЕС». *Центральноукраїнський науковий вісник. Технічні науки.* 2025. Вип. 11(42), ч. II. С.52-62.

13. Вінтенко Б.Ю., Смірнов О.А., Миронець І.В., Смірнова Т.В. «Методи забезпечення відмовостійкості інтелектуальних систем підтримки оператора». *VIII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології",* м. Кропивницький. 24-25 квітня 2025 р. – Кропивницький: ЦНТУ. – 2025. – С. 44-46.

14. Смірнов, О.А., Константинова, Л.В., Конопліцька-Слободенюк, О.К., Козірова, Н.В, Якименко, Н.М., Доренський, О.П., Буравченко, К.О. «Дослідження інструментів штучного інтелекту для роботи з базами даних та аналізу даних». *Кібербезпека: освіта, наука, техніка.* 2025. №3(27), С. 429–448.)

15. Smirnov O., Fedorov E., Neskorodieva A., Neskorodieva T. «Intellectual Classification method of Gymnastic Elements Based on Combinations of Descriptive and Generative Approache». *CEUR Workshop Proceedings* Volume 3664, 2024, Pages 11-23.

16. Вінтенко, Б., Миронець, І., Смірнов, О., Коваленко, А., Конопліцька-Слободенюк, О., Смірнова, Т., Константинова, Л. «Дослідження

застосування систем підтримки оперативного персоналу об'єкту критичної інфраструктури при керуванні енергоблоком АЕС з реактором типу ВВЕР-1000». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 6-26.

17. 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.

18. Kuznetsov O., Ilchenko O., Kryvinska N., Buravchenko K., Smirnov O., Savchenko Iu. «An Empirical Assessment of Leading Blockchain Financial Services». *2023 IEEE 1st Ukrainian Distributed Ledger Technology Forum (UADLTF),* Kyiv, Ukraine, 2023, pp. 1-6,

19. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

20. Malyukov V., Bebeshko B., Lakhno V., Smirnov O., Malyukova I., Mohylnyi H. «Managing the Purchase-Sale Process of Digital Currencies Under Fuzzy Conditions». *Lecture Notes in Networks and Systems,* 2023, 729 LNNS, pp. 104–112.

21. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

22. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А. «Дослідження нормативної документації та стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». VI міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 20-21 квітня 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 35-36.

29. 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.

30. 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.

31. 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

32. 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.

33. 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.

34. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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.

41. 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.

42. 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.

43. 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.

44. 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.

45. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

46. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

47. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

48. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

49. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

50. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.